139
66

ANALYSIS OF NETWORKS WITH DYNAMIC TOPOLOGIES

by

Robert L. Moose, Jr.

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirement for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science and Applications

APPROVED:

Richard E. Nance, Chairman

Donald C. S. Allison

Osman Balci

[illegible]

Ralph L. Disney

Robert Y. Foutz

December 1987

Blacksburg, Virginia

ANALYSIS OF NETWORKS WITH DYNAMIC TOPOLOGIES

by

Robert L. Moose, Jr.

Committee Chairman: Richard E. Nance
Computer Science

(ABSTRACT)

Dynamic hierarchical networks represent an architectural strategy for employing adaptive behavior in applications sensitive to highly variable external demands or uncertain internal conditions. The characteristics of such architectures are described, and the significance of adaptive capability is discussed. The necessity for assessing cost/benefit tradeoffs leads to the use of queueing network models. The general model, a network of $M/M/1$ queues in a random environment, is introduced and then is simplified so that the links may be treated as isolated $M/M/1$ queues in a random environment. This treatment yields a formula for approximate mean network delay by combining matrix-geometric results (mean queue length and mean delay) for the individual links. Conditions under which the analytic model is considered valid are identified through comparison with a discrete event simulation model. Last, performance of the dynamic hierarchy is compared with that of the static hierarchy. This comparison establishes conditions for which the dynamic architecture enables performance equal or nearly equal to performance of the static architecture.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Richard Nance for his continued support and guidance. I greatly appreciate his assistance at all levels throughout the course of this research. I also would like to thank Dr. Ralph Disney for the large number of classroom hours that I spent under his instruction and for many interesting discussions, not all of which were technical.

My lovely and wonderful wife, , deserves (and now receives) thanks for assisting with the typing of this dissertation. Karen Kaster created the pictures in Chapter 3.

Last, I gratefully acknowledge the support of this research by the United States Navy through the Systems Research Center of Virginia Tech.

TABLE OF CONTENTS

Chapter		page
	Abstract	ii
	Acknowledgements	iii
1.	Introduction	1
2.	Literature Review	4
	2.1. Early Analyses of Computer Networks	4
	2.2. Jackson Networks and Product Form	5
	2.3. Gordon-Newell Networks	7
	2.4. Baskett, Chandy, Muntz, and Palacios (BCMP) Networks	8
	2.5. Kelly Networks	10
	2.6. Network Sojourn Time and Overtaking	14
	2.7. Markov Renewal Network Analysis	15
	2.8. Queueing Systems in Random Environments	16
	2.9. Credibility Assessment, Verification, and Validation	18
3.	Dynamic Hierarchical Networks	20
	3.1. Description	20
	3.2. Significance and Motivations	25
	3.2.1. Application to the Physical Problem	25
	3.2.2. Theoretical Interest	26
	3.3. Previous Research	27
4.	Queueing Models	30
	4.1. Network Model	30
	4.2. Approximate Single Link Models	34
	4.2.1. Matrix-Geometric Results	35
	4.2.2. Instantaneous Reconfigurations	42
	4.2.3. Reconfiguration Periods	44
	4.2.4. Other Models	48
	4.3. Network Model with the Departure Process Assumption	48
	4.4. Mean Network Delay	51
5.	Simulation Model	54
	5.1. Model Structure	54
	5.2. Model Implementation	55
	5.3. Experimental Design	56
	5.4. Credibility Assessment	60
6.	Comparison of Analytic and Simulation Models	64
	6.1. Test Networks	64
	6.2. Comparative Results	66

Chapter		page
7.	Comparison of Dynamic and Static Hierarchies	74
	7.1. Test Networks	74
	7.2. Comparative Results	75
8.	Conclusions and Future Research	78
	References	82
Appendix A.	Network Data for Transient Period Length Determination	91
Appendix B.	Network Data for Execution Trace	102
Appendix C.	Network Data for Stress Testing	105
Appendix D.	Network Topologies and Parameter Sets for Analytic and Simulation Model Comparison	108
	D.1. Network 1	108
	D.2. Network 2	112
	D.3. Network 3	116
	D.4. Network 4	121
	D.5. Network 5	126
	D.6. Network 6	132
	D.7. Network 7	136
	D.8. Network 8	141
	D.9. Network 9	150
	D.10. Network 10	159
Appendix E.	Comparative Results for Individual Links	169
Appendix F.	Network Topologies and Parameter Sets for Dynamic and Static Hierarchy Comparison	182
	F.1. Network 1	182
	F.2. Network 2	183
	F.3. Network 3	184
	F.4. Network 4	185
	F.5. Network 5	187
	F.7. Network 7	189
	F.8. Network 8	191
	F.9. Network 9	194
	F.10. Network 10	198
Appendix G.	Simscript II.5 Dynamic Hierarchy Simulation Source Code	205
Appendix H.	Dynamic Hierarchy Matrix-Geometric Analysis Source Code	224

LIST OF TABLES

Table		page
1.	Observed Times to Reach Steady State	59
2.	Event Times for 1,3-Messages	61
3.	Event Times for 3,1-Messages	62
4.	Network Performance Under Stress Testing	63
5.	Representative Link Utilization Levels	65
6.	Comparison of Analytic and Simulation Models (Mean Network Delay)	68
7.	Comparison of Dynamic and Static Hierarchy	76
A.1.	Link Numbers and Capacities— Pilot Runs 1 to 3	92
A.2	Stream Values— Pilot Runs 1 to 3	92
A.3.	Link Numbers and Capacities— Pilot Runs 4 to 6	93
A.4	Stream Values— Pilot Runs 4 to 6	93
B.1.	Link Numbers and Capacities— Execution Trace	102
C.1.	Link Numbers and Capacities— Stress Test	105
D.1.	Link Numbers and Capacities— Network 1, Parameter Set 1	109
D.2.	Link Numbers and Capacities— Network 1, Parameter Set 2	110
D.3.	Link Numbers and Capacities— Network 1, Parameter Set 3	111
D.4.	Link Numbers and Capacities— Network 2, Parameter Set 1	113
D.5.	Link Numbers and Capacities— Network 2, Parameter Set 2	114
D.6.	Link Numbers and Capacities— Network 2, Parameter Set 3	115
D.7.	Link Numbers and Capacities— Network 3, Parameter Set 1	117
D.8.	Link Numbers and Capacities— Network 3, Parameter Set 2	118
D.9.	Link Numbers and Capacities— Network 3, Parameter Set 3	119
D.10.	Link Numbers and Capacities— Network 4, Parameter Set 1	122
D.11.	Link Numbers and Capacities— Network 4, Parameter Set 2	123
D.12.	Link Numbers and Capacities— Network 4, Parameter Set 3	124
D.13.	Link Numbers and Capacities— Network 5, Parameter Set 1	127
D.14.	Link Numbers and Capacities— Network 5, Parameter Set 2	128
D.15.	Link Numbers and Capacities— Network 5, Parameter Set 3	130
D.16.	Link Numbers and Capacities— Network 6, Parameter Set 1	133
D.17.	Link Numbers and Capacities— Network 6, Parameter Set 2	134
D.18.	Link Numbers and Capacities— Network 6, Parameter Set 3	135
D.19.	Link Numbers and Capacities— Network 7, Parameter Set 1	137
D.20.	Link Numbers and Capacities— Network 7, Parameter Set 2	138
D.21.	Link Numbers and Capacities— Network 7, Parameter Set 3	139
D.22.	Link Numbers and Capacities— Network 8, Parameter Set 1	142
D.23.	Link Numbers and Capacities— Network 8, Parameter Set 2	143
D.24.	Link Numbers and Capacities— Network 8, Parameter Set 3	144
D.25.	Link Numbers and Capacities— Network 8, Parameter Set 4	146
D.26.	Link Numbers and Capacities— Network 8, Parameter Set 5	147
D.27.	Link Numbers and Capacities— Network 8, Parameter Set 6	148
D.28.	Link Numbers and Capacities— Network 9, Parameter Set 1	151
D.29.	Link Numbers and Capacities— Network 9, Parameter Set 2	153
D.30.	Link Numbers and Capacities— Network 9, Parameter Set 3	155

Table		page
D.31.	Link Numbers and Capacities— Network 9, Parameter Set 4	157
D.32.	Link Numbers and Capacities— Network 10, Parameter Set 1	161
D.33.	Link Numbers and Capacities— Network 10, Parameter Set 2	163
D.34.	Link Numbers and Capacities— Network 10, Parameter Set 3	165
D.35.	Link Numbers and Capacities— Network 10, Parameter Set 4	167
E.1.	Comparative Results for Links— Network 1, Parameter Set 1	169
E.2.	Comparative Results for Links— Network 1, Parameter Set 2	169
E.3.	Comparative Results for Links— Network 1, Parameter Set 3	169
E.4.	Comparative Results for Links— Network 2, Parameter Set 1	170
E.5.	Comparative Results for Links— Network 2, Parameter Set 2	170
E.6.	Comparative Results for Links— Network 2, Parameter Set 3	170
E.7.	Comparative Results for Links— Network 3, Parameter Set 1	171
E.8.	Comparative Results for Links— Network 3, Parameter Set 2	171
E.9.	Comparative Results for Links— Network 3, Parameter Set 3	171
E.10.	Comparative Results for Links— Network 4, Parameter Set 1	172
E.11.	Comparative Results for Links— Network 4, Parameter Set 2	172
E.12.	Comparative Results for Links— Network 4, Parameter Set 3	172
E.13.	Comparative Results for Links— Network 5, Parameter Set 1	173
E.14.	Comparative Results for Links— Network 5, Parameter Set 2	173
E.15.	Comparative Results for Links— Network 5, Parameter Set 3	173
E.16.	Comparative Results for Links— Network 6, Parameter Set 1	174
E.17.	Comparative Results for Links— Network 6, Parameter Set 2	174
E.18.	Comparative Results for Links— Network 6, Parameter Set 3	174
E.19.	Comparative Results for Links— Network 7, Parameter Set 1	175
E.20.	Comparative Results for Links— Network 7, Parameter Set 2	175
E.21.	Comparative Results for Links— Network 7, Parameter Set 3	175
E.22.	Comparative Results for Links— Network 8, Parameter Set 1	176
E.23.	Comparative Results for Links— Network 8, Parameter Set 2	176
E.24.	Comparative Results for Links— Network 8, Parameter Set 3	176
E.25.	Comparative Results for Links— Network 8, Parameter Set 4	177
E.26.	Comparative Results for Links— Network 8, Parameter Set 5	177
E.27.	Comparative Results for Links— Network 8, Parameter Set 6	177
E.28.	Comparative Results for Links— Network 9, Parameter Set 1	178
E.29.	Comparative Results for Links— Network 9, Parameter Set 2	178
E.30.	Comparative Results for Links— Network 9, Parameter Set 3	179
E.31.	Comparative Results for Links— Network 9, Parameter Set 4	179
E.32.	Comparative Results for Links— Network 10, Parameter Set 1	180
E.33.	Comparative Results for Links— Network 10, Parameter Set 2	180
E.34.	Comparative Results for Links— Network 10, Parameter Set 3	181
E.35.	Comparative Results for Links— Network 10, Parameter Set 4	181
F.1.	Link Numbers and Capacities— Network 1, Parameter Set 3	182
F.2.	Link Numbers and Capacities— Network 2, Parameter Set 3	183
F.3.	Link Numbers and Capacities— Network 3, Parameter Set 3	184
F.4.	Link Numbers and Capacities— Network 4, Parameter Set 3	185
F.5.	Link Numbers and Capacities— Network 5, Parameter Set 3	187
F.6.	Link Numbers and Capacities— Network 7, Parameter Set 2	189

Table		page
F.7.	Link Numbers and Capacities— Network 8, Parameter Set 2	191
F.8.	Link Numbers and Capacities— Network 8, Parameter Set 3	192
F.9.	Link Numbers and Capacities— Network 8, Parameter Set 6	193
F.10.	Link Numbers and Capacities— Network 9, Parameter Set 2	195
F.11.	Link Numbers and Capacities— Network 9, Parameter Set 3	196
F.12.	Link Numbers and Capacities— Network 9, Parameter Set 4	197
F.13.	Link Numbers and Capacities— Network 10, Parameter Set 1	199
F.14.	Link Numbers and Capacities— Network 10, Parameter Set 2	200
F.15.	Link Numbers and Capacities— Network 10, Parameter Set 3	202
F.16.	Link Numbers and Capacities— Network 10, Parameter Set 4	203

LIST OF FIGURES

Figure		page
1.	Example Dynamic Hierarchy with a Physically Varying Topology	22
2.	Example Dynamic Hierarchy with a Logically Varying Topology	23
3.	Example Bus-Based Dynamic Hierarchy	24
4.	Moving Average Graphs For Determination of Transient Period Length	58
A.1.	Network for Transient Period Determination	91
A.2	Moving Average of Mean Queue Length, Queue 1: Pilot Runs 1 to 3	94
A.3	Moving Average of Mean Queue Length, Queue 1: Pilot Runs 4 to 6	94
A.4	Moving Average of Mean Queue Length, Queue 2: Pilot Runs 1 to 3	95
A.5	Moving Average of Mean Queue Length, Queue 2: Pilot Runs 4 to 6	95
A.6	Moving Average of Mean Queue Length, Queue 3: Pilot Runs 1 to 3	96
A.7	Moving Average of Mean Queue Length, Queue 3: Pilot Runs 4 to 6	96
A.8	Moving Average of Mean Queue Length, Queue 4: Pilot Runs 1 to 3	97
A.9	Moving Average of Mean Queue Length, Queue 4: Pilot Runs 4 to 6	97
A.10	Moving Average of Mean Link Delay, Queue 1: Pilot Runs 1 to 3	98
A.11	Moving Average of Mean Link Delay, Queue 1: Pilot Runs 4 to 6	98
A.12	Moving Average of Mean Link Delay, Queue 2: Pilot Runs 1 to 3	99
A.13	Moving Average of Mean Link Delay, Queue 2: Pilot Runs 4 to 6	99
A.14	Moving Average of Mean Link Delay, Queue 3: Pilot Runs 1 to 3	100
A.15	Moving Average of Mean Link Delay, Queue 3: Pilot Runs 4 to 6	100
A.16	Moving Average of Mean Link Delay, Queue 4: Pilot Runs 1 to 3	101
A.17	Moving Average of Mean Link Delay, Queue 4: Pilot Runs 4 to 6	101
B.1.	Network for Execution Trace	102
C.1.	Network for Stress Testing	105
D.1.	Dynamic Hierarchy 1 Topology	108
D.2.	Dynamic Hierarchy 2 Topology	112
D.3.	Dynamic Hierarchy 3 Topology	116
D.4.	Dynamic Hierarchy 4 Topology	121
D.5.	Dynamic Hierarchy 5 Topology	126
D.6.	Dynamic Hierarchy 6 Topology	132
D.7.	Dynamic Hierarchy 7 Topology	136
D.8.	Dynamic Hierarchy 8 Topology	141
D.9.	Dynamic Hierarchy 9 Topology	150
D.10(a).	Dynamic Hierarchy 10 Topology	159
D.10(b).	Dynamic Hierarchy 10 Topology (continued)	160
F.1.	Static Hierarchy 1 Topology	182

Figure page

F.2. Static Hierarchy 2 Topology 183
F.3. Static Hierarchy 3 Topology 184
F.4. Static Hierarchy 4 Topology 185
F.5. Static Hierarchy 5 Topology 187
F.6. Static Hierarchy 7 Topology 189
F.7. Static Hierarchy 8 Topology 191
F.8. Static Hierarchy 9 Topology 194
F.9. Static Hierarchy 10 Topology 198
G.1 File Format 1— Dynamic Hierarchy Simulation 206
H.1 File Format 2— Link Arrival Rate Computation 226
H.2 File Format 3— Matrix-Geometric Mean Queue Length and Mean Link Delay Computation 226
H.3 File Format 4— Dynamic Hierarchy Network Delay Computation 227
H.4 File Format 5— Square-Root Capacity Assignment and Static Hierarchy Network Delay Computation 227

CHAPTER 1

INTRODUCTION

This research addresses the analysis of a class of computer communication networks whose members are referred to as *dynamic hierarchical networks*, or *dynamic hierarchies*. The dynamic hierarchy is an architectural concept that represents a generalization of the conventional tree structured architecture in which the network operates under a centralized, hierarchical mode of control. An overriding characteristic of these conventional (static) hierarchies is that, at the root of a tree-structured topology, there exists a single node that exercises primary control. Secondary capabilities filter down through the remainder of the network in a hierarchical manner. With this basic characteristic in force at all times, if the topology is allowed to vary, the resultant network is a dynamic hierarchical network.

The dynamic hierarchy is intended for applications in which it is possible and desirable to distinguish among multiple *scenarios*, external situations or sets of internal conditions. Each configuration (state of the topology) and apex node of the network corresponds to a different scenario. Configuration changes coincide with changes in the scenario.

The scenario and configuration can be considered dependent on the state of an *environment*. This view leads to a model of the dynamic hierarchy as a network of $M/M/1$ queues in a random environment. Simplification of this model enables the links (servers) to be treated separately as $M/M/1$ queues in a random environment. Composition of the results for the individual links yields a characterization of network performance. The primary steps in this research are definition of the general and simplified analytic models, validation of the simplified model by comparison with a simulation model, and use of the simplified model to compare the dynamic hierarchy with the conventional, static hierarchy.

The remainder of this dissertation is organized as follows: Chapter 2 reviews the literature on queueing networks, queues in random environments, and credibility assessment, verification, and validation of simulation models. (Discussion of the latter area is brief and is intended to identify some recent papers from the literature and sources of additional information).

Chapter 3 describes the dynamic hierarchy in more detail, discusses the significance and motivation of the concept and its analysis, and summarizes previous research on the dynamic hierarchy capacity assignment problem.

Chapter 4 first defines the general dynamic hierarchy model, a network of $M/M/1$ queues in a random, environment. Then, the necessary background on matrix-geometric methods (from [NEUM81b]) are provided and two models of transmission links in the dynamic hierarchy are defined. The network model subsequently is simplified and a formula for mean network delay is derived.

Chapter 5 describes the discrete event simulation component of this research. The model structure, model implementation in SIMSCRIPT, and experimental design are discussed. Additionally, five aspects of this modeling effort that support the credibility of the simulation model are discussed.

Chapter 6 compares the analytic and simulation models. The analytic model is validated for a set of conditions, whose boundary is characterized by the observations and conclusions contained in this chapter.

In Chapter 7, the goal of an analysis of relative costs and benefits of the dynamic hierarchy is addressed by comparing performance of the dynamic hierarchy with that of the static hierarchy.

CHAPTER 2

LITERATURE REVIEW

The approach taken in the modeling and analysis of the dynamic hierarchy is influenced by various ideas and results from the computer network and queueing network literature. Analysis and synthesis of computer-communication networks through the use of queueing network models is rooted in early work on the link capacity assignment problem and subsequent extensions. This work relies on the assumption of a product form network to enable a decomposition and the resultant simplified analysis.

2.1. Early Analyses of Computer Networks

The earliest work in this area is that of Kleinrock [KLEL64]. His modeling assumptions include the ***independence assumption,*** under which the length of a message is reassigned according to an exponential distribution at each link in its path. The resultant model is of the Jackson variety and its validity is established through simulation studies [KLEL64,KLEL76]. Consequently, the product form, joint queue length distribution † leads to relatively simple, closed form expressions for mean delay (sojourn time) at the individual links and mean delay of messages through the network.

The expressions enable link capacities to be optimized in a straightforward manner, yielding the ***square-root*** assignment strategy.

Extensions and variants of Kleinrock's model appear in [KLEL70,KLEL73,KLEL76], [MEIB71a,MEIB71b,MEIB72], [GERM77], and [KOMM78]. The capacity assignment problem as well as other topics in computer network design, analysis, and operation are

† See, for example, the product form, joint queue length distributions for the Jackson and Kelly networks.

covered by the texts [SCHM77,SCHM87] and [TANS81].

2.2. Jackson Networks and Product Form

The most important aspect leading to the success of Kleinrock (and related) work is the tractability achieved through a product form distribution for the joint queue length process. This product form enables an approach to the analysis that may be characterized by the steps: separate, analyze, combine. First the network is decomposed (separated) into a collection of queues that may be treated as if they were isolated single servers. Then these queues are analyzed individually. Lastly, the individual results are combined to produce an expression for some network performance measure.

The tractability achievable through product form networks, coupled with their wide applicability, has contributed to the extensive analysis reported in the literature. The first work dealing with arbitrarily interconnected networks is that of Jackson [JACJ57,JACJ63]. The queueing systems investigated therein have become known as *Jackson networks.* In [JACJ57], the M node (called *departments* by Jackson) network is characterized as follows:

(1) Each node may contain multiple, identical servers.

(2) External arrivals to node m occur according to a Poisson process with rate λ_m.

(3) The queuing discipline at each node is first-come-first-served.

(4) Service times at a node are independent, identically distributed (IID) exponential random variables.

(5) Movement of customers from node to node (or to outside the network) follows a specified set of routing probabilities.

The main result is that the joint network (stationary) queue length distribution has a product form. That is, if P_k^m is the queue length distribution for a node m viewed as if it were in isolation, then the network queue length distribution is

$$P(k_1, k_2, \ldots, k_m) = P_{k_1}^1 P_{k_2}^2 \cdots P_{k_m}^m,$$

where the arrival rates to the nodes are determined by the *traffic equations* (in the current terminology).

In [JACJ63], Jackson extends his work to allow the rate of the Poisson arrival process to the network to vary with the total number of customers in the network and the service rate for each queue to vary with the number of customers at that queue. Additionally, he characterizes the idea of (random walk) customer *routings* in more detail. The joint network queue length distribution again is of product form as stated in the following main result:

If $\pi = [\sum_{K=0}^{\infty} W(K)T(K)]^{-1} > 0$, then the stationary network queue length distribution is given by

$$p(\bar{k}) = \pi w(\bar{k}) W(S(\bar{k})),$$

where $\bar{k} = (k_1, k_2, \ldots, k_n)$ is a (queue length process) state vector,

$$S(\bar{k}) = \sum_{n=1}^{N} k_n,$$

$$W(K) = \prod_{i=1}^{K-1} \lambda(i),$$

$$T(K) = \sum_{\bar{k}:\, S(\bar{k})=K} w(\bar{k}), \text{ and}$$

$$w(\bar{k}) = \prod_{n=1}^{N} \prod_{i=1}^{k_n} \frac{e(n)}{\mu(n,i)}$$

is the product form term. The remaining notation (λ, μ, and e) is defined in Section 2 of [JACJ63].

2.3. Gordon-Newell Networks

The systems investigated by Jackson have become known as *open networks* . These systems are *open* in the sense that customers (from an infinite population) arrive from the outside, pass through the network, and then depart to the outside. Consequently, the total number of customers in the network varies. In a *closed network* , as discussed by Gordon and Newell in [GORW67], neither arrivals to nor departures from the network occur. The number of customers remains fixed (and finite) and these customers continually circulate, queueing and receiving service.

Specifically, in [GORW67] it is assumed that

(1) N customers circulate among M nodes with p_{ij} the probability of movement from node i to node j following a service completion.

(2) Node i contains r_i servers and service times there (at any server) are IID, exponential random variables with mean μ_i^{-1} .

A state vector for the network has the form $(n_1, n_2, \ldots, n_M)$, where $n_i \leq N$ is the number of customers at node i . Since $\sum_{i=1}^{M} n_i = N$, the queue lengths are not independent and this state descriptor can be given as, for example, $(n_2, n_3, \ldots, n_M)$. Define $x_1, x_2, \ldots, x_M$ as a positive solution of the system of equations

$$\sum_{i=1}^{M} p_{ik}(\mu_i x_i) = \mu_k x_k, \quad k = 1, 2, \ldots, M.$$

Additionally,

$$\alpha_k(n_k) = \begin{cases} n_k, & n_k \leq r_k, \\ r_k, & n_k \geq r_k, \end{cases} \text{ and}$$

$$\beta_k(n) = \prod_{j=1}^{n} \alpha_k(j).$$

Then, the network queue length probabilities are

$$P(n_1, n_2, \ldots, n_M) = \prod_{i=1}^{M} \frac{x_i^{n_i}}{\beta_i(n_i)} G^{-1}(N),$$

where $G(N) = \sum_{\bar{n}:\ \sum_{j=1}^{M} n_j = N} \prod_{i=1}^{M} \frac{x_1^{n_i}}{\beta_i(n_i)}$ is the *normalization constant.*

2.4. Baskett, Chandy, Muntz, and Palacios (BCMP) Networks

In [BASF75], Basket, et al. combine and extend the systems and results of Jackson and Gordon and Newell. They consider queueing networks with different classes of customers and in which the network may be open or closed with respect to each customer class. A network that is open with respect to some classes and closed with respect to others is said to be *mixed*.

The network is described informally as follows:

- The number of nodes is N.
- The number of customer classes is R.
- A class r customer completing service at node i moves to node j and becomes a class s customer (s may equal r) with probability $P_{i,r;j,s}$.
- For an open system, the Poisson arrival rate may depend on the number of customers in the network. For a mixed system, each class r for which the network is open has a Poisson arrival process whose rate may depend on the number of class r customers present.
- Service disciplines considered are:

(1) Multi-server, first-come-first-served with state dependent, exponential ser-

vice rate.

(2) Single server, processor sharing with service times whose distribution has a rational Laplace transform.

(3) Infinite server, class dependent service time distributions with rational Laplace transforms.

(4) Single server, preemptive resume, last-come-first-served, class dependent service time distributions with rational Laplace transforms.

Network states are described by a vector $x = (x_1, x_2, \ldots, x_N)$. The form of x_i, the vector describing the state of node i, depends on the type (1-4, as previously described) of node i. For example, at an FCFS (type 1) node i, $x_i = (x_{i1}, x_{i2}, \ldots, x_{in_i})$, where x_{ij} is the class of the *jth* customer and n_i is the number of customers at node i.

One interpretation of the matrix $P = \{P_{i,r;j,s}\}$ of customer switching probabilities is that it defines a reducible Markov chain with states (i,r) and irreducible subchains $E_1, E_2, \ldots, E_m$. Each subchain is either open or closed (in the same sense that the network is open or closed with respect to customer classes). Arrival rates $\{e_{ir}\}$ are determined through traffic equations for each subchain.

The network state distribution is given in the following theorem [BASF75, pp. 253-254]:

For a network of [nodes] which is open, closed, or mixed in which each [node] is of type 1, 2, 3, or 4, the equilibrium state probabilities are given by

$$P(S = x_1, x_2, \ldots, x_N) = Cd(S) f_1(x_1) f_2(x_2) \cdots f_N(x_N),$$

where C is a normalizing constant chosen to make the equilibrium state probabilities sum to 1, $d(S)$ is a function of the number of customers in the system, and each f_i is a function that depends on the type of [node] i.

If [node] i is of type 1, then $f_i(x_i) = (1/\mu_i)^{n_i} \prod_{j=1}^{n_i} e_{ix_{ij}}$

. . .

The network queue length distribution is derived by considering aggregate states $y = (y_1, y_2, \ldots, y_N)$, where $y_i = (n_{i1}, n_{i2}, \ldots, n_{ir})$ and n_{ir} is the number of class r customers at node i. The resultant probabilities are [BASF75, p. 254]

$$P(S = (y_1, y_2, \ldots, y_N)) = Cd(S)g_1(y_1)g_2(y_2) \cdots g_N(y_N),$$

where

if [node] i is of type 1, then $g_i(y_i) = n!\{\prod_{r=1}^{R}(1/n_{ir}!)[e_{ir}]^{n_{ir}}\}(1/\mu_i)^{n_i}$

. . .;

2.5. Kelly Networks

Kelly [KELF75,KELF76,KELF79,KELF82] analyzes a class of networks whose generality is comparable to that of Basket, et. al. Kelly's treatment is more systematic, employing properties of reversed and reversible processes to establish results for the networks of interest. The model in [KELF75] is defined as follows:

- The network contains J queues (nodes).
- Customers may be of I types.
- The network is either open (for all customer types) or closed (for all customer types).
- Type i customers completing service at node j move to node k with relative rate ("probability intensity") $\lambda_{jk}(i)$.
- For node j, $\phi_j(n_j)$ is the total service effort when n_j customers are present, $\gamma_j(l, n_j)$ is the proportion of this effort directed to the customer in position l, $\delta_j(m, n_j + 1)$ is the probability that a customer arriving to find n_j customers at

node j joins the queue in position m.

- Service requirements are exponentially distributed with mean 1 (one).

The state descriptor for this system is $C = (c_1, c_2, \ldots, c_J)$, where $c_j = (c_j(1), c_j(2), \ldots, c_j(n_j))$ is the state of node j with n_j customers and $c_j(l)$ the type of the customer in position l.

For a closed network, with state space L, let

$$A_j(c_j) = \prod_{l=1}^{n_j} \frac{\alpha_j(c_j(l))}{\phi_j(l)} \quad \text{for } n_j \geq 1$$

and $A_j(c_j) = 1$ for $n_j = 0$. Then the network state distribution is

$$P(C) = b \prod_{j=1}^{J} A_j(c_j) \quad \text{for all } C \in L,$$

where b is a normalization constant and $(\alpha_1(i), \alpha_2(i), \ldots, \alpha_J(i))$ is a solution of the traffic equations for customer type i.

For an open network, let $\nu_k(i)$ be the rate of Poisson arrivals of type i customers to node k from outside the network and $\mu_j(i)$ be the relative rate at which type i customers completing service at node k depart the network. Then, if the stability condition holds for each node, letting

$$A_j(c_j) = \prod_{l=1}^{n_j} \frac{\alpha_j(c_j(l))}{\phi_j(l)} \quad \text{for } n_j \geq 1$$

and $A_j(c_j) = 1$ for $n_j = 0$, the network state distribution is

$$P(C) = b \prod_{j=1}^{J} A_j(c_j) \quad \text{for all } C \in L.$$

L now denotes the state space of the open network. Again b is a normalization constant and the $\alpha_j(i)$ are the solution of the (open network) traffic equations.

This system may be modified in various ways to allow different characteristics, such as zero service effort in some cases and mean service time dependent on customer type.

In [KELF76] Kelly addresses similar problems for a different type of system. Subsequently Kelly ([KELF79]) refers to the previously described systems as *migration processes* and the current systems as *networks of queues*. The basic network definition is almost the same as that of [KELF75]. The primary difference is that customer movement now is governed by *routes* instead of probability intensities. Customers of type i follow the route $r(i,1)$, $r(i,2)$, . . . , $r(i,S(i))$, where $r(i,j)$ is the jth node in the route and $S(i)$ is the number of nodes in the route.

As a result of the use of fixed routes, the network state descriptor for this open network is defined with additional information as follows [KELF76, p. 421]:

> Let $t_j(l)$ be the type of the customer in position l in queue j and let $s_j(l)$ be the stage along his route that this customer has reached. Let $c_j(l) = (t_j(l), s_j(l))$. Then
>
> $$c_j = (c_j(1), c_j(2), \ldots, c_j(n_j))$$
>
> describes the state of queue j and
>
> $$C = (c_1, c_2, \ldots, c_J)$$
>
> is a Markov process representing the state of the system.

Characteristics of the individual queues (exponential, mean 1 (one) service requirements, ϕ_j, γ_j, δ_j) are identical to those of the migration process.

If this system is stable, the network state distribution is given by

$$\pi(C) = b \prod_{j=1}^{J} A_j(c_j)$$

where

$$A_j(c_j) = b \prod_{l=1}^{n_j} \frac{\alpha_j(t_j(l), s_j(l))}{\phi_j(l)},$$

where b is a normalization constant and

$$\alpha_j(i, s) = \begin{cases} \nu(i) & \text{if } r(i, s) = j, \\ 0 & \text{otherwise.} \end{cases}$$

Further (Corollary 2 [KELF76, p. 423]), the "state of [each queue] is independent of the state of the rest of the system" and the marginal queue length distribution at each queue is proportional to that of an $M/M/$-type queue, with an appropriately defined arrival rate, in isolation.

Generalizations of this model are:

- Arbitrarily distributed (IID) service requirements (See [KELF76, pp. 426-427] and the supporting result of Barbour [BARA76]).
- Poisson arrival rates that depend on the number of customers in the network.

Further generalizations and variants of network of queues are discussed in Kelly's text [KELF79] and in [KELF82]. The introduction of networks of *quasi-reversible* queues represents an important extension. A queue is called quasi-reversible

> [If] its state $x(t)$ is a stationary Markov process with the property that the state of the queue at time t_0, $x(t_0)$, is independent of:
> (i) the arrival times of class c customers, $c \in C$, subsequent to time t_0;
> (ii) the departure times of class c customers, $c \in C$, prior to time t_0 [KELF79, pp. 65-66].

The following result holds [KELF79] (Theorem 3.6):

> If a queue is quasi-reversible then:
> (i) arrival times of class c customers, for $c \in C$, form independent Poisson processes;
> (ii) departure times of class c customers, for $c \in C$, form independent Poisson processes.

A network can be defined in which the behavior and transition rates are such that each queue j would be a quasi-reversible queue if it were in isolation. Such a network in equilibrium is called a network of quasi-reversible queues. Let $X = (x_1, x_2, \ldots, x_J)$ be the network process, where x_j is the state of queue j. Then the stationary distribution of the network state is

$$\pi(x_1, x_2, \ldots, x_J) = \pi_1(x_1)\pi_2(x_2) \cdots \pi_J(x_J),$$

where $\pi_j(x_j)$ is the stationary distribution queue j would have if it were in isolation.

The *symmetric queue* is an important special case of the quasi-reversible queue. Behavior of a symmetric queue is similar to that of the previously discussed queues with the additional (symmetry) condition that $\gamma = \delta$. Furthermore, Kelly shows that these queues remain quasi-reversible when service requirements have general, class dependent distributions. Additional extensions address customer class changes and more general arrival processes.

Closed networks of quasi-reversible queues are treated similarly. The corresponding stationary distribution is

$$\pi(x_1, x_2, \ldots, x_J) = B\pi_1(x_1)\pi_2(x_2) \cdots \pi_J(x_J),$$

where B is a normalization constant and $\pi_j(x_j)$ is the stationary distribution for queue j considered in isolation. In the general case, customer class changing is allowed and the normalization constant has a slightly different form.

2.6. Network Sojourn Time and Overtaking

Results on the sojourn time problem in queueing networks are not as general as those just discussed for the network queue length problem. An early attempt to solve the sojourn time problem, for acyclic Jackson networks, is that of Lemoine [LEMA77]. However, it is pointed out by Mitrani [MITI79] and proved by Simon and Foley [SIMB79] that Lemoine's result is not correct with the claimed generality. In [LEMA79] Lemoine amends his result, the Laplace transform for network sojourn time, to apply only to "an acyclic Jackson network in which every two nodes are connected by at most one directed path (e.g., if the network is a tree)."

Subsequent work concentrates primarily on networks and paths within them that possess the *non-overtaking* property. A path is non-overtaking if "a customer traveling along the path cannot be overtaken by a subsequent arrival or indirectly by the effects of subsequent arrivals" [WALJ80 p. 1000]. Walrand and Varaiya [WALJ80] show that "the sojourn times of a customer at the various nodes of a non-overtaking path [in a Jackson network] are all mutually independent." Daduna [DADH82] derives the Laplace-Stieltjes transform for sojourn time on a non-overtaking path in a Gordon-Newell network with multiple customer classes. Melamed [MELB82] considers non-overtaking paths in open, Kelly-type networks with multiple customer classes, class changes, class dependent routing, and class dependent service characteristics. He determines the Laplace transform for the sojourn time of a customer on such paths (in steady state), thus showing that the sojourn time is the sum of the exponential sojourn times experienced by the customer at the queues in the path. The focus of Kelly and Pollet in [KELF83] is non-overtaking paths in closed networks of FCFS, single server queues with multiple customer classes, class dependent routing, customer class changes, and exponential service requirements. They derive the Laplace transform of the joint distribution of sojourn times at the nodes in a non-overtaking section of a route.

2.7. Markov Renewal Network Analysis

Another important body of work is that of Disney and others on the characterization and analysis of various processes, especially flow processes, in queueing networks.

Much of this work is covered in the papers [DISR75,DISR81] by Disney, [DISR85] by Disney and König, and the text [DISR87] by Disney and Keissler. The primary approach taken (see [DISR87, p. xiv]) is to '[establish] the probability structure of a random process of interest . . ., e.g., that it is a Markov renewal process', establish 'a result that is used to obtain all finite-dimensional joint probability distributions for the process characterized, e.g., the semi-Markov kernel', and then 'explore properties of [the process] in depth.' Systems and processes of interest include queues with feedback, finite capacity queues with overflow, and flow processes in Markov and Jackson networks.

2.8. Queueing Systems in Random Environments

The first paper appearing in the open literature on $M/M/1$ queues in random environments (which are referred to here as *RE-queues*) seems to be that of Eisen and Tainiter [EISM63]. They assume a two state environment process and derive generating functions and first moments for queue length and waiting (sojourn) time.

A portion of the doctoral thesis of Scott [SCOM64] is devoted to the analysis of a variant of the model introduced by Eisen and Tainiter. Scott considers the case where the arrival rate varies with the environment and the service rate remains constant. He formulates a model with an $N+1$ state environment process but restricts his derivations to the case of $N=1$. The use of Laplace transforms and generating functions dominates the analysis of interarrival times and 'number of arrivals'. Scott also assumes that 'the length of the next inter-arrival time depends only on the arrival rate at its start' [SCOM64, p. 99]. This assumption differs from that of Eisen and Tainiter model, in which rate and distribution changes coincide with environment state changes.

The work of Yechiali and Naor [YECU71] essentially reformulates the Eisen and

Tainiter model and derives the generating function and mean of the queue length distribution. (This work and that of Scott appear to have been done without knowledge of the paper by Eisen and Tainiter.) In [YECU73], Yechiali generalizes the model to allow rates that vary with both the environment and the queue length as well as an environment process with an arbitrary (finite) number of states. However, his analysis of the queue length distribution, which again is carried out through the use of generating functions, is restricted to the case where the rates depend only on the state of the environment process.

Neuts' first contributions to this area are reported in his paper on a variant of the $M/G/1$ queue in a random environment [NEUM69]. He assumes that arrival rate and distribution changes coincide with environment state changes but that the service time of a customer is governed by the service time distribution in effect at the time the customer enters service. The service time distributions are general (with finite means). Additionally, the environment process may have an arbitrary (finite) number of states. Neuts employs matrix techniques, generating functions, and Laplace transforms to analyze various aspects of this system, including the arrival process, the busy cycle, and queue lengths.

In later works [NEUM77a, NEUM78b], Neuts investigates in detail the direct generalization (with an m state environment, $2 \leq m \leq \infty$) of the Eisen and Tainiter RE-queue. He exploits the special structure of the transition matrix to derive, through the use of matrix methods, the *matrix-geometric* queue length distribution as well as a number of other results. The model is generalized in [NEUM78b] to allow a multiserver queue. Related publications by Neuts include [NEUM77b, NEUM78a, NEUM80, NEUM81a] and the comprehensive text [NEUM81b] on *phase-type* distributions and stochastic processes with matrix-geometric solutions.

Various results established by Purdue are reported in [PURP74,PURP78]. Through the use of matrix and transform techniques, he concentrates on the *arrival-service function* (a conditional, joint distribution which involves the arrival, service, and environment processes) and the busy period.

Additional literature relating to the RE-queue and its analysis includes that of Evans [EVAR64,EVAR67], the Ph.D. Thesis by Wallace [WALV69], which introduces the *quasi birth and death* processes, Ramaswami [RAMV80], Miller [MILD81], Purdue and Linton [PURP81], and Linton and Purdue [LIND82].

2.9. Credibility Assessment, Verification, and Validation

Although the method of cross-validating an analytic model by comparison with a simulation model has received little direct attention in the literature, model validation and credibility assessment have been discussed extensively. Shannon [SHAR75], Law and Kelton [LAWA82], and Banks and Carson [BANJ84] agree that the steps of the validation process should include development of a model with high face validity, testing of the model assumptions, and comparison of model output (input-output transformations) with real system output (input-output transformations). Additionally, Law and Kelton suggest statistical validation procedures.

Possible steps in the verification process include modular development and debugging of the simulation program, examination of the code by individuals other than the author(s), execution tracing and tracing under extreme conditions, running simple cases for which the results are known, graphically displaying intermediate results [LAWA82], using logic flow diagrams, checking reasonableness of output for a variety of input, and writing well documented code [BANJ84].

More recently, a comprehensive view of these topics and their relation to the model development process is taken. Balci [BALO86] discusses the simulation model life cycle and [BALO86,BALO87] addresses credibility assessment in terms of the following steps: formulated problem verification, feasibility assessment of simulation, system and objectives definition verification, model qualification, communicative model verification, programmed model verification, experimental design verification, data validation, and model validation. The latter six steps form the process of quality assurance of the experimental model. Related work includes [BALO84] (a bibliography on credibility assessment and validation literature) by Balci and Sargent, [NANR81,NANR87a] by Nance, and [BALO85] by Balci and Nance.

Banks, Gerstein, and Searles [BANJ87] take a broad view of the validation and verification processes. The authors describe various approaches from the literature to simulation model development and discuss validation and verification as activities that should be performed throughout the model development process.

CHAPTER 3

DYNAMIC HIERARCHICAL NETWORKS

3.1. Description

A dynamic hierarchy is suitable for an application that is sensitive to multiple external situations or seeks to adapt to changing internal conditions. Each external situation or collection of internal conditions defines a *scenario*. For each scenario, an apex node (and a corresponding hierarchical configuration of the topology) is designated as the most beneficial. At any instant, the network conforms to one of the specified configurations. When the scenario changes, the network undergoes a transition, with the appropriate node becoming the apex of the hierarchy corresponding to the reconfigured topology. Choice of the apex node for each configuration is a design decision, which may be constrained by the intended application.

In general, a dynamic hierarchy with K nodes has $L \geq K-1$ physical interconnections (links). In each configuration, there exist $K-1$ active links (whose service rates are nonzero) and $L-K+1$ inactive links (which have service rates of zero and thus carry no messages). The set of active links is such that the interconnections form a tree structured topology. A link is inaccessible and may be considered nonexistent in any configuration in which it is inactive.

Different configurations generally have different sets of active and inactive links. A link may be active in every configuration, but no link may be inactive in every configuration.

Varying sets of active and inactive links contribute to a network topology that is physically variable. As scenario changes occur, various links are logically enabled or inhibited, which, together with changes in the apex node, induces a dynamic topology. A network of this type may also be viewed as one with a varying set of redundant interconnections. The need for this redundancy may be motivated by constraints on network performance measures such as survivability, reliability, and response time.

Note that in the case where $L = K - 1$, the network topology is physically static. Every link is active in all configurations. The interconnections thus remain fixed. However, the network topology is logically variable as a result of changes in the apex node (and the corresponding changes in the hierarchical distribution of primary control). If this simple connectivity satisfies the requirements of a given application, such a network enables the realization of the benefits of the dynamic hierarchy without the added cost of redundant links.

Figures 1 and 2 contain examples which should clarify the difference between physically and logically dynamic topologies. (Note that in these figures and others, only one link is depicted between each pair of connected nodes. Each such link is interpreted as a link pair with one member of the pair transmitting in each direction. Further, in Figures 1, 2, and 3, for ease of discussion, link pairs are numbered separately but single, unidirectional links are not). Let A_i and I_i be the sets of active links and inactive links, respectively, in configuration i, $i = 1, 2, \ldots, M$, where M is the number of allowable configurations.

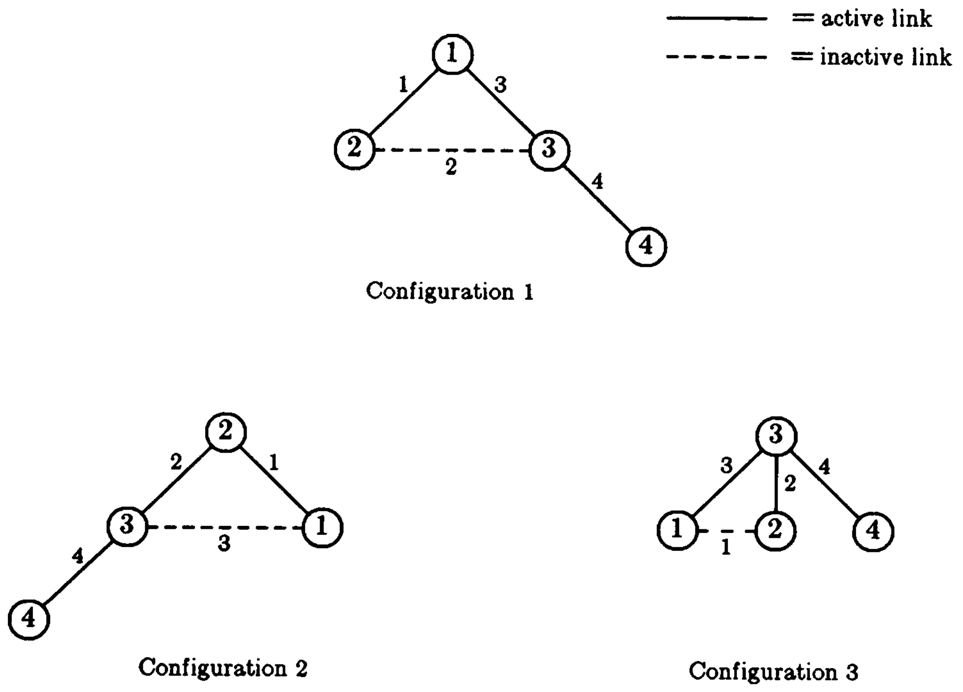


Figure 1. Example Dynamic Hierarchy with a Physically Varying Topology

Both example networks have

$K = 4$ nodes and

$M = 3$ configurations.

The network of Figure 1, which exhibits a physically varying topology, has

$L = 4$,

$A_1 = \{1, 3, 4\}$, $I_1 = \{2\}$

$A_2 = \{1, 2, 4\}$, $I_2 = \{3\}$

$A_3 = \{2, 3, 4\}$, and $I_3 = \{1\}$

As required, the elements of A_i produce a tree structured topology for each configuration i.

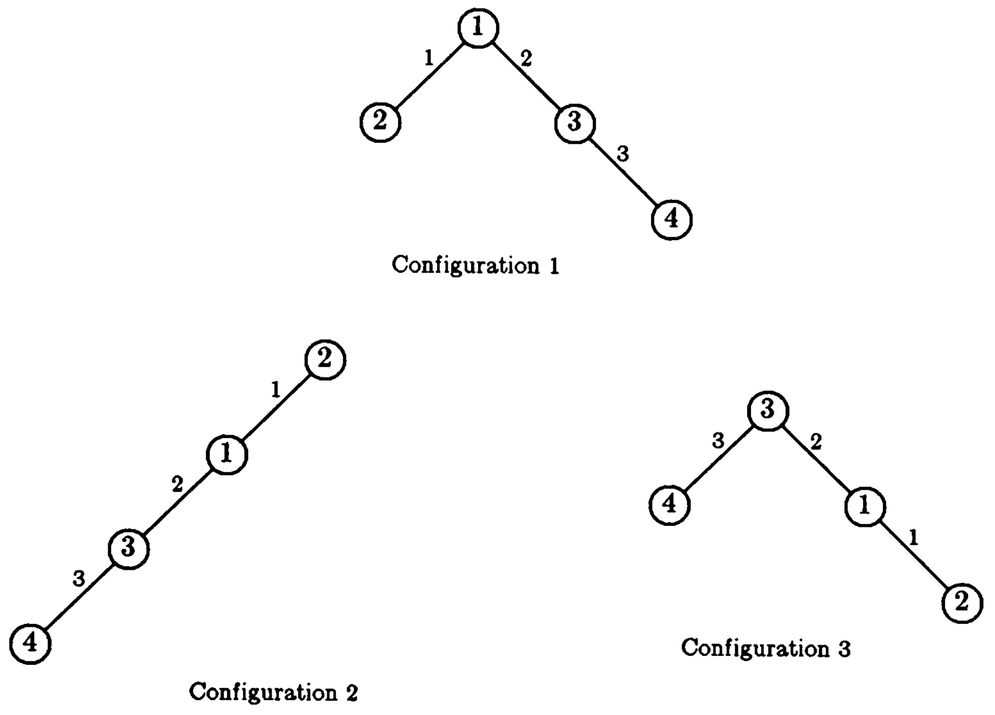


Figure 2. Example Dynamic Hierarchy with a Logically Varying Topology

Figure 2 contains a dynamic hierarchy whose topology is logically variable. This network has

$L = 3$,

$A_i = \{1, 2, 3\}$ and $I_i = \phi$, $i = 1, 2, 3$.

The foregoing characterization of dynamic hierarchical networks is not meant to restrict the class of network topologies included. Although point-to-point, tree-structured topologies are of primary interest, this characterization applies equally well to other topological classes. If, for example, common bus topologies are of interest, the network has only one physical link. Such a network, subject to changing external situations, can be analyzed as a network with a logically variable topology by considering logical links, with effective capacities, between appropriate node pairs. In this case, variability results from defining logical links between node pairs.

The configurations of a possible bus-based counterpart of the network of Figure 1 are depicted in Figure 3. The hierarchical control structures corresponding to the topologies of Figure 1 are effected by defining logical links between the appropriate node pairs. Each pair connected by an active link in the point-to-point network utilizes a logical transmission path (represented by a dotted line in Figure 3) in the bus network.

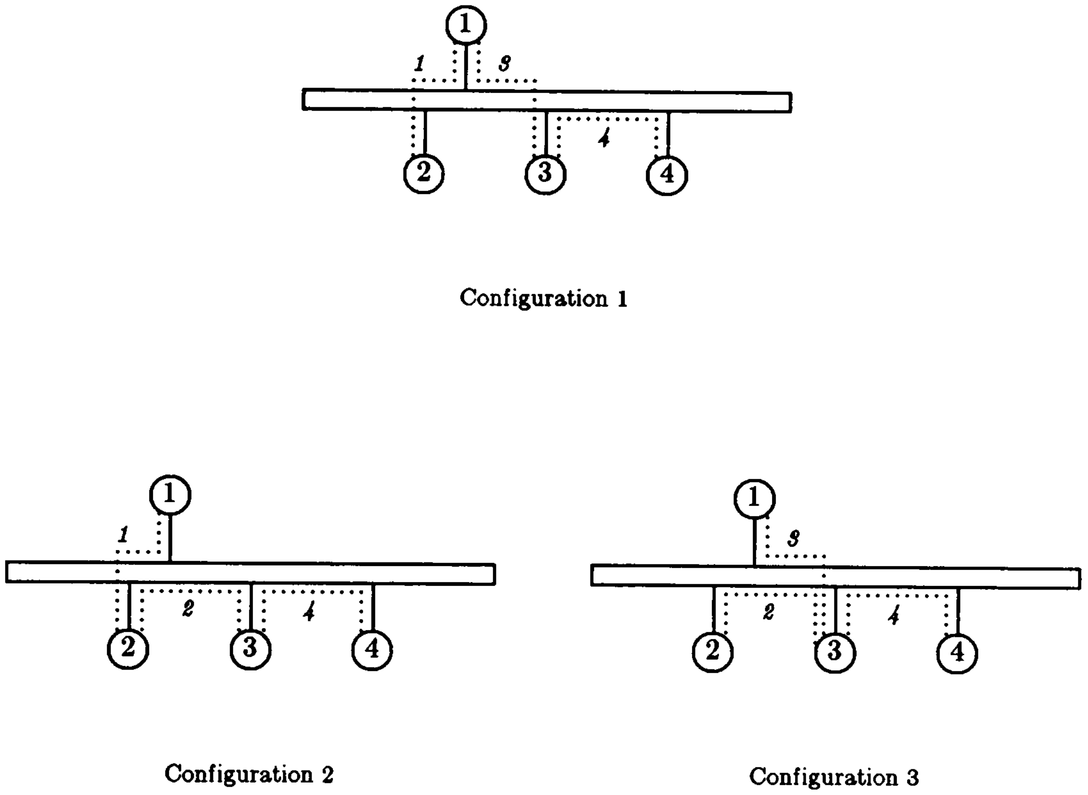


Figure 3. Example Bus-Based Dynamic Hierarchy

Consider a link j in a physically dynamic hierarchy and let j be inactive in at least one configuration. Then for each configuration i in which link j is active, the service rate of messages at j is some positive number μC_j. For each configuration in which link j is inactive, no traffic is transmitted by j and thus the service rate at j is zero.

For a bus-based dynamic hierarchy, it must be assumed that in each configuration, each node pair that communicates directly over a logical link is guaranteed a predetermined effective capacity by the protocol. The analysis depends fundamentally on the existence of fixed capacity transmission lines between specific node pairs. If the protocol does not provide these capacities, the analytic results do not hold. Suppose then that these capacities are available. Then a logical link j that exists in configuration i has some positive service rate μC_j. For the analysis, if link j does not exist in some configuration, then it is considered to exist conceptually but to have a service rate of zero.

Neither physically dynamic topologies nor bus and ring topologies are addressed directly in this paper. The model of the logically dynamic, point-to-point hierarchy applies to physically dynamic, point-to-point hierarchies and bus- and ring-based hierarchies [†]. However, the analytic results are expected to be less accurate when applied to networks in these topological classes.

3.2. Significance and Motivations

3.2.1. *Application to the Physical Problem*

Two separate but related considerations establish the dynamic hierarchy as a significant concept. First, the literature indicates that local area networking, distributed computing, and combinations of the two will continue to experience increases in popularity and applicability. The dynamic hierarchy is intended as a means for providing local area networking and distribution of primary control in a particular application area. This

† Protocol issues and other practical problems of network operation should be considered separately for each topological class.

area is characterized by the need for real-time or time-critical response and a desire to improve network survivability, reliability, availability, or other performance measures [NANR82] by employing a distributed mode of network operation while maintaining some form of global control. A reluctance or an inability to allow fully autonomous operation of the network elements may be due to the existence of natural control partitionings or a hierarchical command and control structure in the proposed application environment. Global control with limited distribution may be achieved through the use of the dynamic hierarchy.

Primary advantages of this limited distribution of control include an increased ability to meet time-critical response requirements and a higher degree of flexibility in handling the variability of traffic patterns (arrival rates). Reconfiguration brings critical links and nodes closer to the apex of the hierarchy, thus increasing the ability of these resources to provide service in a timely manner. Higher flexibility results from choosing apex nodes and configurations that are best able to meet constraints (timing or otherwise) imposed by the different scenarios. The network need not operate under a static topology and control structure resulting from a compromise design to render adequate performance over all scenarios.

3.2.2. *Theoretical Interest*

The second supporting consideration derives from the work of Neuts, Purdue, and others on queues in random environments (See Section 2.8 of the literature review). Typical applications of the theory established therein include the modeling of systems that experience rush hour behavior in their arrival streams and of service counters that are subject to server breakdowns or rest periods. These and similar characteristics are not easily included in a traditional single server queueing model. The RE-queue represents a more detailed and realistic model in such cases.

Outside of the authors' preliminary research (e.g, [NANR87b]), a network generalization of the RE-queue has not yet been proposed. Dynamic hierarchy protocols are discussed in [NAGS86] but the analysis therein is not queue-theoretic. An alternate analytic model and the treatment of *reconfiguration periods* are introduced in [BHAU86].

The research discussed herein involves modeling the dynamic hierarchy as a network of $M/M/1$ queues in a random environment, an *RE-network*. Queue-theoretic results of this research will contribute significantly to a basis for understanding the performance characteristics of dynamic hierarchical (computer communication) networks. Further, the author believes that these results may be applied with equal validity to systems other than computer communication networks. For example, just as there exist systems such as bank tellers and traffic lights, which experience rush hour arrival patterns and are reasonably modeled as RE-queues, systems such as transportation networks with seasonal demand levels and plant control networks with variable production rates also exists, for which an RE-network may represent a reasonable model. Thus, the set of potential beneficiaries of this research includes both computer network analysts and others whose interests are outside (but possibly related to) the field of computer networks.

3.3. Previous Research

Previous work on the dynamic hierarchy link capacity assignment problem is reported in [MOOR83], [NANR85,NANR87b]. First, an approximation of mean network delay (sojourn time) is derived. This approximation represents the basic measure of network performance for subsequent optimization and analysis. Next, a number of suboptimal, probabilistic and heuristic capacity assignment strategies are defined. Statistical techniques then are used to compare the effects of these strategies and to identify the best strategies.

For conventional networks, given the assumptions of Kleinrock [KLEL64], a closed form expression for mean delay is derived through the application of elementary queueing theory. This expression is extended as follows to provide an approximate measure of delay in the dynamic hierarchy: Let $\pi^{(i)}$ be the stationary probability of occurrence of configuration i. Now consider configuration i as if it were the topology of a static network and let $T^{(i)}$ be mean delay (as derived by Kleinrock) for that network. Mean delay for the dynamic hierarchy is approximated by taking the weighted sum, where the $\pi^{(i)}$ are the weights, of individual configuration mean delay. That is, T, approximate mean delay, is given by

$$T = \sum_{i=1}^{M} \pi^{(i)} T^{(i)}.$$

The dynamic hierarchy capacity assignment problem is:

Given the set of configurations, the configuration probabilities, and a characterization of network traffic, select link capacities that minimize total cost subject to an upper bound on mean delay.

Two sets of strategies are created to assign capacities. Members of the first set, referred to as the probabilistic strategies, are created by adapting previous analytic results from conventional network design. Consider again configuration i as if it were the topology of a static network. Let $C_j^{(i)}$ be a capacity assignment for link j that is in some sense optimal for this network (for example, the $C_j^{(i)}$ might be calculated through the cost minimization counterpart of Kleinrock's square-root strategy [KLEL76, p. 350]). Then the capacity assignment for link j in the dynamic hierarchy is

$$C_j = \sum_{i=1}^{M} \pi^{(i)} C_j^{(i)}.$$

Different formulae for the $C_j^{(i)}$ and variants of the weighted sum construction yield different assignment strategies.

Members of the second set, referred to as the heuristic strategies, are algorithmic in nature and assume a discrete cost function. These strategies are constructed by first defining a group of capacity assignment heuristics and then taking various combinations of these heuristics to produce composite capacity assignment algorithms. This approach follows the method of Maruyama, et al. in, for example, [MARK76].

As the final step, the strategies are compared through the use of analysis of variance (ANOVA) procedures. Statistical comparisons are necessary because the strategies are approximate and heuristic in nature and the need exists to extend the conclusions of any comparisons to all dynamic hierarchies. These tests are reported fully in [NANR87b].

CHAPTER 4

QUEUEING MODELS

4.1. Network Model

The ideal goal in analyzing the dynamic hierarchy is to derive theoretically exact solutions for queue length probabilities and mean waiting time from an appropriate network model. An a network of $M/M/1$ queues in a random environment is a natural candidate. The exact analysis of such a model is beyond the scope of this research. However, to provide a basis for subsequent simplification, a description of the general network model follows. This description assumes that network configuration changes are instantaneous, an assumption that is dropped in one of the single link models.

A note on the variability of arrival rates.

Generally it is possible to identify multiple situations in a dynamic hierarchy application environment. Each situation is characterized by a different set of external arrival rates, the rates of message arrivals to nodes from outside the network. The combined effect of variable external arrival rates and a varying topology is that the links experience a different set of internal arrival rates, composite arrival rates due to external arrivals and to message forwarding within the network, for each situation.

In some cases, variability of arrival rates is caused entirely or partially by changing conditions inside the network. However, it remains possible, as an abstraction, to view variable internal arrival rates as being caused by variable external arrival rates (and a varying topology).

In either of these cases the arrival rates and the network are considered to be influenced by an external environment. Each state of the environment corresponds to one of the scenarios (set of external arrival rates or internal conditions), which are in one-to-one correspondence with network configurations/potential apex nodes.

The assumptions that are necessary to model the dynamic hierarchy as an RE-network include the following:

(1) Scenario changes occur according to a random (Markov) environment process.

(2) The network configuration for each scenario is given.

(3) When the environment process is in state i, messages with source node j and destination node k arrive at node j according to a Poisson process with rate $\gamma_{jk}^{(i)}$.

(4) Given the state of the environment process, the arrival processes of (3) are mutually independent.

(5) Nodal processing times are negligible.

(6) Messages pass through the network in a store-and-forward fashion. That is, for each source/destination pair j, k in configuration i, there exists a uniquely specified shortest path $\psi_{jk}^{(i)}$ of links connecting nodes j and k. A message arriving at node j is alternately stored at the source or an intermediate node and transmitted across the next link in $\psi_{jk}^{(i)}$. (The symbol ψ_{jk}, without the superscript (i), is used in later sections to denote the emphasis on logically dynamic hierarchies, in which message paths do not vary with the environment state and configuration). Full receipt of the message is required prior to each forwarding operation. The message departs the network following its delivery to destination node k.

(7) At each node, there exists a separate first-come-first-served queue with unlimited buffer space for each outgoing link.

(8) Propagation times are negligible.

(9) Each node possesses sufficient processing capability to operate all incoming and outgoing links (transmissions) simultaneously.

(10) Links are physically capable of transmission in one direction only. Bidirectional transmission is enabled by connecting nodes with link pairs whose components transmit in opposite directions.

(11) Links are noiseless and error free.

(12) The lengths of arriving messages are exponentially distributed with mean μ^{-1} bits. Additionally, at each intermediate node in the path of a message, upon entering service, the length of the message is reset according to the same exponential distribution (A variant of Kleinrock's independence assumption [KLEL64]).

(13) The message length processes of (12) are mutually independent.

(14) The collections of processes of (3) and (12) are independent of each other.

Under these assumptions, definition of the appropriate stochastic process representing an RE-network is straightforward. Let $E = \{E(t);\ t \geq 0\}$, the environment process, be a stationary, irreducible Markov process with finite state space $S_E = \{1, 2, \ldots, M\}$, where M is the number of possible environments. Consider a network of L queues, each of which has infinite waiting room and serves customers according to a first-come-first-served discipline. Each queue $l \in \{1, 2, \ldots, L\}$ operates under the influence of the environment process as follows: On $\{E(t) = i\}$, $i \in S_E$,

(1) Arrivals from outside the network occur according to a Poisson process with rate $\lambda'^{(i)}_l$.

(2) Service times are IID random variables following an exponential distribution with mean $\frac{1}{\mu C_l}$. (The mean message length is μ^{-1} bits and the transmission capacity is C_l bits/second).

Assume that the service time processes are mutually independent. Further, assume that given the state of E, the external arrival processes are mutually independent, and the collections of service time processes and arrival processes are independent of each other.

For routing purposes, messages are assumed to be typed according to their source and destination (nodes). A *jk*-message, a message with source j and destination k, arrives to node j, follows a fixed (unique for each environment state) path $\psi^{(i)}_{jk}$ of links through the network, and departs after reaching node k.

Now let $\underline{N} = \{\underline{N}(t);\ t \geq 0\}$ be the process of network queue lengths, where

$$\underline{N}(t) = (N_1(t), N_2(t), \ldots, N_L(t))$$

and $N_l(t)$ is the length (including the customer in service, if any) of queue l at time t. Then the process of interest, $(\underline{N}, E) = \{\underline{N}(t), E(t);\ t \geq 0\}$ is a Markov process with state space $(\bigtimes_{l=1}^{L} S_{N_l}) \times S_E$, where $S_{N_l} = \{0, 1, 2, \ldots\}$ is the state space of a single queue length process.

To model the dynamic hierarchy as above, the correspondence must be specified between the external arrival processes (in the form of source/destination node pairs) and the resultant processes of external arrivals to the individual servers (links). Let $\Gamma^{(i)} = \{\gamma^{(i)}_{jk}\}$ be the matrix of arrival rates, expressed in terms of source/destination pairs, for configuration i. Consider a single node j with $\{l_1, l_2, \ldots, l_n\}$ the set of outgoing links that are connected to the node and are active in configuration i. For each $l \in \{l_1, l_2, \ldots, l_n\}$ define the external arrival rate to link l in configuration i as

$$\lambda'^{(i)}_l = \sum_{\{jk: l \in \psi^{(i)}_{jk}\}} \gamma^{(i)}_{jk}.$$

It can be shown that in configuration i, the external arrival process to link l is Poisson with rate $\lambda'^{(i)}_l$. Briefly, on $\{E(t) = i\}$, the original external arrival processes are independent, Poisson with rates $\gamma^{(i)}_{jk}$. Thus, the superposition of the specified processes is Poisson with rate $\lambda'^{(i)}_l$ [CINE75]. Further, given the state of the environment process, these individual link, external arrival processes are mutually independent and the collection of these processes is independent of the collection of service time (message length) processes. Note that $\lambda'^{(i)}_l$ accounts for arrivals from outside the network only. The composite rate $\lambda^{(i)}_l$ of message arrivals to link l in configuration i includes contributions due to message forwarding inside the network. In general, derivation of the $\lambda^{(i)}_l$'s requires detailed knowledge of the steady state message flow rates.

4.2. Approximate Single Link Models

Since the solution to the RE-network is not expected to be of product form and the rates of flow in the network are largely unknown, an additional assumptions is necessary and is introduced in Section 4.3. In this section, matrix-geometric methods first are discussed and then are applied as a technique for modeling and analyzing the individual queues as if they are RE-queues in isolation.

In modeling the dynamic hierarchy, nodal processing times are assumed negligible and each node is considered to possess sufficient processing capability to operate all incoming and outgoing links simultaneously. Thus, the links and their associated queues are the individual service systems of interest. Nodes are considered points at which messages are routed to the appropriate links or to outside of the network [†]. Two variants of the RE-queue are introduced to model these individual systems. The first variant incorporates the assumption that network configuration transitions (reconfigurations) are instantaneous. The second allows for a reconfiguration period preceding each transition.

(The following notational conventions are employed in this section: Symbols in bold type denote vectors. Depending on the context in which they appear, symbols in normal, non-bold type denote matrices or scalars. The symbols **0** and **1** denote the appropriate size vectors of zeros and ones, respectively).

4.2.1. *Matrix-Geometric Results*

Several main results from the RE-queue literature are needed to analyze the model of a dynamic hierarchy link in isolation. These results relate primarily to queue length and virtual waiting time. Since Neuts is the primary contributor to the theory of *phase processes*, or *PH-processes,* of which the RE-queue is an application, the presentation in this section follows the order and a modified form of the notation of his text on *matrix-geometric methods* [NEUM81b]. (The theory of *quasi birth and death processes,* which are contained in the set of PH-based processes, originates with the dissertation of Wallace [WALV69]).

Neuts observes that in "the construction of algorithmic solutions [of stochastic models], it is overwhelmingly clear that the *structure* [‡] is of paramount importance. The

† Nodes may also be considered message producers and consumers. However, the distinction between this view and the current view that messages originate outside the network and ultimately exit the network is not important with respect to this analysis.
‡ "Structure" refers to the structure as reflected in the transition matrix.

specific analytic form of the elements of the transition probability matrix is of far less consequence . . .' [NEUM81b, p. 2]. The overriding characteristics of the matrix-geometric approach are

- Exploitation of structures of transition matrices or infinitesimal generators to reach solutions through probabilistic approaches. Transform methods are avoided when possible. In general, the transition matrices and generators are expressed in terms of sub-matrices that are defined by partitioning the original matrices. Most of the resultant solutions involve matrix operations.
- A 'concern for solution methods that are implementable in a general and numerically stable manner and offer detailed information on at least some of the more complex models encountered in the study of practical queueing systems.' [NEUM81b, p. 3] This characteristic falls under the general heading of ***computational probability***, which Neuts defines in the preface to [NEUM81b]. The form of the (scalar) elements of a particular matrix is of much less importance than the ability to devise a robust algorithm to solve numerically one or more matrix equations.

A Markov process † of the $GI/M/1$ type is one whose transition matrix $\tilde{Q}$ has the

† Since continuous time Markov processes are of interest, this presentation skips the analogous theory for Markov chains.

form

$$\tilde{Q} = \begin{bmatrix} B_0 & A_0 & 0 & 0 & 0 & . & . & . \\ B_1 & A_1 & A_0 & 0 & 0 & . & . & . \\ B_2 & A_2 & A_1 & A_0 & 0 & . & . & . \\ B_3 & A_3 & A_2 & A_1 & A_0 & . & . & . \\ B_4 & A_4 & A_3 & A_2 & A_1 & . & . & . \\ . & . & . & . & . & . & & \\ . & . & . & . & . & & . & \\ . & . & . & . & . & & & . \end{bmatrix}$$

where the B_i and A_i are $M \times M$ nonnegative matrices. The Markov process $\tilde{Q}$ is assumed to be irreducible. $\tilde{Q}$ has state space $\{(i,\ j)\text{: } i \geq 0,\ 1 \leq j \leq M\}$, where the states are ordered lexicographically. The set $\{(i,\ 1), \ldots, (i,\ M)\}$ is called *level* i.

Let $\mathbf{x} = [\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots]$, where $\mathbf{x}_i = [x_i^{(1)}, \ldots, x_i^{(M)}]$ for each i, be the invariant probability vector of $\tilde{Q}$ and define

$$A = \sum_{k=0}^{\infty} A_k .$$

Under appropriate conditions (irreducible $\tilde{Q}$, nonsingular B_0 and A_1, and $A\mathbf{1} = \mathbf{0}$), Theorem 1.7.1 of Neuts [NEUM81b, pp. 32-33], which is not repeated here, gives the stationary probability vector $\mathbf{x}$ satisfying $\mathbf{x}\tilde{Q} = \mathbf{0}$, $\mathbf{x}\mathbf{1} = 1$ for the general $GI/M/1$ type process.

When the generator $\tilde{Q}$ has a block tridiagonal structure, that is

$$\tilde{Q} = \begin{bmatrix} B_0 & A_0 & 0 & 0 & 0 & . & . & . \\ B_1 & A_1 & A_0 & 0 & 0 & . & . & . \\ 0 & A_2 & A_1 & A_0 & 0 & . & . & . \\ 0 & 0 & A_2 & A_1 & A_0 & . & . & . \\ 0 & 0 & 0 & A_2 & A_1 & . & . & . \\ . & . & . & . & . & . & & \\ . & . & . & . & . & & . & \\ . & . & . & . & . & & & . \end{bmatrix},$$

the $\tilde{Q}$ process is said to be a ***quasi birth and death process*** or ***QBD process.*** In this case, the matrix A simplifies to

$$A = A_0 + A_1 + A_2.$$

The general theorem is restated as follows [NEUM81b, pp. 82-83]:

> The process $\tilde{Q}$ is positive recurrent if and only if the minimal nonnegative solution R [the ***rate matrix***] to the matrix-quadratic equation
>
> $$R^2 A_2 + RA_1 + A_0 = 0$$
>
> has all its eigenvalues inside the unit disk [denoted by $sp(R) < 1$] and the finite system of equations
>
> $$\mathbf{x}_0(B_0 + RB_1) = \mathbf{0}$$
> $$\mathbf{x}_0(I - R)^{-1}\mathbf{1} = 1$$
>
> has a unique positive solution $\mathbf{x}_0$.
>
> If the matrix A is irreducible, then $sp(R) < 1$ if and only if
>
> $$\boldsymbol{\pi} A_2 \mathbf{1} > \boldsymbol{\pi} A_0 \mathbf{1},$$
>
> where $\boldsymbol{\pi}$ is the stationary probability vector of A.
>
> The stationary probability vector $\mathbf{x} = [\mathbf{x}_0, \mathbf{x}_1, \ldots]$ of $\tilde{Q}$ is given by
>
> $$\mathbf{x}_i = \mathbf{x}_0 R^i, \text{ for } i \geq 0.$$
>
> The (equivalent) equalities
>
> $$RA_2\mathbf{1} - A_0\mathbf{1} = RB_1\mathbf{1} - B_0\mathbf{1} = 0$$
>
> hold.

Waiting times in queues modeled as QBD processes are analyzed as absorption times in finite Markov processes. The appropriate Markov process has generator

$$\tilde{Q^0} = \begin{bmatrix} 0 & 0 & 0 & 0 & . & . & . \\ A_2 & D & 0 & 0 & . & . & . \\ 0 & A_2 & D & 0 & . & . & . \\ 0 & 0 & A_2 & D & . & . & . \\ . & . & . & . & . & & \\ . & . & . & . & & & . \\ . & . & . & . & & & . \end{bmatrix},$$

where A_2 and D are $M \times M$ matrices whose forms depend on the model under consideration.

Define $K = -D^{-1}A_2$. Let $\boldsymbol{y}(0) = [\boldsymbol{y}_0(0), \boldsymbol{y}_1(0), \boldsymbol{y}_2(0), \ldots]$ be the initial probability vector of $\tilde{Q^0}$ and $\boldsymbol{W}(x) = [W^{(1)}(x), W^{(2)}(x), \ldots, W^{(M)}(x)]$ be the vector of absorption probabilities, where $W^{(j)}(x) = P(\text{absorption into } (0, j) \text{ by time } x)$. The primary results of interest here are [NEUM81b, pp. 133-134]:

(1) The vector of Laplace-Stieltjes transforms of the components of $\boldsymbol{W}(x)$ is

$$\boldsymbol{w}(s) = \sum_{k=0}^{\infty} \boldsymbol{y}_k(0)[(sI - D)^{-1}A_2]^k \text{ , for } s \geq 0 \text{ and}$$

(2) The mean vector $-\boldsymbol{w}'(0)$ is finite if and only if the vector $\sum_{k=1}^{\infty} k\boldsymbol{y}_k(0)$ is finite. It is then given by

$$-\boldsymbol{w}'(0) = \sum_{k=1}^{\infty} \boldsymbol{y}_k(0) \sum_{\nu=0}^{k-1} K^{\nu}(-D^{-1})K^{k-\nu}.$$

Consider now an $M/M/1$ queue in a random environment. The environment is assumed to be an M state Markov process E with irreducible generator Q. On $\{E(t) = i\}$, arrivals occur according to a Poisson process with rate $\lambda^{(i)}$, service times are IID exponential random variables with mean $\frac{1}{\mu^{(i)}}$, and the arrival and service time processes are independent. The arrival and service rates change instantaneously when E changes state. The queueing discipline is first-come-first-served.

Let $N(t)$ be the queue length at time t. Define $\boldsymbol{\lambda} = [\lambda^{(1)}, \lambda^{(2)}, \ldots, \lambda^{(M)}]$ and $\boldsymbol{\mu} = [\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(M)}]$ as the arrival and service rate vectors, respectively. Then $(N, E) = \{(N(t), E(t));\ t \geq 0\}$ is a QBD process with generator

$$\tilde{Q} = \begin{bmatrix} Q - \Delta(\boldsymbol{\lambda}) & \Delta(\boldsymbol{\lambda}) & 0 & 0 & \cdots \\ \Delta(\boldsymbol{\mu}) & Q - \Delta(\boldsymbol{\lambda}+\boldsymbol{\mu}) & \Delta(\boldsymbol{\lambda}) & 0 & \cdots \\ 0 & \Delta(\boldsymbol{\mu}) & Q - \Delta(\boldsymbol{\lambda}+\boldsymbol{\mu}) & \Delta(\boldsymbol{\lambda}) & \cdots \\ 0 & 0 & \Delta(\boldsymbol{\mu}) & Q - \Delta(\boldsymbol{\lambda}+\boldsymbol{\mu}) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

where

$$\begin{aligned} \Delta(\boldsymbol{u}) &= \Delta([u^{(1)}, u^{(2)}, \ldots, u^{(M)}]) \\ &= \begin{bmatrix} u^{(1)} & 0 & \cdots & 0 \\ 0 & u^{(2)} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & u^{(M)} \end{bmatrix} \end{aligned}$$

The corresponding results regarding queue length probabilities are [NEUM81b, p. 258]:

> Let $\boldsymbol{\pi}$ be the stationary probability vector of Q. Then the queue is *stable* if and only if
>
> $$\boldsymbol{\pi}\boldsymbol{\lambda} < \boldsymbol{\pi}\boldsymbol{\mu}.$$
>
> The matrix R is the minimal solution of the equation
>
> $$R^2\Delta(\boldsymbol{\mu}) + R[Q - \Delta(\boldsymbol{\lambda}+\boldsymbol{\mu})] + \Delta(\boldsymbol{\lambda}) = 0,$$
>
> and . . . provided [the stability condition holds], we have
>
> $$R\boldsymbol{\mu} = \boldsymbol{\lambda}.$$

Also [NEUM81b, p. 258]:

> The stationary probability vector $\boldsymbol{x} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots]$ of the stable queue is given by
>
> $$\boldsymbol{x}_k = \boldsymbol{\pi}(I - R)R^k, \text{ for } k \geq 0.$$

The matrix R may be computed as follows [NEUM81b, pp. 38, 258]:

1. $R(0) = 0$
2. $R(n+1) = \{[R(n)]^2 \Delta(\mu) + \Delta(\lambda)\}[\Delta(\lambda + \mu) - Q]^{-1}$
3. Repeat step 2 to determine R^* as the first $R(n+1)$ for which
   $MAX_{i,j}\{R_{ij}(n+1) - R(n)\} < \epsilon$
   for some small $\epsilon > 0$.
   This R^* is taken as the approximate value of R.

Waiting times in this queue are analyzed by considering a version of (N, E) in which no arrivals occur. The modified process begins with an arbitrary (random) number of customers and services them until the last customer departs, at which time the process is absorbed into one of the states of level 0. If the initial distribution of the number of customers present $\boldsymbol{y}(0)$ (as discussed previously) is chosen to be the stationary queue length distribution, $\boldsymbol{x}$, for (N, E) at arbitrary times, the time until absorption in the modified process is the virtual waiting time (the time that a virtual customer would spend waiting in the queue) for (N, E).

The generator of the absorbing process is

$$\tilde{Q}^0 = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots \\ \Delta(\mu) & Q - \Delta(\mu) & 0 & 0 & \cdots \\ 0 & \Delta(\mu) & Q - \Delta(\mu) & 0 & \cdots \\ 0 & 0 & \Delta(\mu) & Q - \Delta(\mu) & \cdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

so that $A_2 = \Delta(\mu)$, $D = Q - \Delta(\mu)$, and $K = -[Q - \Delta(\mu)]^{-1}\Delta(\mu)$. Thus, the vector of Laplace-Stieltjes transforms of the virtual waiting time distributions is

$$W_V^*(s) = \boldsymbol{\pi}(I - R)\sum_{k=0}^{\infty} R^k[(sI + \Delta(\mu) - Q)^{-1}\Delta(\mu)]^k$$

and mean virtual waiting time is

$$\begin{aligned}\boldsymbol{W}_V &= -\boldsymbol{W}_V^{*\prime}(0)\mathbf{1}\\ &= \boldsymbol{\pi}(I-R)\sum_{k=1}^{\infty}R^k\sum_{\nu=0}^{k-1}[-(Q-\Delta(\boldsymbol{\mu}))^{-1}\Delta(\boldsymbol{\mu})]^{\nu}[-(Q-\Delta(\boldsymbol{\mu}))]^{-1}\\ &\qquad\cdot[-(Q-\Delta(\boldsymbol{\mu}))^{-1}\Delta(\boldsymbol{\mu})]^{k-\nu}\mathbf{1}.\end{aligned}$$

($\boldsymbol{W}_V^{*\prime}(s)$ denotes the vector of derivatives of the elements of $\boldsymbol{W}_V^{*}(s)$). Mean virtual time in the system (time in the queue plus service time) is thus

$$\bar{S}_V = \bar{W}_V + (\boldsymbol{\pi\mu})^{-1},$$

where $(\boldsymbol{\pi\mu})^{-1}$ is the mean *effective service time* [NEUM81b, p. 272].

Other waiting times can be analyzed if the appropriate initial distributions for the $\tilde{Q}^0$ process are known. For example, if the vector $\boldsymbol{z} = [\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2, \ldots]$ of stationary queue length probabilities for the $\tilde{Q}$ process at arrival times is known, and $\boldsymbol{y}(0)$ is set equal to $\boldsymbol{z}$, then the time until absorption into a state of level 0 of $\tilde{Q}^0$ is the customer waiting time (not including service time) in $\tilde{Q}$.

4.2.2. *Instantaneous Reconfigurations*

Consider a single link j in a dynamic hierarchy. Let $E = \{E(t);\ t \geq 0\}$, a Markov process with finite state space $S_E = \{1, 2, , \ldots, M\}$ and irreducible generator

$$Q = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdot & \cdot & \cdot & \sigma_{1M} \\ \sigma_{21} & \sigma_{22} & \cdot & \cdot & \cdot & \sigma_{2M} \\ \cdot & \cdot & \cdot & & & \cdot \\ \cdot & \cdot & & \cdot & & \cdot \\ \cdot & \cdot & & & \cdot & \cdot \\ \sigma_{M1} & \sigma_{M1} & \cdot & \cdot & \cdot & \sigma_{MM} \end{bmatrix},$$

be the model of the external environment. Let $\boldsymbol{\pi} = [\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(M)}]$ be the invariant probability vector of Q. (This process represents the environment of the entire network as well as that of each link. However, it is discussed here in terms of its effect on a single link).

Assume that on $\{E(t) = i\}$ the composite arrival process to link j is Poisson with rate $\lambda_j^{(i)}$, the length of a message entering service is reassigned according to an exponential distribution with mean μ^{-1}, the randomly chosen message lengths are independent, and the arrival and message length processes are independent. Let C_j be the transmission capacity of link j so that the service rate on link j in every configuration is μC_j. Assume further that the time required to effect a reconfiguration is negligible. Then reconfigurations can be considered to occur instantaneously with environment state changes.

The following additional assumptions are included in this model:

- Waiting room (buffer space) available for messages queued at a link can be considered unlimited.
- Messages are transmitted, without interruption, on a first-come-first-served basis.
- Propagation times are negligible.
- Links are noiseless and error free.

This model is an RE-queue with arrival rate vector $\boldsymbol{\lambda}_j = [\lambda_j^{(1)}, \lambda_j^{(2)}, \ldots, \lambda_j^{(M)}]$ and service rate vector $\boldsymbol{\mu}_j = [\mu C_j, \mu C_j, \ldots, \mu C_j]$. The generator for the joint environment-queue length process (N_j, E) is

$$\tilde{Q}_j = \begin{bmatrix} Q-\Delta(\lambda_j) & \Delta(\lambda_j) & 0 & 0 & \cdots \\ \mu C_j I & Q-(\Delta(\lambda_j)+\mu C_j I) & \Delta(\lambda_j) & 0 & \cdots \\ 0 & \mu C_j I & Q-(\Delta(\lambda_j)+\mu C_j I) & \Delta(\lambda_j) & \cdots \\ 0 & 0 & \mu C_j I & Q-(\Delta(\lambda_j)+\mu C_j I) & \cdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

The queue is stable if and only if $\frac{1}{\mu C_j}\pi\lambda_j < 1$.

Stationary queue length probabilities, mean queue length, and mean virtual waiting time for link j (in isolation) are computed by applying the previously discussed iterative method to determine R_j, the minimal solution to

$$R_j^2 \mu C_j I + R_j[Q-(\Delta(\lambda_j)+\mu C_j I)] + \Delta(\lambda_j) = 0,$$

and then computing the necessary matrix-geometric and related quantities.

4.2.3. *Reconfiguration Periods*

Consider now the following operating characteristic of the dynamic hierarchy: The arrival and message length processes are influenced by an external environment process as before. Additionally, preceding each reconfiguration, the network enters a reconfiguration state for a time interval known as a ***reconfiguration period*** [BHAU86] (See also [NAGS86]). Such a period allows the network nodes time to perform any processing necessary to enable operation in the new configuration. Let A_N be the set of potential apex nodes and A_L be the set of links connecting members of A_N. Then during a reconfiguration period, message transmission on links in A_L ceases. Messages arriving to a node in A_N (including external arrivals) intended for transmission over a link in A_L are queued at the link until the end of the reconfiguration period. Transmission on links not in A_L

continues. Note that a link connecting a node in A_N and a node not in A_N is not in A_L. Thus, message transmission on such a link continues during a reconfiguration period.

To model this behavior, first the states $1_0, 2_0, \ldots, M_0$ are added to S_E to form the state space S_{E_1} of the environment process E_1. The ordering $1_0, 1, 2_0, 2, \ldots, M_0, M$ is imposed on the elements of S_{E_1}. When an environment transition from *normal state* $h \in \{1, 2, \ldots, M\} = S_E$ occurs, instead of entering state $i \in S_E$, the environment enters *reconfiguration state* $i_0 \in \{1^0, 2^0, \ldots, M^0\} = S_E^0$. Simultaneously, the network enters configuration i_0, which is identical to configuration i, but in which operation follows the previously described reconfiguration procedure. The interval in which the network configuration is i_0 corresponds to a reconfiguration period. During this period, messages arrive from outside the network according to Poisson processes with rates identical to those induced by environment i.

Immediately following a period of residency in state i_0, the environment enters, with probability one, state i. The end of the reconfiguration period coincides with this transition. Normal network operation in configuration i proceeds. Arrivals from outside the network occur according to Poisson processes with rates induced by environment i.

For each link j not in A_L, regardless of whether the environment is in a state $i_0 \in S_E^0$ or a state $i \in S_E$, the length of a message entering service at the link is assumed to be reassigned according to an exponential distribution with mean μ^{-1}. The service rate at a link j with capacity C_j thus remains μC_j for every configuration. A similar assumption holds for links in A_L with the exception that the service rates on these links are zero when the environment is in a reconfiguration state.

The independence assumptions of the previous link model are retained in this variant. That is, message lengths at link j are independent and given the state of the environment, the composite arrival process and the message length process are independent.

The remaining conventional assumptions (unlimited buffer space, FCFS service, negligible propagation times, and noiseless, error free links) are retained also. Further, it is assumed that a message being transmitted on a link in A_L when service is interrupted for a reconfiguration period is placed at the front of the queue and retransmitted in full when normal service resumes.

Let $E_1 = \{E_1(t);\ t \geq 0\}$, the environment process, be a Markov process with state space $S_{E_1} = \{1_0, 1, 2_0, \ldots, M_0, M\}$ and irreducible generator Q_1. Q_1 has the following nondiagonal elements:

$$\begin{aligned}
&Q_1(h,\ i) = 0,\ h,\ i \in S_E,\\
&Q_1(h,\ i_0) = \sigma_{hi_0},\ h \in S_E,\ i_0 \in S_E^0,\\
&Q_1(h_0,\ i) = 0,\ h_0 \in S_E^0,\ i \in S_E,\ i \neq h,\\
&Q_1(h_0,\ h) = \sigma_{h_0h}.
\end{aligned}$$

Thus, as previously described, from any normal state h, the environment may enter (in one step) any reconfiguration state and from any reconfiguration state h_0, it may enter only the associated normal state h.

Note that $Q_1(h,\ h_0)$ may be nonzero. Hence, state h may be followed in two steps by itself. The effect on the network of such a sequence of transitions is that it remains in configuration h and resumes normal operation after a reconfiguration period (Recall that configuration h_0 is, by definition, identical to configuration h).

The environment influences a link j as follows: On $\{E_1(t) = i\}$ for $i \in S_E$, the composite arrival process to link j is Poisson with rate $\lambda_j^{(i)}$ and the length of a message entering service is reassigned according to an exponential distribution with mean μ^{-1}. On $\{E_1(t) = i_0\}$ for $i_0 \in S_E^0$, the composite arrival process to link j is Poisson with rate $\lambda_j^{(i_0)}$. If link j is in A_L, no messages are transmitted (the service rate is zero). Otherwise, transmission proceeds as before at rate μC_j.

This model of a dynamic hierarchy link is an RE-queue with server interruptions. If $j \in A_L$, then (N_j, E_1) has arrival rate vector $\boldsymbol{\lambda}_j = [\lambda_j^{(1_0)}, \lambda_j^{(1)}, \lambda_j^{(2_0)}, \lambda_j^{(2)}, \ldots, \lambda_j^{(M_0)}, \lambda_j^{(M)}]$ and service rate vector $\boldsymbol{\mu}_j = [0, \mu C_j, 0, \mu C_j, \ldots, 0, \mu C_j]$. For j not in A_L, $\boldsymbol{\lambda}_j = [\lambda_j^{(1_0)}, \lambda_j^{(1)}, \lambda_j^{(2_0)}, \lambda_j^{(2)}, \ldots, \lambda_j^{(M_0)}, \lambda_j^{(M)}]$ and $\boldsymbol{\mu}_j = [\mu C_j, \mu C_j, \mu C_j, \mu C_j, \ldots, \mu C_j, \mu C_j]$. The $\lambda_j^{(i)}$ for $i \in S_E$ are the same as those of the previous model. However, the effect of service interruptions on links in A_L propagates throughout the network so that in general, $\lambda_j^{(i_0)} \neq \lambda_j^{(i)}$ for all i_0 irrespective of whether $j \in A_L$.

The generator for (N_j, E_1) is

$$\tilde{Q}_j = \begin{bmatrix} Q_1 - \Delta(\boldsymbol{\lambda}_j) & \Delta(\boldsymbol{\lambda}_j) & 0 & 0 & \cdots \\ \Delta(\boldsymbol{\mu}_j) & Q_1 - \Delta(\boldsymbol{\lambda}_j + \boldsymbol{\mu}_j) & \Delta(\boldsymbol{\lambda}_j) & 0 & \cdots \\ 0 & \Delta(\boldsymbol{\mu}_j) & Q_1 - \Delta(\boldsymbol{\lambda}_j + \boldsymbol{\mu}_j) & \Delta(\boldsymbol{\lambda}_j) & \cdots \\ 0 & 0 & \Delta(\boldsymbol{\mu}_j) & Q_1 - \Delta(\boldsymbol{\lambda}_j + \boldsymbol{\mu}_j) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}.$$

The stability condition for this queue is $\frac{\boldsymbol{\pi}\boldsymbol{\lambda}_j}{\boldsymbol{\pi}\boldsymbol{\mu}_j} < 1$, where

$\boldsymbol{\pi} = [\pi^{(1_0)}, \pi^{(1)}, \pi^{(2_0)}, \ldots, \pi^{(M_0)}, \pi^{(M)}]$ is the invariant probability vector of Q_1.

As with the previous model, an appropriate matrix-quadratic equation may be written and its minimal solution R_j used to compute stationary queue length probabilities, mean queue length, and mean virtual waiting time.

4.2.4. *Other Models*

In separate but related work, Bhat and Nance [BHAU86] propose a third model for links (and nodes) in the dynamic hierarchy. Each link is modeled as an independent M/M/1 queue and three periods of operation are considered: ***normal operation, reconfiguration***, and ***adjustment***. A reconfiguration period is a period during which the network configuration changes and transmission between potential apex nodes ceases. An adjustment period is the first busy period of a link (particularly a link between two potential apex nodes) following a reconfiguration period. Mean delay for a link is derived by combining the components due to each of these periods. Mean nodal delay is derived similarly.

A primary advantage of this approach is that it enables examination of the busy period behavior of links (and nodes) following reconfigurations and the impact of this adjustment behavior on network performance. The models described here do not recognize adjustment periods explicitly. An adjustment period is included in the period of normal operation following a reconfiguration but is not analyzed separately. This approach, using RE-queue models, is better suited for examining steady-state network characteristics and the effects over time of a variable environment on the network.

4.3. Network Model with the Departure Process Assumption

Exact analysis of the RE-network model of Section 4.1 through the multidimensional Markov process $(\underset{\sim}{N}, E)$ is beyond the scope of this research. Further, the network is not separable in the sense that the product form networks of Kelly (or others) are separable. To decompose the network into a collection of single server (RE-) queues, an additional assumption is introduced. Results for single queues are obtained by viewing them as if they were RE-queues in isolation. The single queue results are combined to produce a characterization of network performance.

The assumption that leads to a decomposition of the network is:

Departure process assumption: Assume that at each queue, the process of jk-message departures is statistically identical to the process of jk-message arrivals to the queue.

The following result now holds.

Theorem 1.

Given the departure process assumption

(1) For each jk, each flow process of jk-messages in the network is a Poisson process in a random environment (an *RE-Poisson* process).

(2) The jk-arrival processes to a link l, for all jk such that $l \in \psi_{jk}$, are mutually independent given the environment state.

(3) The composite arrival process to each link is an RE-Poisson process.

proof:

First, the following characteristic of logically dynamic hierarchies is reiterated and its implication is noted: A jk-message passes through the network over a unique, shortest path ψ_{jk}. Since ψ_{jk} is the shortest path from node j to node k, it contains no repeated links. Once a message traverses ψ_{jk} and reaches node k, it departs the network. Thus, indirect (and direct) feedback is not possible. Further, since the network is tree-structured and all queueing is FCFS with single servers, overtaking is not possible.

Complications due to feedback and overtaking therefore are absent from these networks.

The validity of (1) is established by observing that as a result of the departure process assumption, the jk-arrival process at node j, which is a Poisson process in a random environment by definition, propagates through the network to become a collection of RE-Poisson processes of jk-messages arrivals to and departures from each link in the path ψ_{jk}. The members of this collection are statistically identical to each other and to the jk-arrival process at node j.

Result (2) follows from the departure process assumption and the tree-structured, non-overtaking nature of the network. By definition, the external jk-arrival processes, for all jk, are mutually independent given the environment state. For a fixed jk, nowhere in the network does the flow of jk-messages experience cycles or overtaking. Further, for any $l \in \psi_{jk}$, there exists only one jk-arrival process to queue l. This jk-arrival process is the propagation to queue l of the external jk-arrival process to node j. The external jk-arrival processes are mutually independent given the environment state. It follows from the departure process assumption that the propagations of the members of any subset of this collection of processes are mutually independent given the environment state. Specifically, for each l, this independence property holds among the jk-arrival processes to queue l, for jk such that $l \in \psi_{jk}$.

The third result follows from (1) and (2). For any l, for each jk such that $l \in \psi_{jk}$, on $\{E(t) = i\}$ the jk-arrival process to queue l is a Poisson process with rate $\gamma_{jk}^{(i)}$ and the elements of the collection of such processes at queue l are mutually independent. Thus, on $\{E(t) = i\}$ the composite arrival process to link l is a Poisson process with rate $\lambda_l^{(i)} = \sum_{jk:l \in \psi_{jk}} \gamma_{jk}^{(i)}$. Therefore, the composite arrival process to link l is an RE-Poisson process with rate vector $\boldsymbol{\lambda}_l = [\lambda_l^{(1)}, \lambda_l^{(2)}, \ldots, \lambda_l^{(M)}]$.

■

Corollary 1

Given the departure process assumption, queue l in an RE-network is an RE-queue with arrival rate vector $\boldsymbol{\lambda}_l = [\lambda_l^{(1)}, \lambda_l^{(2)}, \ldots, \lambda_l^{(M)}]$, where $\lambda_l^{(i)} = \sum_{jk:l \in \psi_{jk}} \gamma_{jk}^{(i)}$.

4.4. Mean Network Delay

Theorem 1 and Corollary 1 show that the departure process assumption has the desired effect of simplifying a network (in a random environment) of queues of unknown probabilistic nature to a network of RE-queues with known arrival rates (The service rates are known in any case). In this section, the individual RE-queue results, which are summarized in Section 4.2, are combined to derive a formula for approximate mean network delay.

For each $j \in \{1, 2, \ldots, L\}$, let $\boldsymbol{\lambda}_j = [\lambda_j^{(1)}, \lambda_j^{(2)}, \ldots, \lambda_j^{(M)}]$ and $\boldsymbol{\mu}_j = [\mu C_j, \mu C_j, \ldots, \mu C_j]$ be the arrival rate and service rate vectors, respectively. Assume that $\boldsymbol{\pi}\boldsymbol{\lambda}_j < \boldsymbol{\pi}\boldsymbol{\mu}_j$ and let R_j be the rate matrix for queue j, that is, the minimal solution of

$$R_j^2 \Delta(\boldsymbol{\mu}_j) + R_j[Q - \Delta(\boldsymbol{\lambda}_j + \boldsymbol{\mu}_j)] + \Delta(\boldsymbol{\lambda}_j) = 0.$$

Then the stationary environment/queue length probabilities for queue j are

$$x_{jk} = \boldsymbol{\pi}(I - R_j)R_j^k, \quad k \geq 0$$

and mean virtual time at queue j is

$$S_{Vj} = \boldsymbol{\pi}(I - R_j)\sum_{k=1}^{\infty} R_j^k \sum_{\nu=0}^{k-1} [-(Q - \Delta(\boldsymbol{\mu}_j))^{-1}\Delta(\boldsymbol{\mu}_j)]^{\nu}[-(Q - \Delta(\boldsymbol{\mu}_j))]^{-1}$$
$$\cdot [-(Q - \Delta(\boldsymbol{\mu}_j))^{-1}\Delta(\boldsymbol{\mu}_j)]^{k-\nu}\mathbf{1} + (\boldsymbol{\pi}\boldsymbol{\mu}_j)^{-1}.$$

Define the following additional notation:

W_{jk} = network delay of a jk-message.

T_l = delay of a message at queue l.

T = network delay of a message.

p_{jk} = probability that a message is a jk-message.

Then mean network delay is

$$\begin{aligned} T &= E[T] \\ &= \sum_j \sum_k E[T \mid jk\text{—message}] P(jk\text{—message}) \\ &= \sum_j \sum_k p_{jk} E[W_{jk}] \qquad \text{on } \{jk\text{—message}\} \; T = W_{jk} \\ &= \sum_j \sum_k p_{jk} \sum_{l \in \psi_{jk}} E[T_l] \qquad \text{since } W_{jk} = \sum_{l \in \psi_{jk}} T_l . \end{aligned}$$

The mean arrival rate of jk-messages is $\sum_{i=1}^{M} \pi^{(i)} \gamma_{jk}^{(i)}$ and the mean arrival rate of all messages is $\sum_j \sum_k \sum_{i=1}^{M} \pi^{(i)} \gamma_{jk}^{(i)} = \sum_{i=1}^{M} \pi^{(i)} \sum_j \sum_k \gamma_{jk}^{(i)} = \sum_{i=1}^{M} \pi^{(i)} \gamma^{(i)}$, where $\gamma^{(i)} = \sum_j \sum_k \gamma_{jk}^{(i)}$.

Now let

$$\begin{aligned} p'_{jk} &= \textit{weight} \text{ of } jk\text{-traffic} \\ &= \frac{\sum_{i=1}^{M} \pi^{(i)} \gamma_{jk}^{(i)}}{\sum_{i=1}^{M} \pi^{(i)} \gamma^{(i)}} \end{aligned}$$

and suppose that mean time at queue l, $E[T_l]$, can be approximated by mean virtual time at queue l, $\bar{S}_{Vl}$. Then mean network delay is approximated by

$$\begin{aligned} T' &= \sum_j \sum_k p'_{jk} \sum_{l \in \psi_{jk}} \bar{S}_{Vl} \\ &= \sum_j \sum_k \left(\frac{\sum_{i=1}^{M} \pi^{(i)} \gamma_{jk}^{(i)}}{\sum_{i=1}^{M} \pi^{(i)} \gamma^{(i)}} \right) \sum_{l \in \psi_{jk}} \bar{S}_{Vl} \end{aligned}$$

$$= \sum_{l=1}^{L} \left(\sum_{i=1}^{M} \pi^{(i)} \gamma^{(i)} \right)^{-1} \bar{S}_{Vl} \sum_{j} \sum_{\substack{k \\ jk:\, l \in \psi_{jk}}} \sum_{i=1}^{M} \pi^{(i)} \gamma_{jk}^{(i)}$$

$$= \sum_{l=1}^{L} \frac{\bar{\lambda}_l}{\sum_{i=1}^{M} \pi^{(i)} \gamma^{(i)}} \bar{S}_{Vl},$$

where

$$\bar{\lambda}_l = \sum_{j} \sum_{\substack{k \\ jk:\, l \in \psi_{jk}}} \sum_{i=1}^{M} \pi^{(i)} \gamma_{jk}^{(i)}$$

$$= \sum_{i=1}^{M} \pi^{(i)} \sum_{j} \sum_{\substack{k \\ jk:\, l \in \psi_{jk}}} \gamma_{jk}^{(i)}$$

$$= \sum_{i=1}^{M} \pi^{(i)} \lambda_l^{(i)}.$$

Note that $\lambda_l^{(i)}$ is the composite arrival rate to queue l when the environment state is i.

Thus, $\bar{\lambda}_l$ is the mean composite arrival rate to queue l.

CHAPTER 5

SIMULATION MODEL

A simulation model is employed in this investigation to provide confidence interval and point estimates that are used to assess the accuracy of the mean network delay approximation. Specifically, in this cross-validation effort, the results of the two models are compared to determine the conditions under which the approximation is accurate.

Many details may be included in a simulation model. Decisions to include or exclude details should be guided by the study objectives. The objectives of this study are stated roughly in the previous paragraph. The accuracy of the mean network delay approximation is affected by the departure process assumption and the use of mean virtual time at a queue to approximate mean actual time at a queue. For the comparison, a source is needed of results that may be viewed as those of an *exact* RE-network (without the departure process assumption or the use of virtual time at a queue). Thus, the dynamic hierarchy simulation models no more or less than an RE-network. It is important to note that the simulation results remain model results and are in no way exact results for the dynamic hierarchy, the real system of interest.

5.1. Model Structure

In line with the study objectives, the structure of the simulation model closely reflects that of the analytic model. The components (objects or processes) of the model are:

Message generator: Generate an RE-Poisson process of jk-messages for a given source/destination pair jk.

Message: Represent the movement of a message through the network.

Node: Represent a node with routing tables.

Link: Represent the actions of a link and the attached queue.

Environment: Generate a Markov process of environment state changes.

Nodes are passive and hold routing tables that identify the next node and link in the path of a message. Nodes are connected (conceptually) by links, which, as in the analytic model, are the servers. Each link contains an FCFS message queue. A message enters the network at its source node, waits (possibly) and receives service at the links in its path, and departs the network at its destination node. The environment generates a Markov process of environment states that affect the network as described with respect to the analytic model. Message generators supply messages according to RE-Poisson processes.

5.2. Model Implementation

The dynamic hierarchy simulation model is implemented in the SIMSCRIPT II.5 † simulation programming language [RUSE83] using the process interaction world-view. This implementation and the simulation experiments are done on the IBM system at Virginia Tech using the CMS and MVS operating systems. The program consists of approximately 1025 lines of code (and in-line documentation) plus approximately 170 lines of introductory documentation.

PROCESS types and their attributes in this program are defined in the *PREAMBLE* of the SIMSCRIPT source code given in Appendix G. Additionally, various attributes and routines are defined at the *SYSTEM* level. *ROUTINES* include those for implementation of the environment transition mechanism, input/output, and support of

† The original implementation was written in Simula 67 [BIRG73] but subsequently was translated to SIMSCRIPT II.5.

the experimental design.

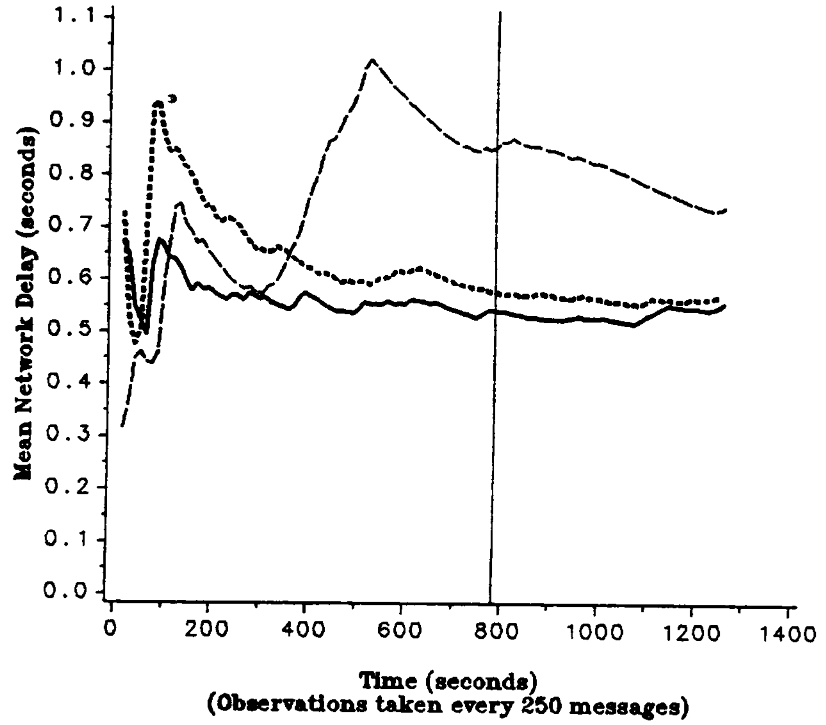
5.3. Experimental Design

The method of batch means is the underlying experimental design for this simulation. Each run consists of a transient period and multiple, consecutive batches.

Estimation of the transient period length is accomplished through simulation runs with the network and parameters given in Appendix A. Six pilot runs are executed. Each has a simulated run length of 1200 seconds. Runs 1, 2, and 3 represent the case where the arrival rates (and the link utilization factors) remain constant (that is, for each link j, $\lambda_j^{(1)} = \lambda_j^{(2)} = \lambda_j$). In runs 4, 5, and 6, the arrival rates vary, with $\lambda_j^{(1)} \neq \lambda_j^{(2)}$ in general, for each j. Within each group, {1,2,3} and {4,5,6}, different runs are defined by using different stream values (or equivalently, using different initial seeds) for the SIMSCRIPT random variate routines. Listings of the stream values used are included in Appendix A.

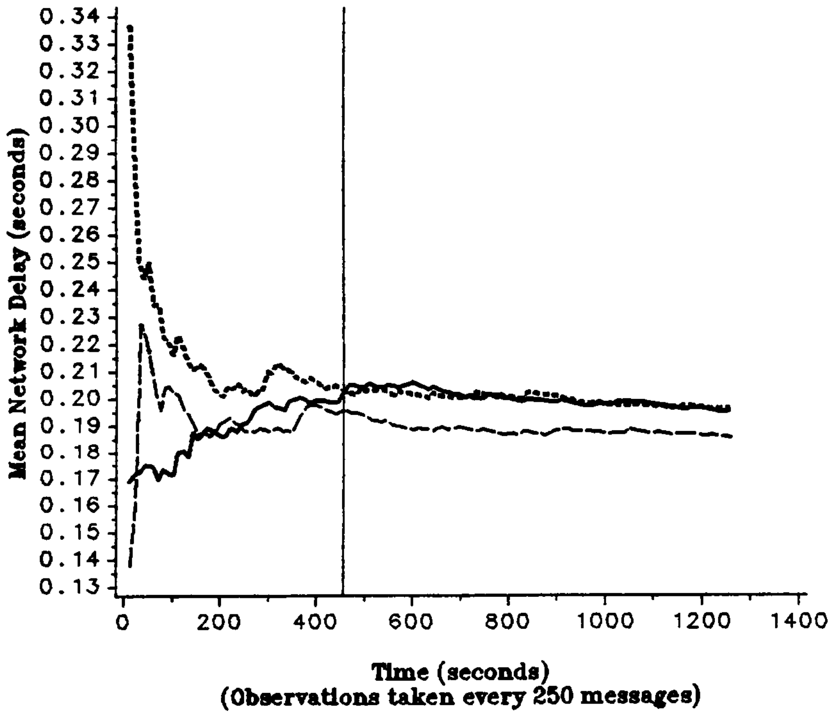
Each pilot run is divided into batches of 250 messages. Within a batch, an observation is accumulated for mean queue length at each queue, mean delay at each queue and mean network delay. At the end of each batch a moving average is calculated (updated) for each of these means. In this way, the run yields a collection of sequences of time/moving average pairs— one sequence for each of the means of interest.

The results of the pilot runs are compiled in graphs that are used to estimate the length of the transient period in general. Two such graphs are shown in Figure 4. Figure 4(a) plots the moving average for mean network delay in runs 1, 2, and 3. Figure 4(b) plots the moving average for mean network delay in runs 4, 5, and 6. Appendix A contains the remaining graphs. Differences among the curves in a single graph are due to statistical variation among multiple runs of the simulation with different random number streams. The time to reach steady state, that is, the transient period length (as visually estimated by the author) is indicated in each graph by a vertical line. The individual transient period length estimates are compiled in Table 1. The average value in this table is 683.2071 seconds and the maximum is 938.6364 seconds.



Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure 4(a)
Moving Average of Mean Network Delay: Pilot Runs 1 to 3



Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

FIGURE 4(b)
Moving Average of Mean Network Delay: Pilot Runs 4 to 6

Table 1. Observed Times to Reach Steady State

performance measure	transient period length
network delay	784.8485
	456.0606
queue length	774.2424
	636.3636
	784.8485
	710.6061
	572.7273
	938.6364
	434.8485
	615.1515
queue delay	790.1515
	636.3636
	742.4242
	715.9091
	562.1212
	848.4848
	721.2121
	572.7273

As a conservative estimate that is expected to be sufficiently large for all cases of interest, 1200 seconds is chosen as the transient period length. For this network, approximately 24300 to 29400 messages pass through the network in that time. Therefore, in subsequent simulation runs, it is assumed that steady state conditions hold upon the departure of the 30000th message. Observations accumulated during the transient period are deleted.

The value 30000 messages is chosen as the batch size as well. Within each batch, an observation is accumulated for the mean and variance of queue length and delay at each link and the mean and variance of network delay. After the final batch, these observations are used to calculate point and confidence interval estimates for mean queue length at each link, mean delay at each link, and mean network delay.

5.4. Credibility Assessment

The steps taken to establish the credibility [BALO87] of the simulation model include: (1) design of the model in a structured manner, (2) implementation in a well known simulation programming language, (3) extensive examination (desk checking) of the implemented code, (4) execution tracing, and (5) stress testing.

Regarding steps (1) and (2), the model was designed in a top-down fashion and subsequently implemented in SIMSCRIPT II.5. The resultant model structure and implementation are described in Sections 3.1 and 3.2, respectively. At various points in the implementation, debugging, and initial experimentation, the code was examined (entirely or in part) to insure its agreement with the model design.

To confirm that the execution of the implemented model proceeds correctly, a trace is generated of selected events in a simulation of the network whose topology and parameters are given in Appendix B. The events traced are: environment activation and transition, message activation, message arrival to a node, message arrival to a link, and message departure from the network. A sample of the trace output is shown in Appendix B.

The event times associated with the first thirty messages to depart are listed in Tables 2 and 3 †. Examination of these tables or the trace output shows:

- 1,3-messages correctly follow the path node 1 $\rightarrow$ link 1 $\rightarrow$ node 2 $\rightarrow$ link 2 $\rightarrow$ node 3. 3,1-messages correctly follow the path node 3 $\rightarrow$ link 3 $\rightarrow$ node 2 $\rightarrow$ link 4 $\rightarrow$ node 1.
- All messages eventually depart the network.

† The specified parameters for this network allow 1,3- and 3,1-messages only. Consequently, no data appears for messages with node 2 as the source or destination.

- No simultaneous arrivals occur.
- No message departs from a link at the same time it arrived at the link (that is, no service time is zero).
- No two messages depart from the same link simultaneously.
- Messages do not overtake one another, which implies the FCFS queueing is handled correctly.

Last, the environment events in the trace show that all intertransition times of the environment are nonzero and no one step transitions of the form $i \rightarrow i$ occur.

Table 2. Event Times for 1,3-Messages

msg ID	A(n1)	A(l1)	D(l1)	A(n2)	A(l2)	D(l2)	A(n3)	D(N)	delay
1	0.0	0.0	0.91	0.91	0.91	1.25	1.25	1.25	1.24846
3	0.53	0.53	1.52	1.52	1.52	2.34	2.34	2.34	1.80793
5	0.81	0.81	1.96	1.96	1.96	2.74	2.74	2.74	1.93865
10	2.9	2.9	2.91	2.91	2.91	4.36	4.36	4.36	1.45912
13	3.88	3.88	3.95	3.95	3.95	4.62	4.62	4.62	0.73519
14	4.02	4.02	4.40	4.40	4.40	4.79	4.79	4.79	0.77213
15	4.76	4.76	5.49	5.49	5.49	5.82	5.82	5.82	1.06234
17	5.65	5.65	6.70	6.70	6.70	7.33	7.33	7.33	1.67495
18	5.86	5.86	6.92	6.92	6.92	7.69	7.69	7.69	1.83226
19	6.33	6.33	8.08	8.08	8.08	8.19	8.19	8.19	1.86819
22	7.30	7.30	8.11	8.11	8.11	9.26	9.26	9.26	1.96292
23	7.7	7.7	8.93	8.93	8.93	9.34	9.34	9.34	1.63542
24	7.77	7.77	10.73	10.73	10.73	10.76	10.76	10.76	2.99026
32	9.83	9.83	10.99	10.99	10.99	11.04	11.04	11.04	1.21596
34	11.07	11.07	11.84	11.84	11.84	12.01	12.01	12.01	0.93625

Table 3. Event Times for 3,1-Messages								
msg ID	A(n3)	A(l3)	D(l3)	A(n2)	A(l4)	D(l4)	A(n1)	D(N)	delay
2	0.0	0.0	0.16	0.16	0.16	0.62	0.62	0.62	0.62131
4	0.56	0.56	0.62	0.62	0.62	0.98	0.98	0.98	0.42273
6	1.0	1.0	1.06	1.06	1.06	1.17	1.17	1.17	0.17217
7	1.56	1.56	2.57	2.57	2.57	3.31	3.31	3.31	1.7492
8	2.27	2.27	3.37	3.37	3.37	3.82	3.82	3.82	1.54549
9	2.68	2.68	4.59	4.59	4.59	5.01	5.01	5.01	2.32609
11	3.25	3.25	6.35	6.35	6.35	6.41	6.41	6.41	3.15370
12 †	3.66	3.66	6.35	6.35	6.35	7.57	7.57	7.57	3.91303
16	5.4	5.4	6.64	6.64	6.64	7.82	7.82	7.82	2.4221
20	7.13	7.13	8.84	8.84	8.84	9.6	9.6	9.6	2.46266
21	7.19	7.19	9.01	9.01	9.01	10.71	10.71	10.71	3.5129
25	7.97	7.97	9.03	9.03	9.03	11.18	11.18	11.81	3.21619
26	8.63	8.63	9.41	9.41	9.41	11.2	11.2	11.2	2.57022
27	8.93	8.93	10.68	10.68	10.68	11.82	11.82	11.82	2.89081
28 †	8.93	8.93	11.4	11.4	11.4	12.55	12.55	12.55	3.62184

A(nk) = arrival time to node k.
A(lj) = arrival time to link j.
D(lj) = departure time from link j.
D(N) = departure time from network.

Stress testing is performed using the topology and parameters given in Appendix C. This network, the environment parameters, capacities, and mean message length are identical to those used for runs 4, 5, and 6 in the determination of the transient period length. The traffic matrices are generated by first noting that with the arrival rates in Section A.2, link 4 experiences the highest utilization, $\rho_4 = 0.65$, and then increasing all rates so that in successive experiments, ρ_4 takes on the values 0.9, 0.95, 0.99, 0.995, and 0.999. The remaining links experience a corresponding increase in utilization.

The simulation output from these experiments does not show any unexpected behavior. As expected, as the network load increases and ρ_4 approaches 1, mean queue length and mean link delay increase rapidly and would become unmanageably large for ρ_4 sufficiently close to 1 (theoretically approaching ∞ as $\rho_4 \rightarrow 1$). These mean values and the

† These lines each contain an event time equal to that of previous line due to finite output precision.

value of mean network delay are given in Table 4. Although mean network delay experiences a seventeen-fold increase, it remains relatively small due to the moderating influence of the other links, whose utilization factors are not close to 1.

Table 4. Network Performance Under Stress Testing

link 4			network
utilization	mean queue length	mean delay	mean delay
0.90	9.9069	1.0966	0.4963
0.95	19.0543	2.0029	0.7707
0.99	165.8173	16.5834	4.7146
0.995	221.5103	22.1765	6.1974
0.999	301.3271	30.1893	8.2966

CHAPTER 6

COMPARISON OF ANALYTIC AND SIMULATION MODELS

As previously discussed, the analytic model is validated by comparison with the results of the simulation model. More precisely, the comparison enables identification of the types of dynamic hierarchy topologies and parameters for which the analytic model yields accurate results.

6.1. Test Networks

The topologies for the ten dynamic hierarchies that are employed as a basis for the comparison are given in Appendix D. Some of these networks are used to test certain hypotheses and others are intended to represent typical dynamic hierarchies. Networks 1, 6, and 7 are used to examine behavior in nonbranching networks in which messages arrive only to the end nodes and are destined only to the end nodes. Networks 1, 2, 3, 4, and 5 are used to examine the effect of an increasingly higher in-degree of links to a node and out-degree of links from a node. Networks 8, 9, and 10 are intended to represent typical dynamic hierarchies. That is, they possess multilevel tree structures and are of moderate size.

Definition of the test networks is completed by specifying various sets of parameters for each network. A set of parameters for a network consists of values for traffic matrices, environment process generator (and the resultant stationary probability vector of the environment process), link capacities, and mean message length. These parameter sets are given along with the topologies in Appendix D. An *experiment* consists of applying the simulation model and analytic model to one topology and parameter set (The terms *topology* and *network* will be used interchangeably where no ambiguity will result).

The parameter sets are used to examine network behavior under conditions of interest and to represent those of typical dynamic hierarchies. All of the parameter sets are used to test the effect of different levels of link utilization. Approximate values of link utilization are listed in Table 5 †. Each network has at least one experiment with medium (0.4 - 0.5) link utilization and at least one with high (0.85 - 0.9) link utilization.

Table 5. Representative Link Utilization Levels

network	parameter set	link utilization
1 to 7	1	0.4444
	2	0.2222
	3	0.8889
8	1	0.4444
	2	0.2222
	3	0.8889
	4	0.3555
	5	0.1807
	6	0.7230
9	1	0.4444
	2	0.4444
	3	0.8889
	4	0.8889
10	1	0.4444
	2	0.4444
	3	0.8889
	4	0.8889

In the parameter sets for networks 1 to 7, the arrival rate matrices are specified so that messages arrive at end nodes and depart from end nodes. For example, in network 3, messages destined for node 1 arrive at nodes 3, 4, and 5, and messages destined for nodes 3, 4, and 5 arrive at node 1. Neither external arrivals nor departures to outside the

† Due to the method of selecting arrival rates and capacities, link utilization factors within a given experiment tend to be approximately equal.

network occur at node 2. In other words, the network handles 3,1-, 4,1-, 5,1-, 1,3-, 1,4-, and 1,5-traffic only. In networks 1, 6, and 7, this type of traffic pattern enables examination of behavior as a single message stream (in each direction) is propagated down a series of links. In networks 1 to 5, this type of pattern enables examination of the effect of combining separate message streams into a new stream that is transmitted across a single link and the effect of splitting a single message stream into different streams that are transmitted across separate links.

An additional aspect that is examined in the experiments on network 9 and network 10 is the effect of increasing rates of environment transitions. For each of these networks, experiments 1 and 3 are performed with a specified environment process generator (which is different for each network). Then, experiments 2 and 4 are performed on each network with an environment process generator that is equal to five times the original generator. In this way, the stationary probability vector of the environment process remains the same as the mean frequency of environment transitions increases.

6.2. Comparative Results

The experimentation and comparison proceed as follows. For each experiment a simulation is run. This run yields point and ninety-five percent confidence interval estimates for mean queue length at each link, mean delay at each link, and mean network delay. Then the analytic model is used to derive numerical values for mean queue length and mean delay at each link and mean network delay for this experiment. A numerical value from the analytic model is considered *accurate* (with respect to the simulation results) if it is contained in the corresponding simulation generated confidence interval. Otherwise, it is considered *inaccurate* and the percentage difference between it and the simulation generated point estimate is computed.

Tabulated results for mean network delay for all test networks are given in Table 6. The columns of this table contain the following data: network number, parameter set number, analytic value for mean network delay (*matrix-geometric network T*), simulation point estimate of mean network delay (*simulation network T*), endpoints of simulation confidence interval for mean network delay (*confidence interval left/right*), indication of whether the analytic value lies within the confidence interval (*T in conf. interval*), and for inaccurate analytic values, the percentage difference between the analytic value and the simulation point estimate relative to the simulation point estimate (*% matrix-geometric T from simulation*).

Table 6. Comparison of Analytic and Simulation Models (Mean Network Delay)

	parameter	matrix geometric	simulation	confidence interval		T in conf.	% matrix geometric T
network	set	network T	network T	left	right	interval	from simulation
1	1	0.1827	0.1840	0.1830	0.1851	no	0.7065
1	2	0.1266	0.1232	0.1205	0.1259	no	2.7957
1	3	1.0766	1.0402	0.9967	1.0837	yes	-
2	1	0.1205	0.1229	0.1223	0.1235	no	1.9528
2	2	0.0842	0.0847	0.0843	0.0851	no	0.5903
2	3	0.6934	0.6804	0.6406	0.7201	yes	-
3	1	0.1168	0.1194	0.1184	0.1203	no	2.1776
3	2	0.0817	0.0826	0.0823	0.0829	no	1.0896
3	3	0.6486	0.6516	0.6228	0.6805	yes	-
4	1	0.0794	0.0817	0.0810	0.0823	no	2.8152
4	2	0.0556	0.0562	0.0559	0.0565	no	1.0676
4	3	0.4548	0.4630	0.4432	0.4829	yes	-
5	1	0.0999	0.1072	0.1063	0.1081	no	6.8097
5	2	0.0683	0.0699	0.0695	0.0702	no	2.2890
5	3	0.6492	0.6768	0.6450	0.7086	yes	-
6	1	0.1917	0.1904	0.1893	0.1915	no	0.6828
6	2	0.1342	0.1334	0.1330	0.1339	no	0.5997
6	3	1.0647	0.9978	0.9641	1.0314	no	6.7048
7	1	0.3343	0.3403	0.3376	0.3429	no	1.7632
7	2	0.2328	0.2342	0.2315	0.2369	yes	-
7	3	1.7200	1.5463	1.4793	1.6133	no	11.2333
8	1	0.0268	0.0270	0.0269	0.0272	no	0.7407
8	2	0.0190	0.0191	0.0190	0.0192	yes	-
8	3	0.1455	0.1492	0.1398	0.1587	yes	-
8	4	0.0237	0.0246	0.0244	0.0247	no	3.6595
8	5	0.0181	0.0185	0.0184	0.0185	no	2.1622
8	6	0.0701	0.0711	0.0690	0.0733	yes	-
9	1	0.0279	0.0284	0.0282	0.0286	no	1.7606
9	2	0.0276	0.0278	0.0276	0.0280	yes	-
9	3	0.1702	0.1685	0.1614	0.1755	yes	-
9	4	0.1444	0.1452	0.1370	0.1534	yes	-
10	1	0.0120	0.0120	0.0120	0.0121	yes	-
10	2	0.0119	0.0119	0.0118	0.0119	yes	-
10	3	0.0655	0.0639	0.0612	0.0666	yes	-
10	4	0.0609	0.0615	0.0596	0.0635	yes	-

Appendix E contains similar tables that list results for the individual links. Each table compares the matrix-geometric results for mean queue length and mean delay for each link with the corresponding confidence interval and point estimates from the simulation model.

Various conclusions can be drawn from these comparisons. The primary positive conclusions are:

(I) The analytic results are accurate for larger and more complex networks due to the effects of mixing (superposing) internal messages flow processes with external message arrival processes.

(II) The analytic results are accurate under high levels of link utilization $(0.85 < \rho_j < 1.0)$.

(III) The analytic results may become more accurate as the frequency of environment transitions increase.

(IV) In cases where the analytic results are marginally inaccurate or marginally accurate, the departure process assumption leads to accurate analytic results for mean queue length but the additional error induced by using mean virtual time at a link to approximate mean (actual) time at a link causes inaccuracy in the analytic results for mean link delay.

The primary negative conclusions are (See also the comment in (IV) regarding mean link delay):

(V) The analytic results are inaccurate for simple networks, such as networks 1 to 7. For tandem networks, such as networks 1, 6, and 7, the inaccuracy becomes more pronounced as the length (number of links from one end to the other) increases.

(VI) The analytic values are inaccurate under low and medium levels levels of link utilization $(0.0 < \rho_j < 0.75)$.

Conclusion (V) is supported by the comparisons for networks 1 to 5, parameter sets 1 and 2, and all but one of the experiments on networks 6 and 7. Except in network 7, parameter set 2 †, analytic mean network delay is inaccurate in all cases. The analytic

† Network 7, parameter set 2 is considered an anomalous case and does not affect the observations and conclusions.

mean queue length and mean link delay results are mixed. In the experiments on networks 2 to 5, analytic mean queue length generally is accurate. In the experiments on networks 1, 6, and 7, this mean generally is inaccurate. Analytic mean link delay is inaccurate in these experiments.

These observed characteristics are attributed to the absence of mixing of internal flow processes and external arrival processes and to the accumulation of error along message paths in these networks. For any jk and path ψ_{jk}, the only node in this path at which external arrivals occur is j. Thus, the flow of jk-messages through the network is never mixed with any external arrival processes. That is, the composite arrival process to each link is either a superposition of multiple flow processes of this type, or one component of a decomposition of a flow process of this type.

It appears that as such processes are propagated through the network without being influenced by subsequently encountered external arrival processes, the composite departure and arrival processes at each link behave less like the RE-Poisson processes addressed in Theorem 1. The departure process assumption does not reflect the behavior of the network with sufficient accuracy in these types of networks.

The most severe inaccuracies occur in the experiments on the tandem networks 1, 6, and 7. In these networks, a single external arrival process (in each direction) passes through the network, behaving increasingly less like an RE-Poisson process at successive links in its path, especially as the path length increases. Thus, the RE-queue model is less valid, which leads to inaccurate results.

Comparisons in one or more experiments for each network except network 10 support conclusion (VI). The utilization levels in these experiments are low to medium. Although the analytic mean queue length and mean link delay results are mixed, analytic mean network delay is inaccurate, and overall, the results for these experiments are considered inaccurate. This conclusion is applicable primarily to relatively simple networks but holds also in more complex networks at low utilization levels.

Although conclusions (V) and (VI) are referred to as *negative*, they are useful and significant in the sense that they identify situations in which the analytic model does not apply. Further, by elimination and in combination with the positive conclusions, conclusions such as (V) and (VI) assist in identifying situations in which the analytic model does apply.

Conclusions (I) and (II) are the positive counterparts of (V) and (VI), respectively. Regarding the effect of the size and complexity of the network, the comparisons indicate a trend toward increasingly accurate results as larger and more complex networks are considered. Networks 8, 9, and 10 represent three steps in increasingly complex, multilevel network topologies. Analytic mean queue length and mean link delay are accurate overall and analytic mean network delay is accurate in three of six experiments (one at low and two at high utilization) on network 8. The analytic results are accurate in all but one of the experiments on networks 9 and 10 (link utilization in these experiments is medium and high).

It is concluded that the higher level of mixing (superposing) of internal flow processes and external arrival process in the more complex networks is responsible for this trend toward increased accuracy. In networks of this type, the flow of jk-messages traverses multiple links and passes through multiple nodes. At each intermediate node in this path of jk-messages, the jk-message flow is mixed with various other internal flow processes and external arrival processes to form the composite arrival process to a specific link. It appears that this mixing, especially with external arrival processes, decreases the distance between the composite arrival process and the assumed RE-Poisson process. This influence offsets the accumulation of error that is observed in the simple networks and thus contributes to more accurate analytic results.

The effect of link utilization level on accuracy of the analytic results is seen in experiments on all of the networks except networks 6 and 7. In all cases (except networks 6 and 7), the analytic results are accurate at a high level of utilization. However, the comparisons do not indicate a trend of increased accuracy in the experiments at medium utilization over the experiments at low utilization.

This behavior is similar to that observed with other heavy traffic approximations in queueing systems (see, for example, the survey discussion in [DISR75]), although the theoretical justification is absent here. It appears that as the probability of reaching the boundary (queue length equal to zero) becomes smaller (as utilization increases), the effects of boundary behavior diminish and the analytic and simulation results become less sensitive to the given distributional assumptions. The consequence is that the two sets of results are in closer agreement, which leads to the conclusion that the analytic values are more accurate.

Conclusion (III) is not supported as well (as the other conclusions) by the comparisons. However, it appears to hold in the cases considered. In two of the networks (9 and 10), parameter sets 1 and 3 use a specified environment process generator. Parameter sets 2 and 4 use a generator that is equal to five times the original generator, thus increasing the transition rates of the environment. The comparisons for these experiments show slight increases in accuracy of the analytic results under higher environment transition rates.

A possible explanation for these increases is that, as in the case of heavy traffic, the analytic and simulation results become less sensitive. Under higher environment transition rates, mean link delay, mean queue length, and mean network delay in the two models approach common values. Thus, the analytic values are considered more accurate.

Last, conclusion (IV) addresses accuracy in the analytic results for the individual links. As reflected by the number of values that lie within the respective confidence intervals, analytic mean queue length tends to be more accurate than analytic mean link delay. This characteristic is most noticeable in experiments for which overall accuracy (or inaccuracy) is marginal. In these marginal cases, the departure process assumption leads to accurate results for mean queue length. However, the additional error introduced by approximating mean actual time at a link with mean virtual time at a link causes the results for mean link delay to be inaccurate.

CHAPTER 7

COMPARISON OF DYNAMIC AND STATIC HIERARCHIES

The final goal of this research is to compare the performance of the dynamic hierarchy with that of the conventional, static hierarchy. This comparison is carried out by first computing mean network delay for the dynamic and static networks corresponding to the sixteen network/parameter set combinations of interest. Mean network delay for the static networks is computed by applying well known results for Jackson networks (as in [KLEL76], for example). Mean network delay for the dynamic networks is computed as previously discussed. Then the two values for each experiment are compared to determine whether the dynamic hierarchy or static hierarchy yields lower mean delay. Additionally, in cases where mean delay in the dynamic hierarchy is higher, the percentage difference between the two values is of interest.

7.1. Test Networks

The dynamic hierarchy topologies and parameter sets (not including link capacities) used in this comparison are those from the previous comparison (of the analytic and simulation models) for which the analytic mean network delay results are accurate. So, the networks (topologies) used are 1 to 5, 7, and 8 to 10.

Definition of comparable static hierarchies proceeds as follows. Given a dynamic hierarchy n, the comparable topology for static hierarchy n is chosen as configuration 1 (one) of the dynamic topology. For each experiment on dynamic hierarchy n, which includes a specified set of traffic matrices, the comparable traffic matrix for the corresponding experiment on static hierarchy n is defined as the mean, over all environment states, of the dynamic hierarchy matrices. That is, let $\Gamma^{(1)}, \Gamma^{(2)}, \ldots, \Gamma^{(M)}$ be the dynamic hierarchy traffic matrices, π be the stationary probability vector of the environment process, and $\Gamma = [\gamma_{jk}]$ be the static hierarchy traffic matrix. Then,

$$\Gamma = \sum_{i=1}^{M} \pi^{(i)} \Gamma^{(i)}.$$

The static hierarchy is modeled as a Jackson network. Link l has composite arrival rate

$$\lambda_l = \sum_{j} \sum_{\substack{k \\ jk: l \in \psi_{jk}}} \gamma_{jk},$$

where ψ_{jk} is the path from node j to node k. These arrival rates, mean message length μ^{-1}, and a total capacity C_{TOT} are used to perform square-root capacity assignments [KLEL64,KLEL76] for the static hierarchy. Use of square-root capacities insures that the mean delay derived for the static hierarchy is optimal (under the requisite assumption). Total capacity is chosen for each network and parameter set so that link utilization levels lie in the same range as those of the corresponding network and parameter set in the comparisons of the analytic and simulation models. The topologies and parameter sets, including capacities, for the static networks are given in Appendix F.

Note that, in addition, these capacities are used to complete the definition of the dynamic hierarchy parameter sets for this comparison.

7.2. Comparative Results

The numerical results of the comparisons are given in Table 7. Originally certain dynamic hierarchies were expected to yield lower mean network delay than their counterpart static hierarchies. The results show that this is not the case in the experiments on which the comparisons are based. However, viewing percentage differences between dynamic hierarchy and static hierarchy mean delay (the last column of Table 7) that are less than one as essentially zero, performance of the dynamic and static networks is equal in four cases †. In other words, these four cases represent dynamic hierarchies for which performance (as measured by mean network delay) is as good as that of their static counterparts. Further, in four additional experiments, mean delay in the dynamic hierarchy falls within ten percent of mean delay in the comparable static hierarchy.

Note that the four cases in which performance of the dynamic and static hierarchies is considered equal represent cases where, in addition, mean delay is lowest. The significance of this observation remains as a possible subject of future investigation.

Table 7. Comparison of Dynamic and Static Hierarchy

network	parameter set	dynamic hierarchy (DH) mean delay	static hierarchy (SH) mean delay	percent DH delay above SH delay
1	3	0.5033	0.42444	18.5798
2	3	0.3200	0.27793	15.1369
3	3	0.2826	0.26049	8.4878
4	3	0.1925	0.17251	11.5877
5	3	0.2506	0.20182	24.1701
6	(none)	-	-	-
7	2	0.1851	0.18536	n/a
8	2	0.0185	0.01847	0.1624
8	3	0.1417	0.12927	9.6155
8	6	0.0680	0.05063	34.3077
9	2	0.0270	0.02681	0.7087
9	3	0.1655	0.13406	23.4522
9	4	0.1413	0.13406	5.4006
10	1	0.0111	0.01101	0.8174
10	2	0.0110	0.01101	0.0908
10	3	0.0607	0.05507	10.2234
10	4	0.0565	0.05507	2.5967

It is conjectured that for the networks and parameter sets considered, the dynamic hierarchy does not yield lower mean network delay than the static hierarchy due to the averaging of mean delay over all environment states. In general, conditional mean delay

† Network 7, parameter set 2, in which the dynamic hierarchy exhibits slightly lower mean network delay, is considered an anomalous case and does not affect the observations and conclusions.

given the environment state exhibits the desired property of being lower under certain environment states and configurations. However, in the remaining configurations, conditional mean delay is higher. When the conditioning is removed, average performance behaves as previously discussed.

The primary conclusion that can be drawn from this comparison is that situations exist in which a dynamic hierarchy may be employed instead of a static hierarchy while incurring little (less than ten percent) or no performance penalty. In these situations, benefits of the dynamic hierarchy— flexibility, survivability, and other characteristics that are achievable through distributed operation with global control— may be realized together with performance equal or nearly equal to that of the conventional, static architecture.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

In assessing the relative costs and benefits of the dynamic hierarchy, a new concept in reconfigurable network architectures, an analytic, queueing model and a discrete event simulation model are defined. The simulation model is used in a cross-validation effort to establish the conditions under which the analytic model is valid. The analytic model is used to derive estimates of performance, as measured by mean network delay, in the dynamic hierarchy.

The dynamic hierarchy is modeled as a network of $M/M/1$ queues in a random (Markov) environment, or RE-network. Introduction of an additional assumption, the departure process assumption, enables the links in the network to be analyzed separately as $M/M/1$ queues in a random environment, or RE-queues.

Two variants of the RE-queue are defined as applicable models of the individual links. The first, which the analysis described herein employs, is referred to as the link model with instantaneous reconfigurations and is appropriate both when the time to perform a network configuration change is negligible and as a performance baseline for the second variant. The second is referred to as the link model with reconfiguration periods. It represents a more realistic model in cases where a network configuration change takes a nonnegligible amount of time.

The individual links are analyzed by applying results from the literature on RE-queues for queue length probabilities and mean virtual waiting time. Mean virtual time at a link (time in the queue plus service time) is used to approximate mean actual time at a link. These results for the individual links then are combined to produce the desired formula for mean network delay.

Using a collection of dynamic hierarchy topologies with various sets of network parameters, results are obtained from the analytic and simulation models. Then the two sets of results are compared. An analytic value is considered accurate if it lies within the corresponding, simulation generated confidence interval. On the basis of this comparison, the following conclusions are drawn:

(I) The analytic results are accurate for larger and more complex networks due to the effects of mixing (superposing) internal messages flow processes with external message arrival processes.

(II) The analytic results are accurate under high levels of link utilization $(0.85 < \rho_j < 1.0)$.

(III) The analytic results may become more accurate as the frequency of environment transitions increase.

(IV) In cases where the analytic results are marginally inaccurate or marginally accurate, the departure process assumption leads to accurate analytic results for mean queue length but the additional error induced by using mean virtual time at a link to approximate mean (actual) time at a link causes inaccuracy in the analytic results for mean link delay.

(V) The analytic results are inaccurate for simple networks, such as networks 1 to 7. For tandem networks, such as networks 1, 6, and 7, the inaccuracy becomes more pronounced as the length (number of links from one end to the other) increases.

(VI) The analytic values are inaccurate under low and medium levels levels of link utilization $(0.0 < \rho_j < 0.75)$.

Further, the positive effect of higher network complexity offsets the negative effect of a low utilization level and the positive effect of a high utilization level offsets the negative effect of low network complexity in many cases.

These conclusions lead to the following characterization of the conditions under which the analytic results (particularly mean network delay) are expected to be accurate:

> The analytic results are accurate for multilevel networks of at least medium complexity and under high levels of link utilization. Accuracy is maintained for simple networks under sufficiently high link utilization levels and for sufficiently complex networks under low and medium link utilization levels.

Last, performance of the dynamic hierarchy is compared with performance of the static hierarchy. The test networks of interest are the previously discussed topologies and parameter sets for which the analytic results are accurate. Comparable, static test networks are defined by fixing each dynamic topology in a single configuration and then deriving comparable parameter sets. Link capacities are assigned according to the square-root strategy. The comparison then is based on values of mean network delay that are computed for each dynamic and static topology and parameter set.

The conclusion that is drawn from this comparison is that situations exist in which a dynamic hierarchy may be employed while incurring little or no performance penalty (with respect to performance of a static hierarchy). Therefore, the benefits of a dynamic architecture— flexibility, survivability, and other characteristics— may be realized in these situations together with performance equal or nearly equal to that of a static hierarchy.

Dynamic hierarchies with point-to-point, logically variable topologies are the networks of primary interest in the research discussed herein. The results and conclusions, in addition, can be viewed as characterizing a performance baseline for dynamic hierarchies of other topological classes. In particular, dynamic hierarchies with topologies based on a common transmission medium (for example, a bus), in which (logical) link capacities may be considered variable, are expected to exhibit performance increases over the dynamic hierarchies considered here.

Two directions seem useful and appropriate for future research. First, the research performed thus far addresses mean performance only. This research illustrates a method by which other characteristics of dynamic hierarchy performance, such as variance of network delay, may be analyzed. Second, as noted in a previous section, the RE-network model is expected to yield less accurate results when applied to dynamic hierarchies with topological bases other than the point-to-point, logically variable type. Further use of the RE-model to analyze network performance with additional capabilities for reconfiguration (for example, reassigned link capacities) merits further investigation. The model is applicable but interpretation of the results may be more difficult.

REFERENCES

BALO84. Balci, O. and R. G. Sargent, 'A Bibliography on the Credibility Assessment and Validation of Simulation and Mathematical Models,' *Simuletter* **15**(3), pp. 15-27 (July 1984).

BALO85. Balci, O. and R. E. Nance, 'Formulated Problem Verification as an Explicit Requirement of Model Credibility,' *Simulation* **42**(2), pp. 76-86 (August 1985).

BALO86. Balci, O., 'Guidelines for Successful Simulation Studies, Parts I and II,' Technical Report TR-85-2, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia (September 1986).

BALO87. Balci, O., 'Credibility Assessment of Simulation Results: The State of the Art,' In *Proc. of the Conf. on Methodology and Validation (1987 Eastern Simulation Conf's.)*, Orlando, Florida, pp. 19-25, The Society for Computer Simulation (April 6-9, 1987).

BANJ84. Banks, J. and J. S. Carson, II, *Discrete-Event System Simulation*, Englewood Cliffs, Prentice-Hall (1984).

BANJ87. Banks, J., D. Gerstein, and S. P. Searles, 'Modeling Processes, Validation, and Verification of Complex Simulations: A Survey,' In *Proc. of the Conf. on Methodology and Validation (1987 Eastern Simulation Conf's.)*, Orlando, Florida, pp. 13-18 (April 6-9, 1987).

BARA76. Barbour, A. D., 'Networks of Queues and the Method of Stages,' *Advances in Applied Probability* **8**(3), pp. 584-591 (September 1976).

BASF75. Baskett, F., K. M. Chandy, R. H. Muntz, and F. G. Palacios, 'Open, Closed, and Mixed Networks of Queues with Different Classes of Customers,' *Journal of the*

ACM **22**(2), pp. 248-260 (April 1975).

BHAU86. Bhat, U. N. and R. E. Nance, 'A Queueing Network Analysis of Dynamic Reconfigurability in an Hierarchical Information Network,' Presented at *Special Interest Conf. on Queueing Networks and Their Applications*, New Brunswick, NJ **9**, ORSA Technical Section/ TIMS College on Applied Probability, (Also TR-86-35, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1986) (January 7-9, 1987).

BIRG73. Bitrwistle, G. M., O.-J. Dahl, B. Myhrhaug, and K. Nygaard, *Simula Begin*, New York, Petrocelli/Charter (1973).

CINE75. Cinlar, E., *Introduction to Stochastic Processes*, Prentice-Hall, Englewood Cliffs (1975).

DADH82. Daduna, H., 'Passage Times for Overtake-Free Paths in Gordon-Newell Networks,' *Advances in Applied Probability* **14**(3), pp. 672-686 (Spetember 1982).

DISR75. Disney, R. L., 'Random Flow in Queueing Networks: A Review and Critique,' *AIIE Trans.* **7**(3), pp. 268-288 (September 1975).

DISR81. Disney, R. L., 'Queueing Networks,' In *Proc. of Symposia in Applied Mathematics* **25**, *Operations Research: Mathematics and Models*, pp. 53-83 (1981).

DISR85. Disney, R. L. and D. König, 'Queueing Networks: A Survey of Their Random Processes,' *SIAM Review* **27**(3), pp. 335-403 (Spetember 1985).

DISR87. Disney, R. L. and P. C. Kiessler, *Traffic Processes in Queueing Networks: A Markov Renewal Approach*, Baltimore, Johns Hopkins University Press (1987).

EISM63. Eisen, M. and M. Tainter, 'Stochatic Variations in Queueing Processes,' *Operations Research* **11**(6), pp. 922-927 (November-December 1963).

EVAR64. Evans, R. V., 'Queueing When Jobs Require Several Services Which Need Not Be Sequenced,' *Management Science* **10**(2), pp. 298-315 (January 1964).

EVAR67. Evans, R. V., 'Geometric Distribution in Some Two-Dimensional Queueing Systems,' *Operations Research* **15**(5), pp. 830-846 (September-October 1967).

GERM77. Gerla, M. and L. Kleinrock, 'On the Topological Design of Distributed Computer Networks,' *IEEE Trans. on Communications* **COM-25**(1), pp. 48-60 (January 1977).

GORW67. Gordon, W. J. and G. F. Newell, 'Closed Queuing Systems With Exponential Servers,' *Operations Research* **15**(2), pp. 254-265 (March-April 1967).

JACJ57. Jackson, J. R., 'Networks of Waiting Lines,' *Operations Research* **5**(4), pp. 518-521 (August 1957).

JACJ63. Jackson, J. R., 'Jobshop-Like Queueing Systems,' *Management Science* **10**(1), pp. 131-142 (October 1963).

KELF75. Kelly, F. P., 'Networks of Queues With Customers of Different Types,' *Journal of Applied Probability* **12**(3), pp. 542-554 (September 1975).

KELF76. Kelly, F. P., 'Networks of Queues,' *Advances in Applied Probability* **8**(2), pp. 416-432 (June 1976).

KELF79. Kelly, F. P., *Reversibility in Stochastic Networks*, New York, John Wiley & Sons (1979).

KELF82. Kelly, F. P., 'Networks of Quasi-Reversible Nodes,' pp. 4-26 in *Applied Probability-Computer Science, The Interface: Proc. of the ORSA-TIMS Boca Raton Symp.*, ed. R. L. Disney and T. Ott, Birkhauser, Cambridge (1982).

KELF83. Kelly, F. P., 'Sojourn Time in Closed Queueing Networks,' *Advances in*

Applied Probability **15**(3), pp. 638-656 (September 1983).

KLEL64. Kleinrock, L., *Communication Nets Stochastic Message Flow and Delay,* McGraw-Hill, New York (1964).

KLEL70. Kleinrock, L., "Analyic and Simulation Methods in Computer Network Design," In *AFIPS Conf. Proc. SJCC 1970* **36**, pp. 569-579 (1970).

KLEL73. Kleinrock, L., "Scheduling, Queueing, and Delays in Time-Shared Systems and Computer Networks," pp. 95-141 in *Computer-Communication Networks,* ed. N. Abramson and F. F. Kuo, Prentice-Hall (1973).

KLEL76. Kleinrock, L., *Queuing Systems Volume II: Computer Applictions,* John Wiley & Sons, New York (1976).

KOMM78. Komatsu, M., N. Nakanishi, H. Sanada, and Y. Tezuka, "Extended Optimum Channel Capacity Assignment Problems for Message-Switching Networks," In *Proc. Fourth International Conf. on Computer Communications,* pp. 487-492 (September 1978).

LAWA82. Law, A. M. and W. D. Kelton, *Simulation Modeling and Analysis,* New York, McGraw-Hill (1982).

LEMA77. Lemoine, A. J., "Networks of Queues— A Survey of Equilibrium Analysis," *Management Science* **24**(4), pp. 464-481 (December 1977).

LEMA79. Lemoine, A. J., "On Total Sojourn Time in Networks of Queues," *Management Science* **25**(10), pp. 1034-1035 (October 1979).

LIND82. Linton, D. and P. Purdue, "An M/G/∞ Queue with Catastrophes," *Opsearch* **9**(3), pp. 183-186 (1982).

MARK76. Maruyama, K. and D. T. Tang, "Discrete Link Capacity Assignment in

Communication Networks,' In *Proc. 3rd International Conf. on Computer Communication*, Toronto, pp. 92-97, International Council for Computer Communication (August 3-6, 1976).

MEIB71a. Meister, B., H. R. Muller, and H. R. Rudin, Jr., 'Optimization of a New Model for Message-Switching Networks,' *Conf. Record IEEE Conf. on Communications ICC 71*, pp. 39/16-39/21 (June 1971).

MEIB71b. Meister, B., H. R. Muller, and H. R. Rudin, Jr., 'New Optimization Criteria for Message-Switching Networks,' *IEEE Trans. on Communication Technology* **COM-19**(3), pp. 256-260 (June 1971).

MEIB72. Meister, B., H. R. Muller, and H. R. Rudin, Jr., 'On the Optimiztaion of Message-Switching Networks,' *IEEE Trans. on Communications* **COM-20**(1), pp. 8-14 (February 1972).

MELB82. Melamed, B., 'Sojourn Times in Queueing Networks,' *Mathematics of Operations Research* **7**(2), pp. 223-244 (May 1982).

MILD81. Miller, D. R., 'Steady-State Algorithmic Analysis of M/M/c Two-Priority Queues with Heterogeneous Rates,' pp. 207-225 in *Applied Probability-Computer Science, The Interface: Proc. of the ORSA-TIMS Boca Raton Symp.*, ed. R. L. Disney and T. Ott., Birkhauser, Cambridge (1981).

MITI79. Mitrani, I., 'A Critical Note on a Result of Lemoine,' *Management Science* **25**(10), pp. 1026-1027 (October 1979).

MOOR83. Moose, R. L. Jr., 'Link Capacity Assignment in Dynamic Hierarchical Networks for Real-Time Applications,' Masters thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia

(1983).

NAGS86. Nagappan, S., 'Protocol Design and Analysis for a Dynamic Hierarchical Local Area Network,' Masters thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia (1986).

NANR81. Nance, R. E., 'Model Representation in Discrete Event Simulation: The Conical Methodology,' Technical Report CS81003-R, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia (March 1981).

NANR82. Nance, R. E., 'Data Transfer Architectures: Development of the Capability for Comparative Evaluation,' NSWC TR 82-345, Naval Surface Weapons Center, Dahlgren, Virginia (March 1982).

NANR85. Nance, R. E. and R. L. Moose, Jr., 'Link Capacity Assignment in Dynamic Hierarchical Networks,' *Submitted for publication* (1985).

NANR87a. Nance, R. E., 'The Conical Methodology: A Framework for Simulation Model Development,' In *Proc. of the Conf. on Methodology and Validation (1987 Eastern Simulation Conf's.)*, Orlando, pp. 38-43, The Society for Computer Simulation (April 6-9, 1987).

NANR87b. Nance, R. E., R. L. Moose, Jr., and R. V. Foutz, 'A Statistical Technique for Comparing Heuristics: An Example from Capacity Assignment Strategies in Computer Network Design,' *Comm. ACM* **30**(5), pp. 430-442 (May 1987).

NEUM69. Neuts, M. F., 'A Queue Subject to Extraneous Phase Changes,' *Advances in Applied Probability* **3**, pp. 78-119 (1971).

NEUM77a. Neuts, M. F., 'The M/M/1 Queue with Randomly Varying Arrival and

Service Rates,' Technical Report No. 77/11, University of Delaware, Department of Statistics and Computer Science (1977).

NEUM77b. Neuts, M. F., 'Algorithms for the Waiting Time Distributions Under Various Queue Disciplines in the M/G/1 Queue with Service Time Distributions of Phase Type,' pp. 177-197 in *Algorithmic Methods in Probability, TIMS Studies in the Management Sciences*, ed. M. F. Neuts, North-Holland, Amsterdam (1977).

NEUM78a. Neuts, M. F., 'Markov Chains with Applications in Queueing Theory, Which Have a Matrix-Geometric Invariant Probability Vector,' *Advances in Applied Probability* **10**(1), pp. 185-212 (June 1978).

NEUM78b. Neuts, M. F., 'Further Results on the M/M/1 Queue with Randomly Varying Rates,' Technical Report 78/4, University of Delaware, Deoartment of Statistics and Computer Science (1978).

NEUM80. Neuts, M. F., 'The Probabilistic Significance of the Rate Matrix in Matrix-Geometric Invariant Vectors,' *Journal of Applied Probability* **17**(1), pp. 291-296 (March 1980).

NEUM81a. Neuts, M. F., 'The c-Server Queue with Constant Service Times and a Versatile Markovian Arrival Process,' In *Applied Probability-Computer Science, The Interface: Proc. of the ORSA-TIMS Boca Raton Symp.*, Cambridge, pp. 31-70, Birkhauser (1981).

NEUM81b. Neuts, M. F., *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*, Baltimore, John Hopkins University Press (1981).

PURP74. Purdue, P., 'The M/M/1 Queue in a Markovian Environment,' *Operations Research* **22**(3), pp. 562-569 (May-June 1974).

PURP78. Purdue, P., 'The Single Server Queue in a Random Environment,' In *Proc. Third Symp. on Operations Research: Methods of Operations Research* **33**, pp. 363-372 (September 1978).

PURP81. Purdue, P. and D. Linton, 'An Infinite-Server Queue Subject to an Extraneous Phase Process and Related Models,' *Journal of Applied Probability* **18**, pp. 236-244 (1981).

RAMV80. Ramaswami, V., 'The N/G/1 Queue and its Detailed Analysis,' *Advances in Applied Probability* **12**(1), pp. 222-261 (March 1980).

RUSE83. Russell, E. C., *Building Simulation Models with SIMSCRIPT II.5*, Los Angeles, CACI (1983).

SCHM77. Schwartz, M., *Computer Communication Network Design and Analysis*, Englewood Cliffs, Prentice-Hall (1977).

SCHM87. Schwartz, M., *Telecommunication Networks: Protocols, Modeling, and Analysis*, Reading, Addison-Wesley (1987).

SCOM64. Scott, M., 'A Study of Some Single-Counter Queueing Processes,' Ph.D. dissertation, University of North Carolina at Chapel Hill (1964).

SHAR75. Shannon, R. E., *Systems Simulation: The Art and Science*, Englewood Cliffs, Prentice-Hall (1975).

SIMB79. Simon, B. and R. D. Foley, 'Some Results on Sojourn Times in Acyclic Jackson Networks,' *Management Science* **25**(10), pp. 1027-1034 (October 1979).

TANS81. Tanenbaum, A. S., *Computer Networks*, Englewood Cliffs, Prentice-Hall (1981).

WALJ80. Walrand, J. and P. Varaiya, 'Sojourn Times and the Overtaking Condition in Jacksonian Networks,' *Advances in Applied Probability* **12**(4), pp. 1000-1018

(December 1980).

WALV69. Wallace, V. L., 'The Solution of Quasi Birth and Death Processes Arising from Multiple Access Computer Systems,' Ph.D. dissertation, Systems Engineering Lab., University of Michigan (1969).

YECU71. Yechiali, U. and P. Naor, 'Queueing Problems with Heterogeneous Arrivals and Service,' *Operations Research* **19**(3), pp. 722-734 (May-June 1971).

YECU73. Yechiali, U., 'A Continuous-Type Birth-and-Death Process Defined on a Continuous-Time Markov Chain,' *Operations Research* **21**(2), pp. 604-609 (1973).

APPENDIX A
NETWORK DATA FOR TRANSIENT PERIOD LENGTH DETERMINATION

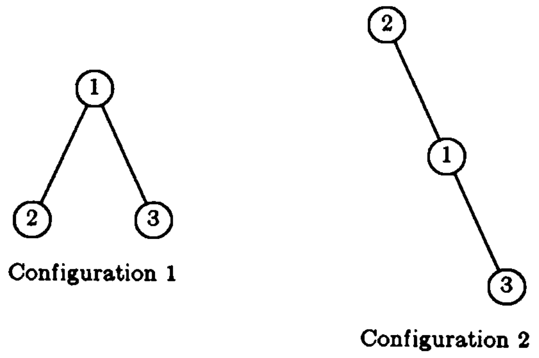


Figure A.1. Network for Transient Period Determination

Pilot Runs 1 to 3

Number of configurations: 2
Number of nodes: 3

General traffic, constant $\rho_j^{(i)} = \rho_j$ for all links j.

Batch size: 250 messages.
Simulated time: 1200.0 seconds.

Environment process generator:

$$Q_E = \begin{bmatrix} -2.0 & 2.0 \\ 2.0 & -2.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5,\ 0.5]$$

Table A.1. Link Numbers and Capacities— Pilot Runs 1 to 3			
link	source node	destination node	capacity
1	1	2	1000.0
2	2	3	1000.0
3	3	2	1000.0
4	2	1	1000.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \Gamma^{(2)} = \begin{bmatrix} 0.0 & 2.0 & 6.0 \\ 4.0 & 0.0 & 3.0 \\ 4.0 & 2.0 & 0.0 \end{bmatrix}$$

SIMSCRIPT stream usage:
- stream1: environment state
- stream2: environment residence time
- stream3: message interarrival time
- stream4: message length

Table A.2 Stream Values— Pilot Runs 1 to 3				
run	stream1	stream2	stream3	stream4
1	1	2	3	4
2	5	6	7	8
3	3	4	5	6

Pilot Runs 4 to 6

Number of configurations: 2
Number of nodes: 3

General traffic, variable $\rho_j^{(i)}$ for all links j.

Batch size: 250 messages
Simulated time: 1200.0 seconds.

Environment process generator:

$$Q_E = \begin{bmatrix} -2.0 & 2.0 \\ 2.0 & -2.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5, 0.5]$$

Table A.3. Link Numbers and Capacities— Pilot Runs 4 to 6			
link	source node	destination node	capacity
1	1	2	1000.0
2	2	3	2000.0
3	3	2	2000.0
4	2	1	1000.0

Mean message length: $\mu^{-1} = 100.0$ bits

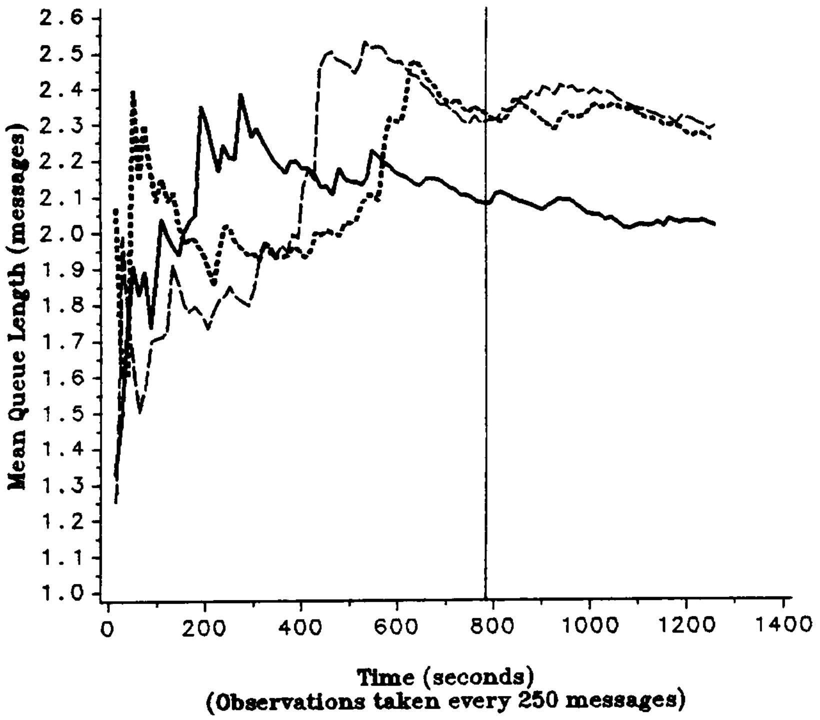
Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 3.0 & 8.0 \\ 4.0 & 0.0 & 2.0 \\ 9.0 & 5.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 2.0 & 5.0 \\ 2.0 & 0.0 & 1.0 \\ 5.0 & 3.0 & 0.0 \end{bmatrix}$$

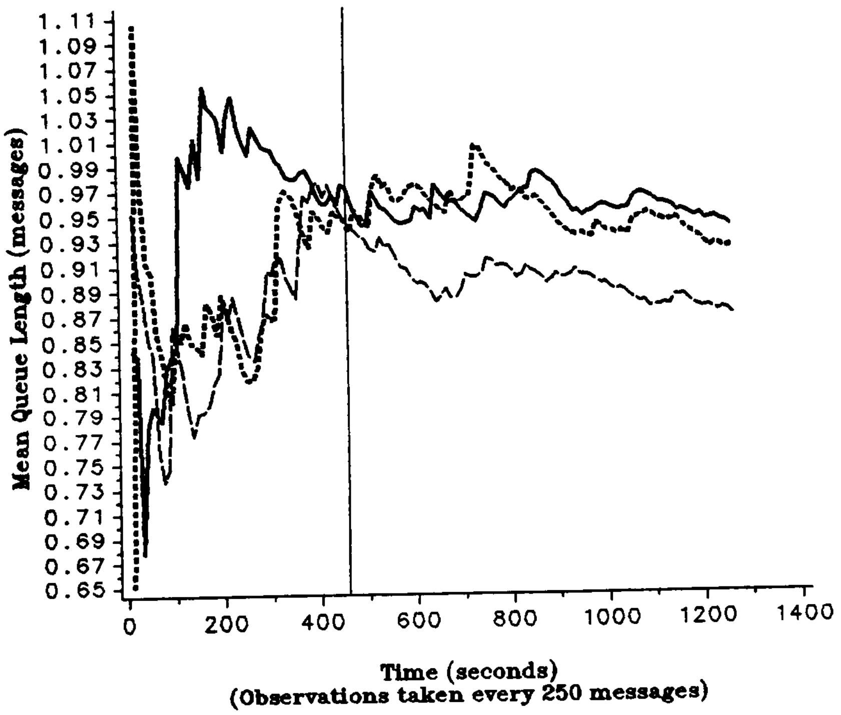
SIMSCRIPT stream usage: Same as above.

Table A.4 Stream Values— Pilot Runs 4 to 6				
run	stream1	stream2	stream3	stream4
4	1	2	3	4
5	5	6	7	8
6	3	4	5	6



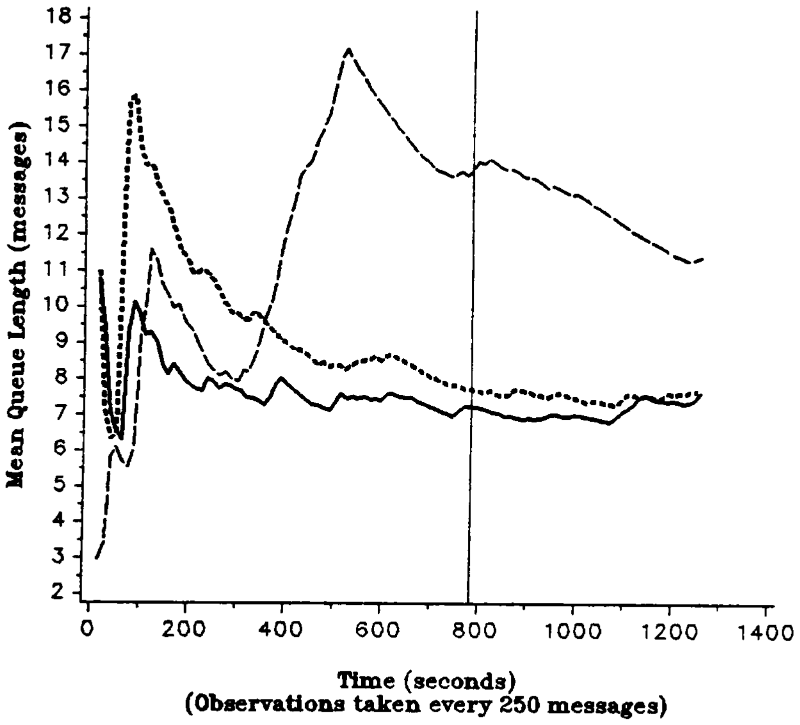
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.2
Moving Average of Mean Queue Length, Queue 1: Pilot Runs 1 to 3



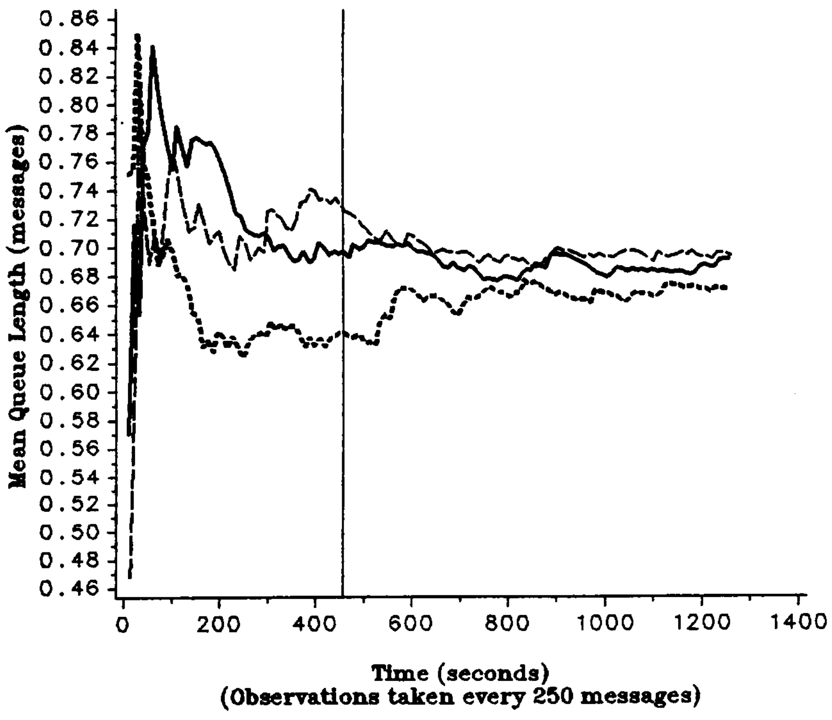
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.3
Moving Average of Mean Queue Length, Queue 1: Pilot Runs 4 to 6



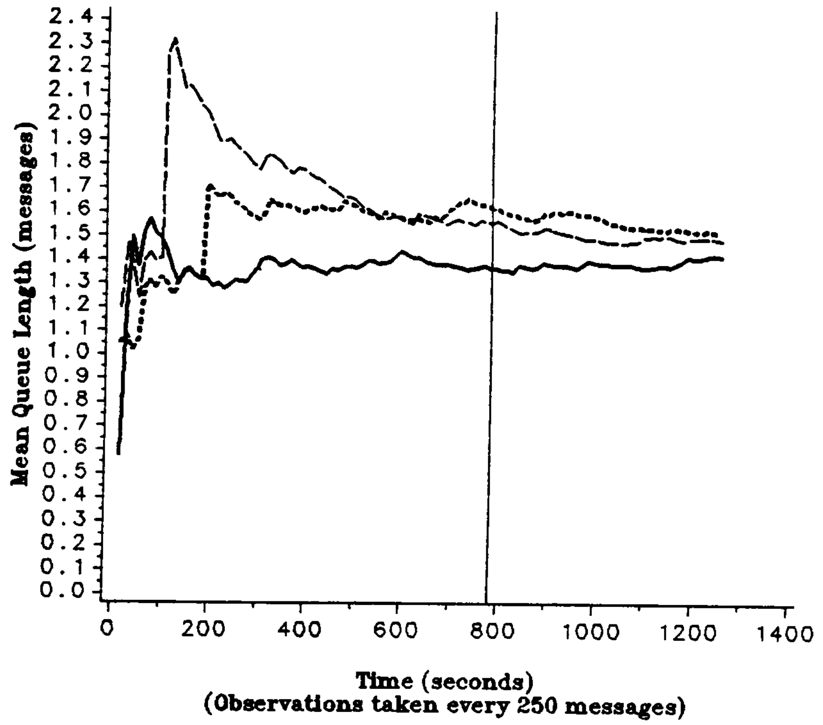
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.4
Moving Average of Mean Queue Length, Queue 2: Pilot Runs 1 to 3



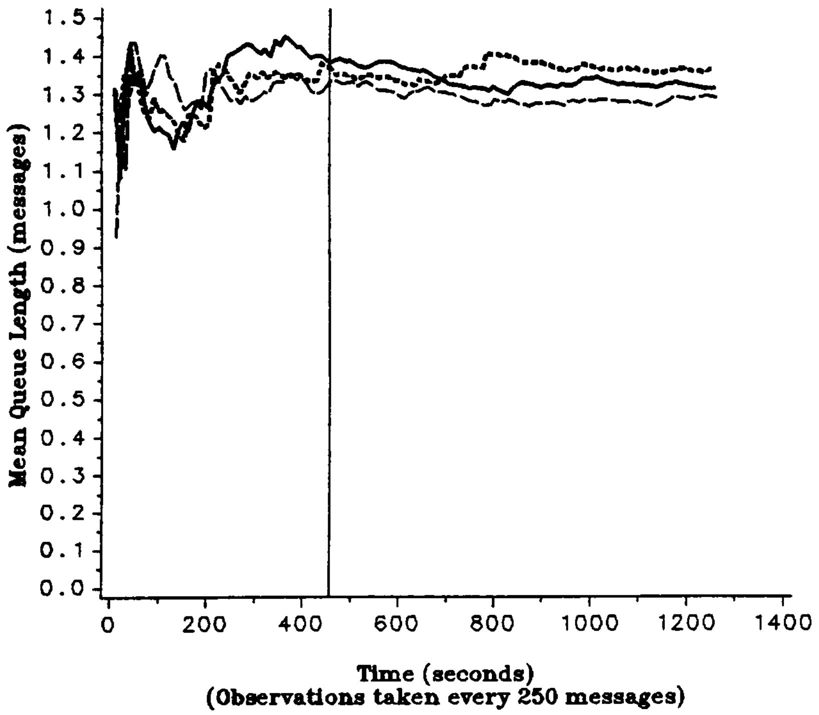
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.5
Moving Average of Mean Queue Length, Queue 2: Pilot Runs 4 to 6



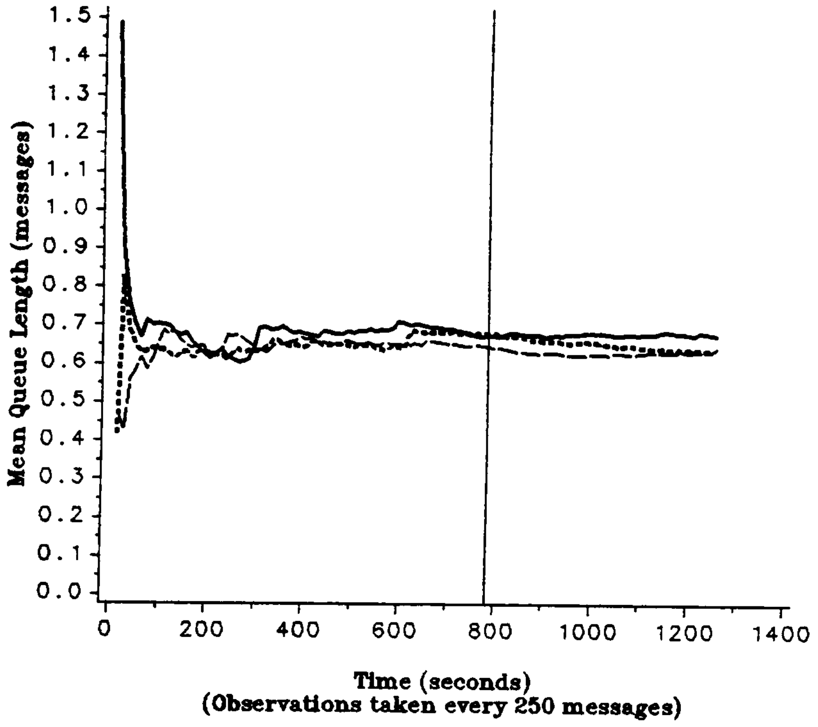
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.6
Moving Average of Mean Queue Length, Queue 3: Pilot Runs 1 to 3



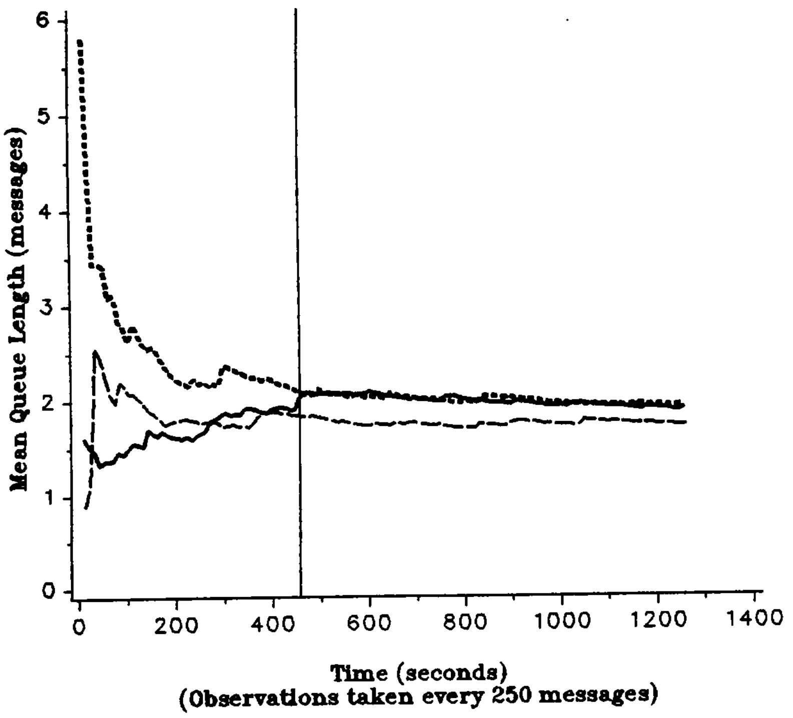
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.7
Moving Average of Mean Queue Length, Queue 3: Pilot Runs 4 to 6



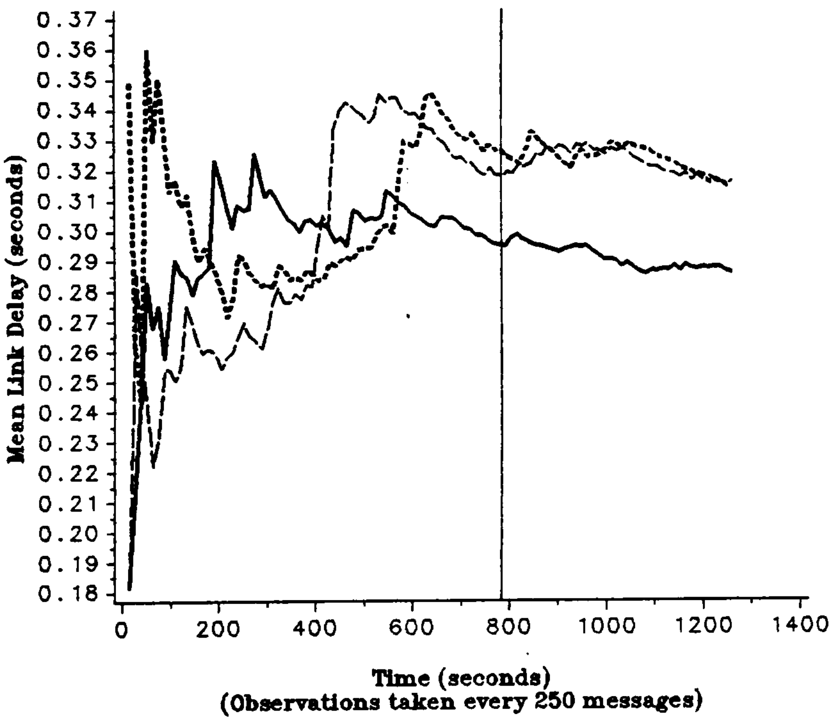
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.8
Moving Average of Mean Queue Length, Queue 4: Pilot Runs 1 to 3



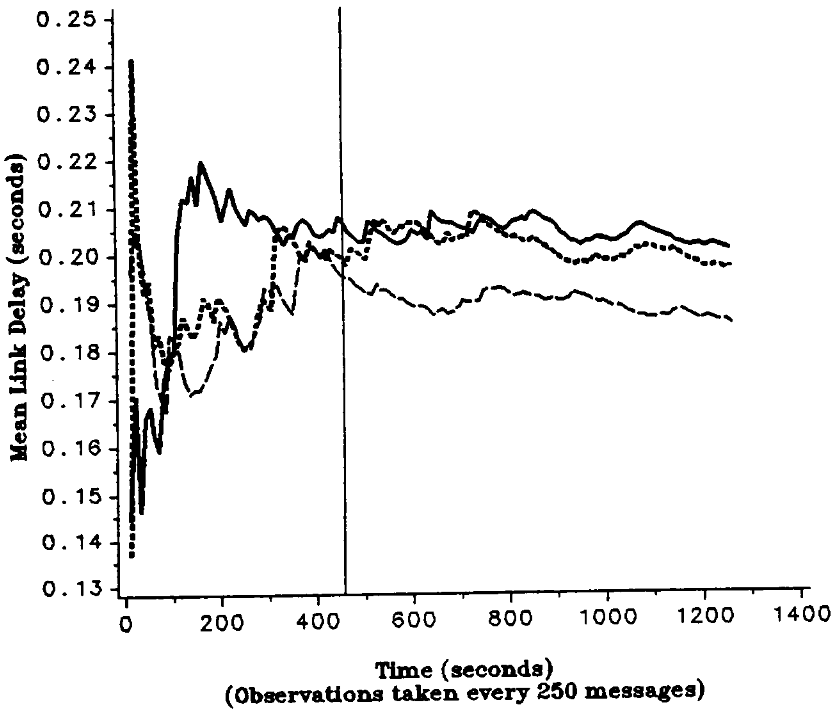
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.9
Moving Average of Mean Queue Length, Queue 4: Pilot Runs 4 to 6



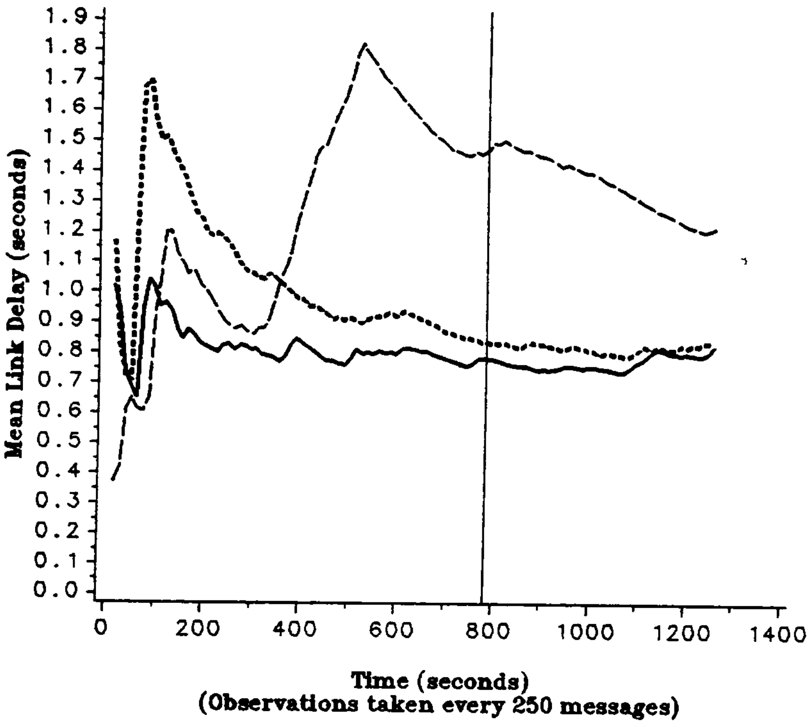
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.10
Moving Average of Mean Link Delay, Queue 1: Pilot Runs 1 to 3



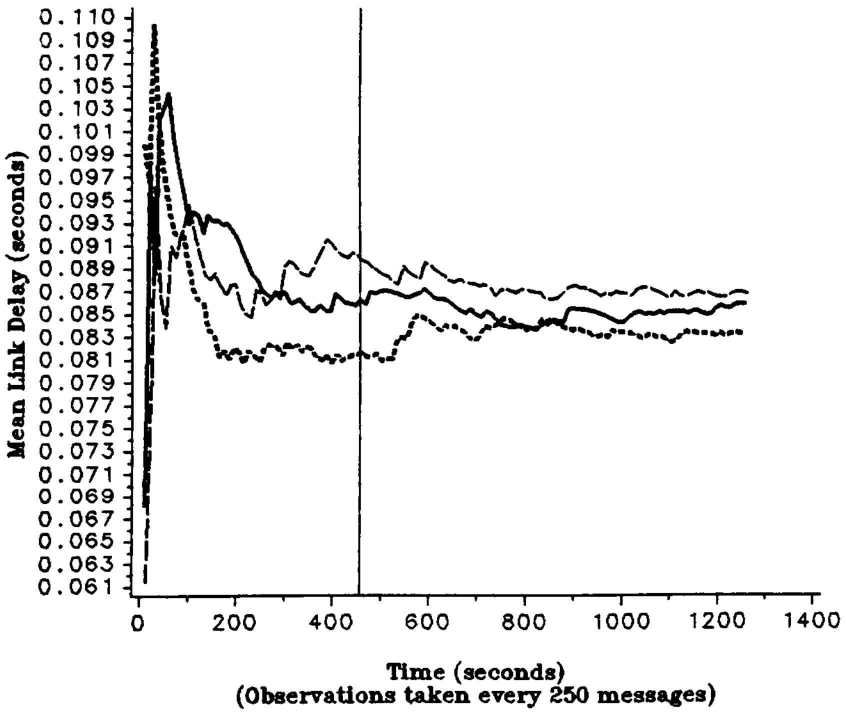
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.11
Moving Average of Mean Link Delay, Queue 1: Pilot Runs 4 to 6



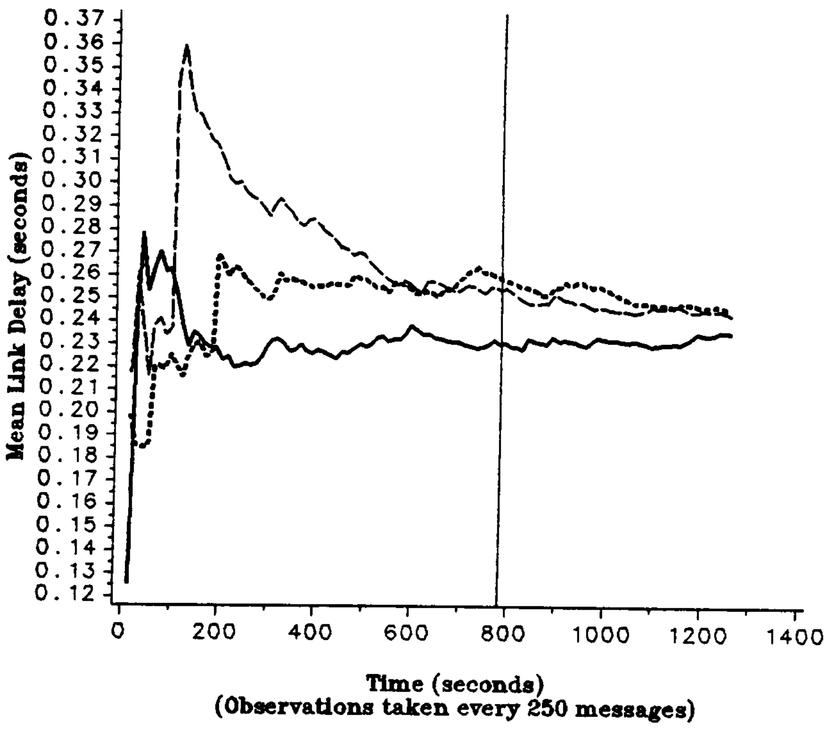
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.12
Moving Average of Mean Link Delay, Queue 2: Pilot Runs 1 to 3



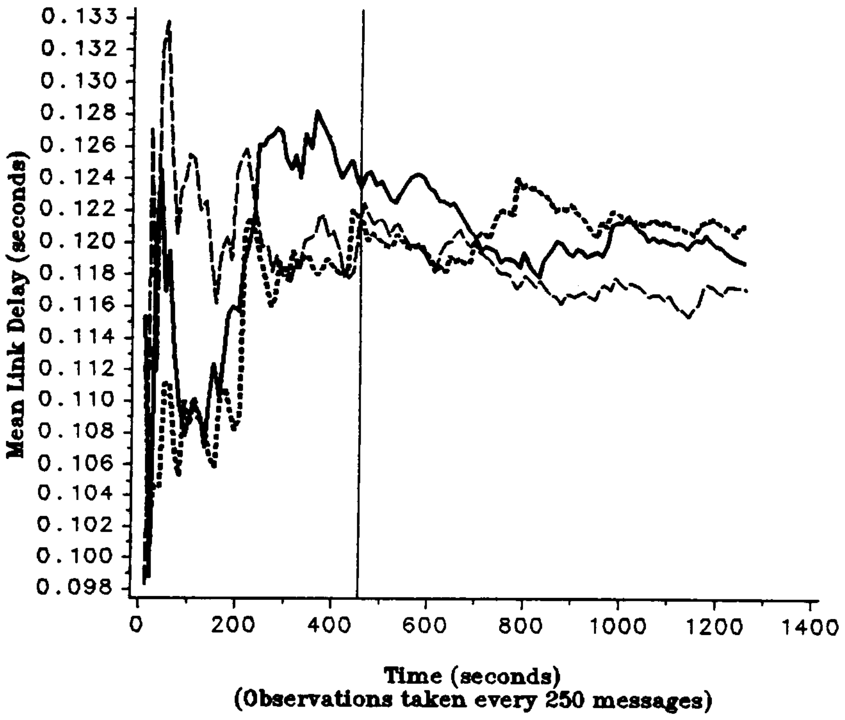
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.13
Moving Average of Mean Link Delay, Queue 2: Pilot Runs 4 to 6



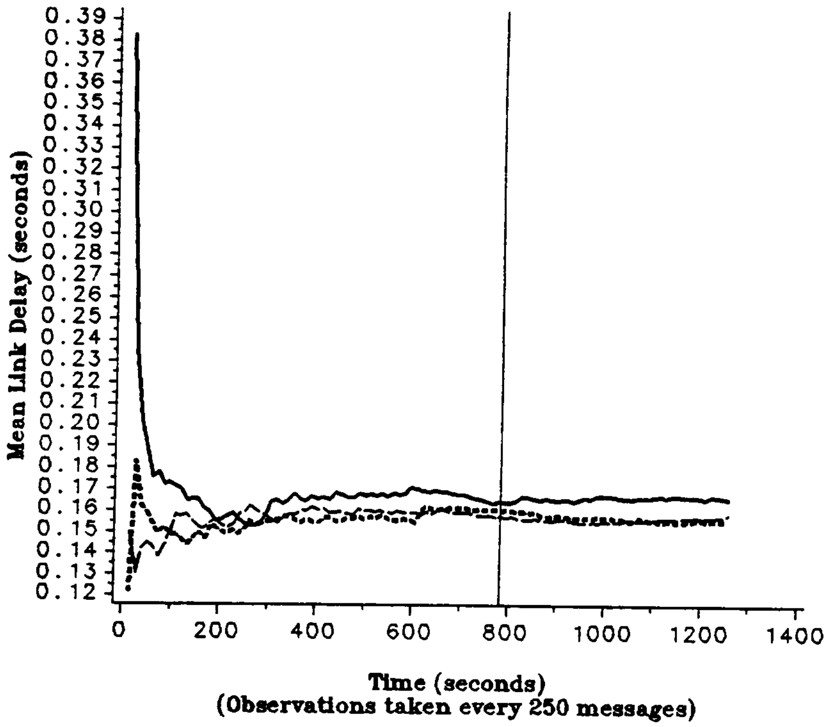
Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.14
Moving Average of Mean Link Delay, Queue 3: Pilot Runs 1 to 3



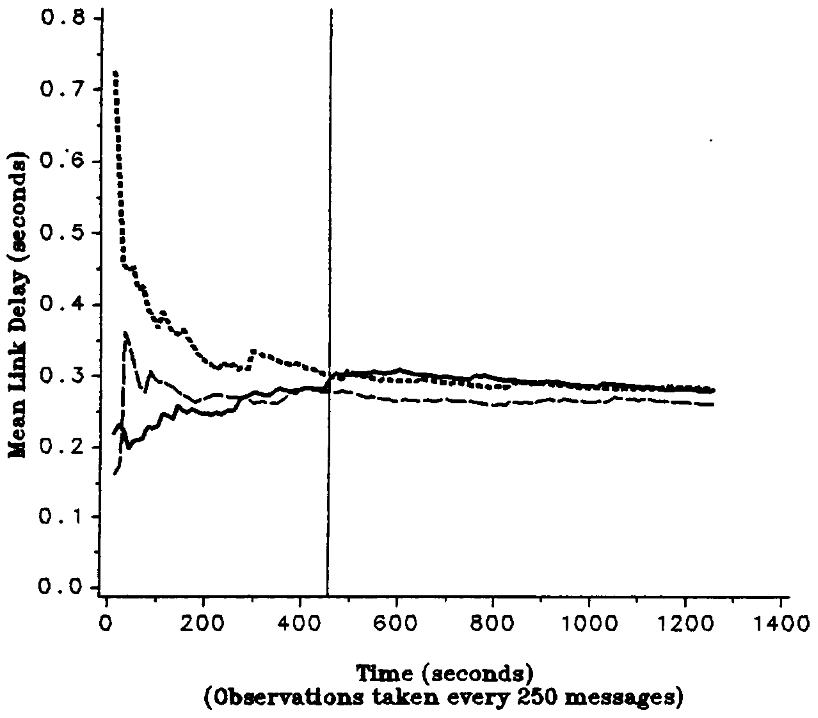
Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.15
Moving Average of Mean Link Delay, Queue 3: Pilot Runs 4 to 6



Legend: Solid = Run 1, Short Dash = Run 2, Long Dash = Run 3

Figure A.16
Moving Average of Mean Link Delay, Queue 4: Pilot Runs 1 to 3



Legend: Solid = Run 4, Short Dash = Run 5, Long Dash = Run 6

Figure A.17
Moving Average of Mean Link Delay, Queue 4: Pilot Runs 4 to 6

APPENDIX B
NETWORK DATA FOR EXECUTION TRACE

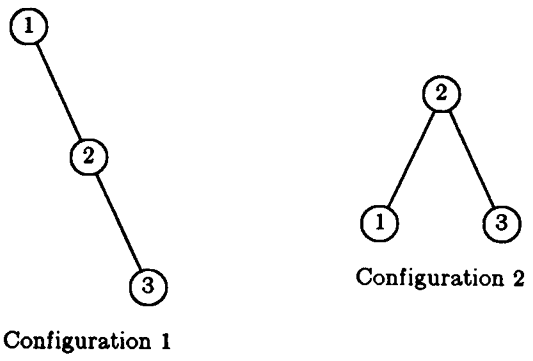


Figure B.1. Network for Execution Trace

Network Parameters

Number of configurations = 2
Number of nodes = 3

Environment process generator:

$$Q_E = \begin{bmatrix} -0.25 & 0.25 \\ 0.25 & -0.25 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.5, 0.5]$$

Table B.1. Link Numbers and Capacities— Execution Trace

link	source node	destination node	capacity
1	2	1	1000.0
2	1	3	1000.0
3	3	1	1000.0
4	1	2	1000.0

Mean message length = 500 bits

Arrival rate matrices: (1,3- and 3,1-traffic only)

$$\Gamma^{(1)} = \Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 1.5 \\ 0.0 & 0.0 & 0.0 \\ 1.5 & 0.0 & 0.0 \end{bmatrix}$$

Sample Trace Output

(Edited to remove unnecessary white space).

```
( 0. ) ENV ACT: INIT STATE: 1
( 0. ) MSG ACT: MSG ID: 1 SRC: 1 DEST: 3
( 0. ) MSG ACT: MSG ID: 2 SRC: 3 DEST: 1
( 0. ) MSG ARRV: NODE: 1 MSG ID: 1
( 0. ) MSG ARRV: LINK: 1 MSG ID: 1
( 0. ) MSG ARRV: NODE: 3 MSG ID: 2
( 0. ) MSG ARRV: LINK: 3 MSG ID: 2
( .16) SERV COMP: LINK: 3 MSG ID: 2 START: 0.
( .16) MSG ARRV: NODE: 2 MSG ID: 2
( .16) MSG ARRV: LINK: 4 MSG ID: 2
( .53) MSG ACT: MSG ID: 3 SRC: 1 DEST: 3
( .53) MSG ARRV: NODE: 1 MSG ID: 3
( .53) MSG ARRV: LINK: 1 MSG ID: 3
( .56) MSG ACT: MSG ID: 4 SRC: 3 DEST: 1
( .56) MSG ARRV: NODE: 3 MSG ID: 4
( .56) MSG ARRV: LINK: 3 MSG ID: 4
( .62) SERV COMP: LINK: 3 MSG ID: 4 START: .56
( .62) MSG ARRV: NODE: 2 MSG ID: 4
( .62) MSG ARRV: LINK: 4 MSG ID: 4
( .62) SERV COMP: LINK: 4 MSG ID: 2 START: .16
( .62) MSG ARRV: NODE: 1 MSG ID: 2
( .62) NET DEP: DELAY: .62131 MSG ID: 2
( .81) MSG ACT: MSG ID: 5 SRC: 1 DEST: 3
( .81) MSG ARRV: NODE: 1 MSG ID: 5
( .81) MSG ARRV: LINK: 1 MSG ID: 5
( .91) SERV COMP: LINK: 1 MSG ID: 1 START: 0.
( .91) MSG ARRV: NODE: 2 MSG ID: 1
( .91) MSG ARRV: LINK: 2 MSG ID: 1
( .98) SERV COMP: LINK: 4 MSG ID: 4 START: .62
( .98) MSG ARRV: NODE: 1 MSG ID: 4
( .98) NET DEP: DELAY: .42278 MSG ID: 4
( 1.00) MSG ACT: MSG ID: 6 SRC: 3 DEST: 1
( 1.00) MSG ARRV: NODE: 3 MSG ID: 6
( 1.00) MSG ARRV: LINK: 3 MSG ID: 6
( 1.06) SERV COMP: LINK: 3 MSG ID: 6 START: 1.00
( 1.06) MSG ARRV: NODE: 2 MSG ID: 6
( 1.06) MSG ARRV: LINK: 4 MSG ID: 6
( 1.17) SERV COMP: LINK: 4 MSG ID: 6 START: 1.06
( 1.17) MSG ARRV: NODE: 1 MSG ID: 6
( 1.17) NET DEP: DELAY: .17217 MSG ID: 6
( 1.25) SERV COMP: LINK: 2 MSG ID: 1 START: .91
( 1.25) MSG ARRV: NODE: 3 MSG ID: 1
( 1.25) NET DEP: DELAY: 1.24846 MSG ID: 1
( 1.52) SERV COMP: LINK: 1 MSG ID: 3 START: .91
( 1.52) MSG ARRV: NODE: 2 MSG ID: 3
( 1.52) MSG ARRV: LINK: 2 MSG ID: 3
( 1.56) MSG ACT: MSG ID: 7 SRC: 3 DEST: 1
( 1.56) MSG ARRV: NODE: 3 MSG ID: 7
( 1.56) MSG ARRV: LINK: 3 MSG ID: 7
( 1.96) SERV COMP: LINK: 1 MSG ID: 5 START: 1.52
( 1.96) MSG ARRV: NODE: 2 MSG ID: 5
( 1.96) MSG ARRV: LINK: 2 MSG ID: 5
( 2.27) MSG ACT: MSG ID: 8 SRC: 3 DEST: 1
( 2.27) MSG ARRV: NODE: 3 MSG ID: 8
( 2.27) MSG ARRV: LINK: 3 MSG ID: 8
( 2.34) SERV COMP: LINK: 2 MSG ID: 3 START: 1.52
( 2.34) MSG ARRV: NODE: 3 MSG ID: 3
( 2.34) NET DEP: DELAY: 1.80793 MSG ID: 3
( 2.57) SERV COMP: LINK: 3 MSG ID: 7 START: 1.56
( 2.57) MSG ARRV: NODE: 2 MSG ID: 7
( 2.57) MSG ARRV: LINK: 4 MSG ID: 7
( 2.68) MSG ACT: MSG ID: 9 SRC: 3 DEST: 1
( 2.68) MSG ARRV: NODE: 3 MSG ID: 9
( 2.68) MSG ARRV: LINK: 3 MSG ID: 9
( 2.74) SERV COMP: LINK: 2 MSG ID: 5 START: 2.34
( 2.74) MSG ARRV: NODE: 3 MSG ID: 5
( 2.74) NET DEP: DELAY: 1.93865 MSG ID: 5
( 2.90) MSG ACT: MSG ID: 10 SRC: 1 DEST: 3
( 2.90) MSG ARRV: NODE: 1 MSG ID: 10
( 2.90) MSG ARRV: LINK: 1 MSG ID: 10
( 2.91) SERV COMP: LINK: 1 MSG ID: 10 START: 2.90
( 2.91) MSG ARRV: NODE: 2 MSG ID: 10
( 2.91) MSG ARRV: LINK: 2 MSG ID: 10
( 3.25) MSG ACT: MSG ID: 11 SRC: 3 DEST: 1
( 3.25) MSG ARRV: NODE: 3 MSG ID: 11
( 3.25) MSG ARRV: LINK: 3 MSG ID: 11
( 3.31) SERV COMP: LINK: 4 MSG ID: 7 START: 2.57
( 3.31) MSG ARRV: NODE: 1 MSG ID: 7
( 3.31) NET DEP: DELAY: 1.74920 MSG ID: 7
( 3.37) SERV COMP: LINK: 3 MSG ID: 8 START: 2.57
( 3.37) MSG ARRV: NODE: 2 MSG ID: 8
( 3.37) MSG ARRV: LINK: 4 MSG ID: 8
( 3.66) MSG ACT: MSG ID: 12 SRC: 3 DEST: 1
( 3.66) MSG ARRV: NODE: 3 MSG ID: 12
( 3.66) MSG ARRV: LINK: 3 MSG ID: 12
( 3.82) SERV COMP: LINK: 4 MSG ID: 8 START: 3.37
( 3.82) MSG ARRV: NODE: 1 MSG ID: 8
( 3.82) NET DEP: DELAY: 1.54949 MSG ID: 8
( 3.88) MSG ACT: MSG ID: 13 SRC: 1 DEST: 3
( 3.88) MSG ARRV: NODE: 1 MSG ID: 13
( 3.88) MSG ARRV: LINK: 1 MSG ID: 13
( 3.95) SERV COMP: LINK: 1 MSG ID: 13 START: 3.88
( 3.95) MSG ARRV: NODE: 2 MSG ID: 13
( 3.95) MSG ARRV: LINK: 2 MSG ID: 13
( 4.02) MSG ACT: MSG ID: 14 SRC: 1 DEST: 3
( 4.02) MSG ARRV: NODE: 1 MSG ID: 14
( 4.02) MSG ARRV: LINK: 1 MSG ID: 14
( 4.36) SERV COMP: LINK: 2 MSG ID: 10 START: 2.91
( 4.36) MSG ARRV: NODE: 3 MSG ID: 10
( 4.36) NET DEP: DELAY: 1.45912 MSG ID: 10
( 4.40) SERV COMP: LINK: 1 MSG ID: 14 START: 4.02
( 4.40) MSG ARRV: NODE: 2 MSG ID: 14
( 4.40) MSG ARRV: LINK: 2 MSG ID: 14
( 4.59) SERV COMP: LINK: 3 MSG ID: 9 START: 3.37
( 4.59) MSG ARRV: NODE: 2 MSG ID: 9
( 4.59) MSG ARRV: LINK: 4 MSG ID: 9
( 4.62) SERV COMP: LINK: 2 MSG ID: 13 START: 4.36
( 4.62) MSG ARRV: NODE: 3 MSG ID: 13
( 4.62) NET DEP: DELAY: .73519 MSG ID: 13
( 4.76) MSG ACT: MSG ID: 15 SRC: 1 DEST: 3
( 4.76) MSG ARRV: NODE: 1 MSG ID: 15
( 4.76) MSG ARRV: LINK: 1 MSG ID: 15
( 4.79) SERV COMP: LINK: 2 MSG ID: 14 START: 4.62
( 4.79) MSG ARRV: NODE: 3 MSG ID: 14
( 4.79) NET DEP: DELAY: .77213 MSG ID: 14
( 5.01) SERV COMP: LINK: 4 MSG ID: 9 START: 4.59
( 5.01) MSG ARRV: NODE: 1 MSG ID: 9
( 5.01) NET DEP: DELAY: 2.32609 MSG ID: 9
( 5.40) MSG ACT: MSG ID: 16 SRC: 3 DEST: 1
( 5.40) MSG ARRV: NODE: 3 MSG ID: 16
```

APPENDIX C
NETWORK DATA FOR STRESS TESTING

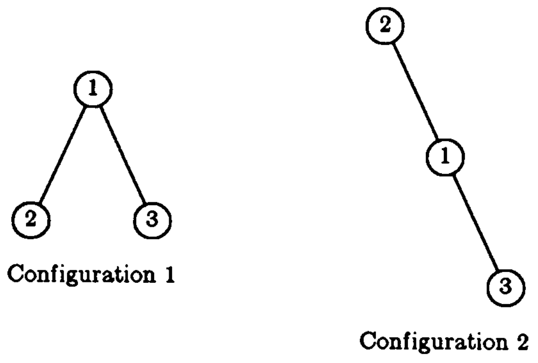


Figure C.1. Network for Stress Testing

Number of configurations = 2.
Number of nodes = 3.

General traffic, variable $\rho_j^{(i)}$ for all links j.

(Number of messages in transient period = 30000. Batch size = 30000 messages.
Number of Batches = 25).

Environment process generator:

$$Q_E = \begin{bmatrix} -2.0 & 2.0 \\ 2.0 & -2.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5, 0.5]$$

Table C.1. Link Numbers and Capacities— Stress Test

link	source node	destination node	capacity
1	1	2	1000.0
2	2	3	2000.0
3	3	2	2000.0
4	2	1	1000.0

Mean message length: $\mu^{-1} = 100.0$ bits

Parameter Set 1 ($\rho_4 = 0.9$) Traffic Matrices

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 4.1538 & 11.0768 \\ 5.5384 & 0.0 & 2.7692 \\ 12.4614 & 6.923 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 2.7692 & 6.923 \\ 2.7692 & 0.0 & 1.3846 \\ 6.923 & 4.1538 & 0.0 \end{bmatrix}$$

Parameter Set 2 ($\rho_4 = 0.95$) Traffic Matrices

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 4.3845 & 11.6920 \\ 5.8460 & 0.0 & 2.923 \\ 13.1535 & 7.3075 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 2.923 & 7.3075 \\ 2.923 & 0.0 & 1.4615 \\ 7.3075 & 4.3845 & 0.0 \end{bmatrix}$$

Parameter Set 3 ($\rho_4 = 0.99$) Traffic Matrices

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 4.6593 & 12.1848 \\ 6.0924 & 0.0 & 3.0462 \\ 13.7079 & 7.6155 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 3.0462 & 7.6155 \\ 3.0462 & 0.0 & 1.5231 \\ 7.6155 & 4.5693 & 0.0 \end{bmatrix}$$

Parameter Set 4 ($\rho_4 = 0.995$) Traffic Matrices

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 4.5924 & 12.2464 \\ 6.1232 & 0.0 & 3.0616 \\ 13.7772 & 7.654 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 3.0616 & 7.654 \\ 3.0616 & 0.0 & 1.5308 \\ 7.654 & 4.5924 & 0.0 \end{bmatrix}$$

Parameter Set 5 ($\rho_4 = 0.999$) Traffic Matrices

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 4.6107 & 12.2952 \\ 6.1476 & 0.0 & 3.0738 \\ 13.8321 & 7.6845 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 3.0738 & 7.6845 \\ 3.0738 & 0.0 & 1.5369 \\ 7.6845 & 4.6107 & 0.0 \end{bmatrix}$$

APPENDIX D
NETWORK TOPOLOGIES AND PARAMETER SETS FOR ANALYTIC AND SIMULATION MODEL COMPARISON

D.1. Network 1

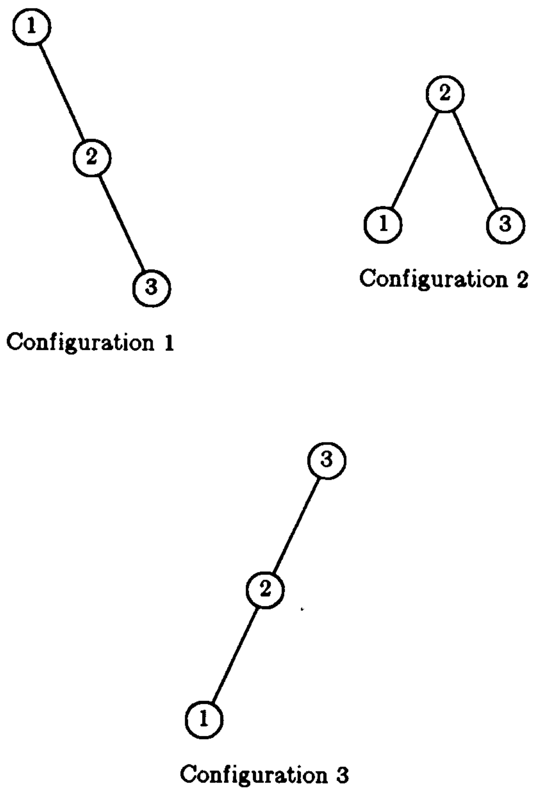


Figure D.1. Dynamic Hierarchy 1 Topology

Network 1, Parameter Set 1

Number of nodes: 3
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -16.0 & 9.0 & 7.0 \\ 9.0 & -10.0 & 1.0 \\ 10.0 & 1.0 & -11.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.3707, 0.3605, 0.2687]$$

Table D.1. Link Numbers and Capacities— Network 1, Parameter Set 1

link	source node	destination node	capacity
1	2	1	1302.39001
2	1	2	2793.03760
3	3	2	1302.39001
4	2	3	2793.03760

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 16.0 \\ 0.0 & 0.0 & 0.0 \\ 11.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 15.0 \\ 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 4.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 1, Parameter set 2

Number of nodes: 3
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -16.0 & 9.0 & 7.0 \\ 9.0 & -10.0 & 1.0 \\ 10.0 & 1.0 & -11.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.3707, 0.3605, 0.2687]$$

Table D.2. Link Numbers and Capacities— Network 1, Parameter Set 2

link	source node	destination node	capacity
1	2	1	1302.39001
2	1	2	2793.03760
3	3	2	1302.39001
4	2	3	2793.03760

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 8.0 \\ 0.0 & 0.0 & 0.0 \\ 5.50 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 7.50 \\ 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 2.0 \\ 0.0 & 0.0 & 0.0 \\ 0.50 & 0.0 & 0.0 \end{bmatrix}$$

Network 1, Parameter Set 3

Number of nodes: 3
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -16.0 & 9.0 & 7.0 \\ 9.0 & -10.0 & 1.0 \\ 10.0 & 1.0 & -11.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.3707,\ 0.3605,\ 0.2687]$$

Table D.3. Link Numbers and Capacities— Network 1, Parameter Set 3

link	source node	destination node	capacity
1	2	1	1302.39001
2	1	2	2793.03760
3	3	2	1302.39001
4	2	3	2793.03760

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 32.0 \\ 0.0 & 0.0 & 0.0 \\ 22.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 30.0 \\ 0.0 & 0.0 & 0.0 \\ 8.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 8.0 \\ 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 \end{bmatrix}$$

D.2. Network 2

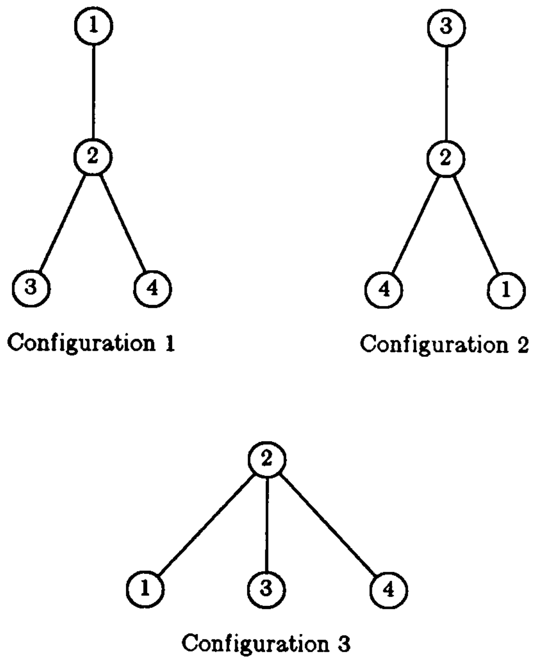


Figure D.2. Dynamic Hierarchy 2 Topology

Network 2, Parameter Set 1

Number of nodes: 4
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -15.0 & 9.0 & 6.0 \\ 1.0 & -4.0 & 3.0 \\ 4.0 & 8.0 & -12.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.1039,\ 0.6753,\ 0.2208]$$

Table D.4. Link Numbers and Capacities— Network 2, Parameter Set 1			
link	source node	destination node	capacity
1	2	1	5996.02490
2	1	2	3208.47754
3	3	2	3097.35010
4	2	3	1814.60254
5	4	2	2898.67505
6	2	4	1393.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 15.0 & 6.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 8.0 & 4.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 17.0 & 0.0 & 0.0 & 0.0 \\ 15.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 5.0 & 13.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 8.0 & 0.0 & 0.0 & 0.0 \\ 12.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 2, Parameter Set 2

Number of nodes: 4
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -15.0 & 9.0 & 6.0 \\ 1.0 & -4.0 & 3.0 \\ 4.0 & 8.0 & -12.0 \end{bmatrix}$$

Environment stationary probabilities:

$\boldsymbol{\pi} = [0.1039, 0.6753, 0.2208]$

Table D.5. Link Numbers and Capacities— Network 2, Parameter Set 2

link	source node	destination node	capacity
1	2	1	5996.02490
2	1	2	3208.47754
3	3	2	3097.35010
4	2	3	1814.60254
5	4	2	2898.67505
6	2	4	1393.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 7.50 & 3.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 2.50 & 0.0 & 0.0 & 0.0 \\ 0.50 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 4.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 8.50 & 0.0 & 0.0 & 0.0 \\ 7.50 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 2.50 & 6.50 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 2, Parameter Set 3

Number of nodes: 4
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -15.0 & 9.0 & 6.0 \\ 1.0 & -4.0 & 3.0 \\ 4.0 & 8.0 & -12.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.1039, 0.6753, 0.2208]$$

Table D.6. Link Numbers and Capacities— Network 2, Parameter Set 3

link	source node	destination node	capacity
1	2	1	5996.02490
2	1	2	3208.47754
3	3	2	3097.35010
4	2	3	1814.60254
5	4	2	2898.67505
6	2	4	1393.8750

Mean message length: $\mu^{-1} = 100.0$ bits

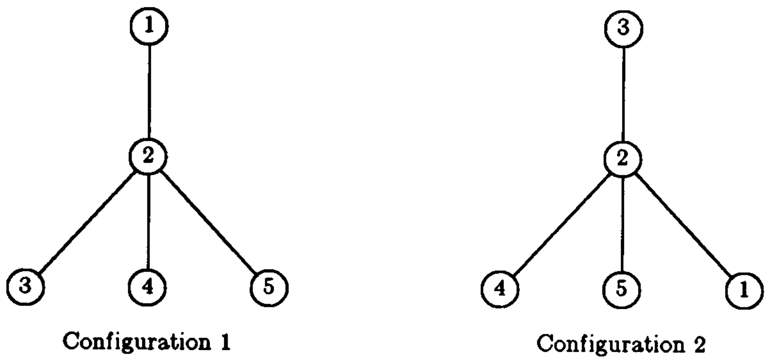
Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 30.0 & 12.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 10.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 16.0 & 8.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 34.0 & 0.0 & 0.0 & 0.0 \\ 30.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 10.0 & 26.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 16.0 & 0.0 & 0.0 & 0.0 \\ 24.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

D.3. Network 3



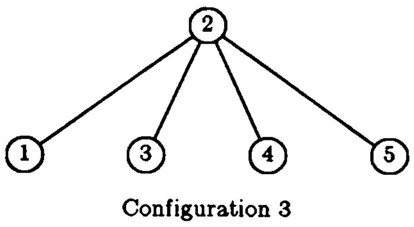


Figure D.3. Dynamic Hierarchy 3 Topology

Network 3, Parameter Set 1

Number of nodes: 5
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -3.0 & 2.0 & 1.0 \\ 3.0 & -12.0 & 9.0 \\ 9.0 & 9.0 & -18.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.6429, 0.2143, 0.1429]$$

Table D.7. Link Numbers and Capacities— Network 3, Parameter Set 1			
link	source node	destination node	capacity
1	2	1	7490.04736
2	1	2	5127.29980
3	3	2	2700.24756
4	2	3	1671.56995
5	4	2	2330.59497
6	2	4	884.09253
7	5	2	2496.20508
8	2	5	2571.63745

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 9.0 & 1.0 & 11.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 11.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 11.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 5.0 & 6.0 & 17.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 17.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 4.0 & 14.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 11.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 15.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 3, Parameter Set 2

Number of nodes: 5
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -3.0 & 2.0 & 1.0 \\ 3.0 & -12.0 & 9.0 \\ 9.0 & 9.0 & -18.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.6429,\ 0.2143,\ 0.1429]$$

Table D.8. Link Numbers and Capacities— Network 3, Parameter Set 2

link	source node	destination node	capacity
1	2	1	7490.04736
2	1	2	5127.29980
3	3	2	2700.24756
4	2	3	1671.56995
5	4	2	2330.59497
6	2	4	884.09253
7	5	2	2496.20508
8	2	5	2571.63745

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 4.50 & 0.50 & 5.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.50 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.50 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.50 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 2.50 & 3.0 & 8.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.50 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 2.0 & 7.0 & 2.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.50 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.50 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.50 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 3, Parameter Set 3

Number of nodes: 5
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -3.0 & 2.0 & 1.0 \\ 3.0 & -12.0 & 9.0 \\ 9.0 & 9.0 & -18.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.6429, 0.2143, 0.1429]$$

Table D.9. Link Numbers and Capacities— Network 3, Parameter Set 3

link	source node	destination node	capacity
1	2	1	7490.04736
2	1	2	5127.29980
3	3	2	2700.24756
4	2	3	1671.56995
5	4	2	2330.59497
6	2	4	884.09253
7	5	2	2496.20508
8	2	5	2571.63745

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 18.0 & 2.0 & 22.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 22.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 22.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 10.0 & 12.0 & 34.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 34.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 16.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 28.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 8.0 & 28.0 & 10.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 22.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 30.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

D.4. Network 4

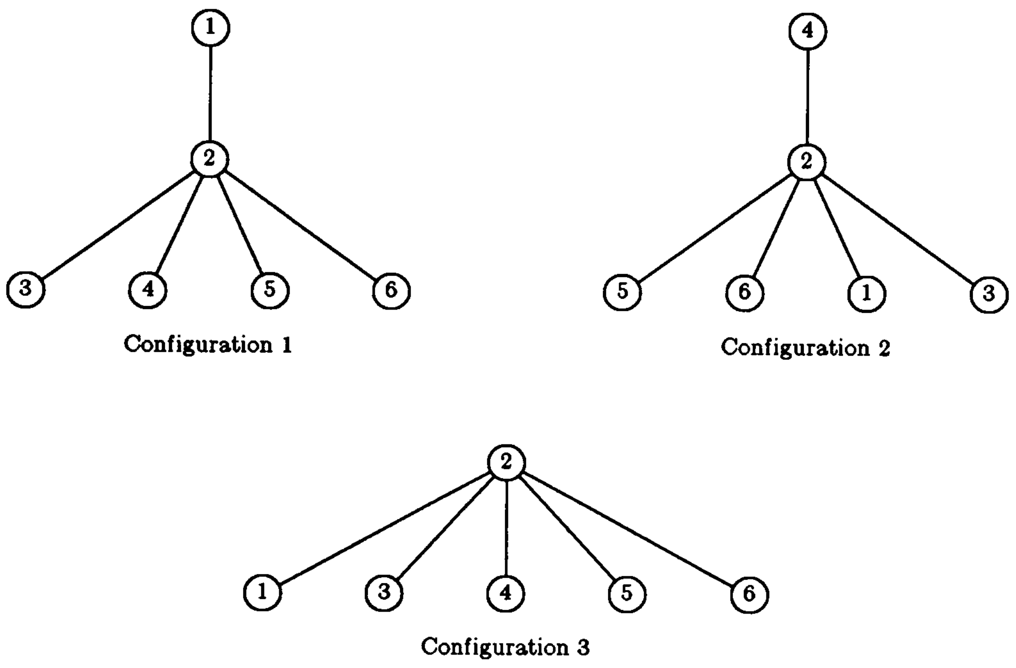


Figure D.4. Dynamic Hierarchy 4 Topology

Network 4, Parameter Set 1

Number of nodes: 6
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -5.0 & 4.0 & 1.0 \\ 5.0 & -12.0 & 7.0 \\ 3.0 & 4.0 & -7.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.4375, 0.250, 0.3125]$$

Table D.10. Link Numbers and Capacities— Network 4, Parameter Set 1

link	source node	destination node	capacity
1	2	1	9871.8750
2	1	2	13359.3750
3	3	2	2742.18750
4	2	3	2868.750
5	4	2	1673.43750
6	2	4	3037.50
7	5	2	3037.50
8	2	5	3656.250
9	6	2	2418.750
10	2	6	3796.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 15.0 & 17.0 & 18.0 & 19.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 11.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 12.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 16.0 & 8.0 & 11.0 & 18.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 12.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 17.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 7.0 & 13.0 & 18.0 & 13.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 4, Parameter Set 2

Number of nodes: 6
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -5.0 & 4.0 & 1.0 \\ 5.0 & -12.0 & 7.0 \\ 3.0 & 4.0 & -7.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.4375, 0.250, 0.3125]$$

Table D.11. Link Numbers and Capacities— Network 4, Parameter Set 2

link	source node	destination node	capacity
1	2	1	9871.8750
2	1	2	13359.3750
3	3	2	2742.18750
4	2	3	2868.750
5	4	2	1673.43750
6	2	4	3037.50
7	5	2	3037.50
8	2	5	3656.250
9	6	2	2418.750
10	2	6	3796.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 7.50 & 8.50 & 9.0 & 9.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 8.0 & 4.0 & 5.50 & 9.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 3.50 & 6.50 & 9.0 & 6.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 4, Parameter Set 3

Number of nodes: 6
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -5.0 & 4.0 & 1.0 \\ 5.0 & -12.0 & 7.0 \\ 3.0 & 4.0 & -7.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.4375, 0.250, 0.3125]$$

Table D.12. Link Numbers and Capacities— Network 4, Parameter Set 3

link	source node	destination node	capacity
1	2	1	9871.8750
2	1	2	13359.3750
3	3	2	2742.18750
4	2	3	2868.750
5	4	2	1673.43750
6	2	4	3037.50
7	5	2	3037.50
8	2	5	3656.250
9	6	2	2418.750
10	2	6	3796.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 30.0 & 34.0 & 36.0 & 38.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 22.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 12.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 28.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 24.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 32.0 & 16.0 & 22.0 & 36.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 24.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 36.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 34.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 14.0 & 26.0 & 36.0 & 26.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 28.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 36.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

D.5. Network 5

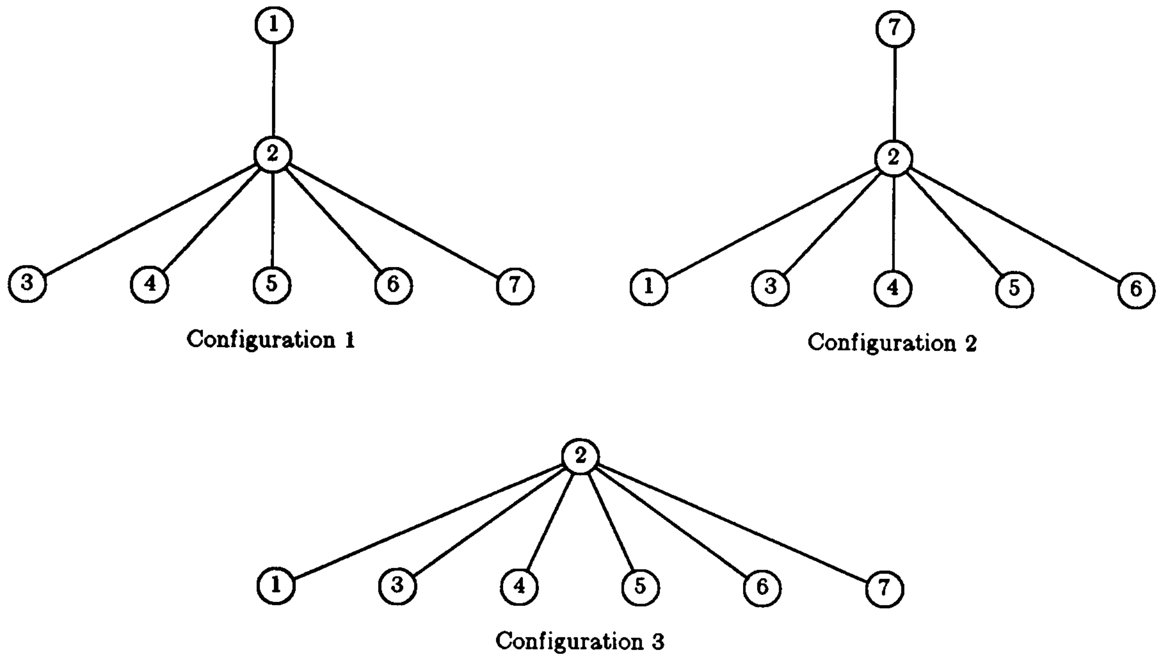


Figure D.5. Dynamic Hierarchy 5 Topology

Network 5, Parameter Set 1

Number of nodes: 7
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -14.0 & 8.0 & 6.0 \\ 3.0 & -5.0 & 2.0 \\ 9.0 & 2.0 & -11.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.2589, 0.5076, 0.2335]$$

Table D.13. Link Numbers and Capacities— Network 5, Parameter Set 1

link	source node	destination node	capacity
1	2	1	10080.42676
2	1	2	12738.19531
3	3	2	3225.39746
4	2	3	3067.71753
5	4	2	1075.90503
6	2	4	2305.91260
7	5	2	2540.04761
8	2	5	2284.31250
9	6	2	2274.00757
10	2	6	1986.20996
11	7	2	965.09253
12	2	7	3094.04248

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 4.0 & 1.0 & 18.0 & 18.0 & 15.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 13.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 17.0 & 16.0 & 3.0 & 1.0 & 13.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 17.0 & 8.0 & 17.0 & 19.0 & 14.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 11.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 5, Parameter Set 2

Number of nodes: 7
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -14.0 & 8.0 & 6.0 \\ 3.0 & -5.0 & 2.0 \\ 9.0 & 2.0 & -11.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.2589, 0.5076, 0.2335]$$

Table D.14. Link Numbers and Capacities— Network 5, Parameter Set 2

link	source node	destination node	capacity
1	2	1	10080.42676
2	1	2	12738.19531
3	3	2	3225.39746
4	2	3	3067.71753
5	4	2	1075.90503
6	2	4	2305.91260
7	5	2	2540.04761
8	2	5	2284.31250
9	6	2	2274.00757
10	2	6	1986.20996
11	7	2	965.09253
12	2	7	3094.04248

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 2.0 & 0.50 & 9.0 & 7.50 & 7.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 8.50 & 8.0 & 1.50 & 0.50 & 6.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 8.50 & 4.0 & 8.50 & 9.50 & 7.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 5.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 5, Parameter Set 3

Number of nodes: 7
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -14.0 & 8.0 & 6.0 \\ 3.0 & -5.0 & 2.0 \\ 9.0 & 2.0 & -11.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.2589, 0.5076, 0.2335]$$

Table D.15. Link Numbers and Capacities— Network 5, Parameter Set 3			
link	source node	destination node	capacity
1	2	1	10080.42676
2	1	2	12738.19531
3	3	2	3225.39746
4	2	3	3067.71753
5	4	2	1075.90503
6	2	4	2305.91260
7	5	2	2540.04761
8	2	5	2284.31250
9	6	2	2274.00757
10	2	6	1986.20996
11	7	2	965.09253
12	2	7	3094.04248

Mean message length: $\mu^{-1} = 100.0$ bits

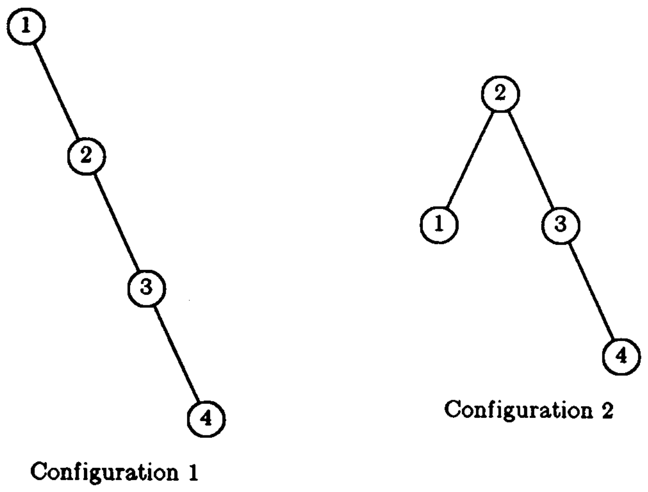
Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 8.0 & 2.0 & 36.0 & 30.0 & 30.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 36.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 26.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 34.0 & 32.0 & 6.0 & 2.0 & 26.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 28.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 12.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 36.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 10.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 12.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 34.0 & 16.0 & 34.0 & 38.0 & 28.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 22.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 36.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

D.6. Network 6



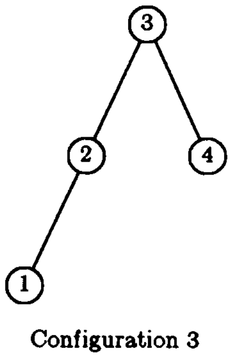


Figure D.6. Dynamic Hierarchy 6 Topology

Network 6, Parameter Set 1

Number of nodes: 4
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -7.0 & 2.0 & 5.0 \\ 6.0 & -16.0 & 10.0 \\ 7.0 & 9.0 & -16.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.4840,\ 0.2245,\ 0.2915]$$

Table D.16. Link Numbers and Capacities— Network 6, Parameter Set 1

link	source node	destination node	capacity
1	2	1	2179.91260
2	1	2	3600.67505
3	3	2	2179.91260
4	2	3	3600.67505
5	4	3	2179.91260
6	3	4	3600.67505

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 18.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 10.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 13.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 19.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 15.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 6, Parameter Set 2

Number of nodes: 4
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -7.0 & 2.0 & 5.0 \\ 6.0 & -16.0 & 10.0 \\ 7.0 & 9.0 & -16.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.4840, 0.2245, 0.2915]$$

Table D.17. Link Numbers and Capacities— Network 6, Parameter Set 2

link	source node	destination node	capacity
1	2	1	2179.91260
2	1	2	3600.67505
3	3	2	2179.91260
4	2	3	3600.67505
5	4	3	2179.91260
6	3	4	3600.67505

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 9.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 5.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 6.50 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 9.50 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 7.50 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 6, Parameter Set 3

Number of nodes: 4
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -7.0 & 2.0 & 5.0 \\ 6.0 & -16.0 & 10.0 \\ 7.0 & 9.0 & -16.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.4840, 0.2245, 0.2915]$$

Table D.18. Link Numbers and Capacities— Network 6, Parameter Set 3

link	source node	destination node	capacity
1	2	1	2179.91260
2	1	2	3600.67505
3	3	2	2179.91260
4	2	3	3600.67505
5	4	3	2179.91260
6	3	4	3600.67505

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 36.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 20.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 26.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 38.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 30.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 4.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

D.7. Network 7

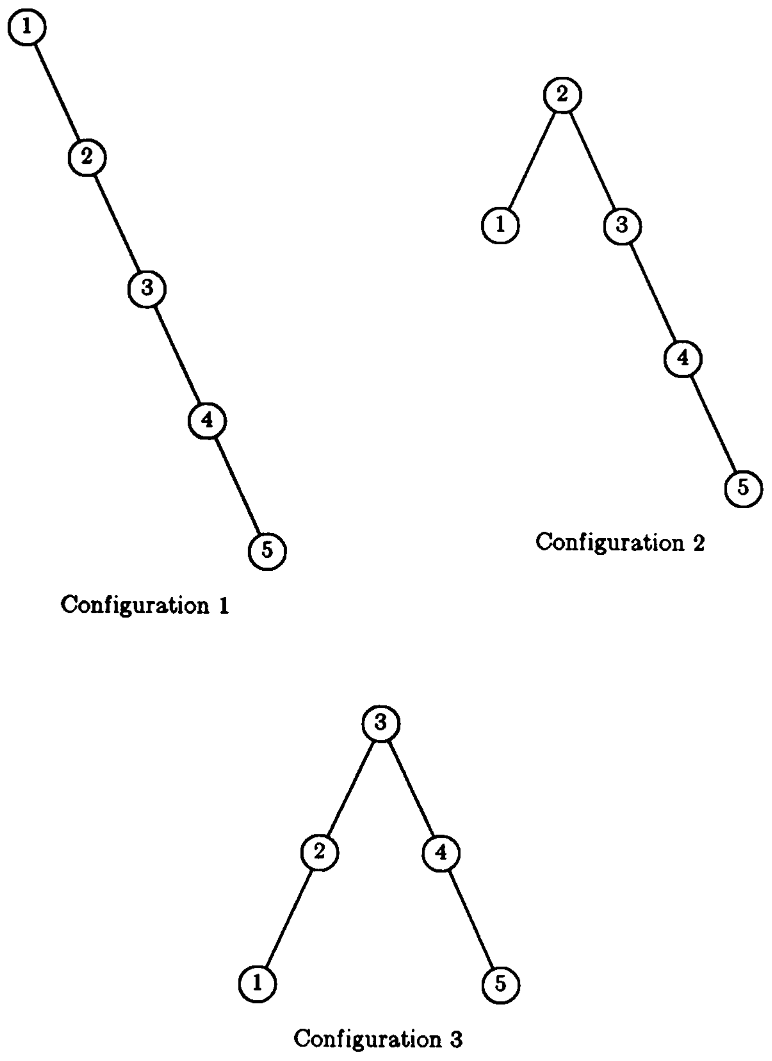


Figure D.7. Dynamic Hierarchy 7 Topology

Network 7, Parameter Set 1

Number of nodes: 5
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -12.0 & 2.0 & 10.0 \\ 5.0 & -8.0 & 3.0 \\ 1.0 & 4.0 & -5.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.1701, 0.3409, 0.5244]$$

Table D.19. Link Numbers and Capacities— Network 7, Parameter Set 1

link	source node	destination node	capacity
1	2	1	3466.03491
2	1	2	966.64502
3	3	2	3466.03491
4	2	3	966.64502
5	4	3	3466.03491
6	3	4	966.64502
7	5	4	3466.03491
8	4	5	966.64502

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 12.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 17.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 3.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 18.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 7, Parameter Set 2

Number of nodes: 5
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -12.0 & 2.0 & 10.0 \\ 5.0 & -8.0 & 3.0 \\ 1.0 & 4.0 & -5.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.1701, 0.3409, 0.5244]$$

Table D.20. Link Numbers and Capacities— Network 7, Parameter Set 2

link	source node	destination node	capacity
1	2	1	3466.03491
2	1	2	966.64502
3	3	2	3466.03491
4	2	3	966.64502
5	4	3	3466.03491
6	3	4	966.64502
7	5	4	3466.03491
8	4	5	966.64502

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 6.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.50 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.50 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 1.50 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Network 7, Parameter Set 3

Number of nodes: 5
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -12.0 & 2.0 & 10.0 \\ 5.0 & -8.0 & 3.0 \\ 1.0 & 4.0 & -5.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.1701, 0.3409, 0.5244]$$

Table D.21. Link Numbers and Capacities— Network 7, Parameter Set 3

link	source node	destination node	capacity
1	2	1	3466.03491
2	1	2	966.64502
3	3	2	3466.03491
4	2	3	966.64502
5	4	3	3466.03491
6	3	4	966.64502
7	5	4	3466.03491
8	4	5	966.64502

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 24.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 4.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 34.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(3)} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 6.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 36.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

D.8. Network 8

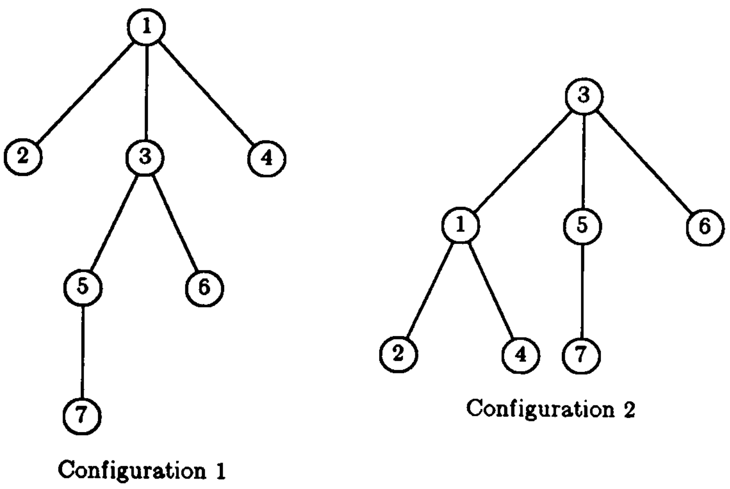


Figure D.8. Dynamic Hierarchy 8 Topology

Network 8, Parameter Set 1

Number of nodes: 7
Number of configurations: 2

Environment process generator:

$$Q_E = \begin{bmatrix} -4.0 & 4.0 \\ 6.0 & -6.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.60, 0.40]$$

Table D.22. Link Numbers and Capacities— Network 8, Parameter Set 1			
link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 15.0 & 18.0 & 14.0 & 10.0 & 11.0 & 7.0 \\ 12.0 & 0.0 & 10.0 & 8.0 & 6.0 & 6.0 & 5.0 \\ 16.0 & 11.0 & 0.0 & 13.0 & 8.0 & 5.0 & 6.0 \\ 17.0 & 11.0 & 10.0 & 0.0 & 3.0 & 7.0 & 4.0 \\ 12.0 & 8.0 & 9.0 & 8.0 & 0.0 & 4.0 & 9.0 \\ 9.0 & 7.0 & 9.0 & 7.0 & 3.0 & 0.0 & 2.0 \\ 10.0 & 4.0 & 10.0 & 5.0 & 8.0 & 4.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 10.0 & 14.0 & 9.0 & 6.0 & 7.0 & 4.0 \\ 8.0 & 0.0 & 11.0 & 5.0 & 5.0 & 6.0 & 7.0 \\ 20.0 & 8.0 & 0.0 & 7.0 & 19.0 & 16.0 & 10.0 \\ 12.0 & 10.0 & 13.0 & 0.0 & 9.0 & 3.0 & 6.0 \\ 8.0 & 7.0 & 12.0 & 6.0 & 0.0 & 7.0 & 9.0 \\ 9.0 & 4.0 & 17.0 & 2.0 & 8.0 & 0.0 & 4.0 \\ 6.0 & 3.0 & 9.0 & 4.0 & 12.0 & 8.0 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 2

Number of nodes: 7
Number of configurations: 2

Environment process generator:

$$Q_E = \begin{bmatrix} -4.0 & 4.0 \\ 6.0 & -6.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.60, 0.40]$$

Table D.23. Link Numbers and Capacities— Network 8, Parameter Set 2

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 7.50 & 9.0 & 7.0 & 5.0 & 5.50 & 3.50 \\ 6.0 & 0.0 & 5.0 & 4.0 & 3.0 & 3.0 & 2.50 \\ 8.0 & 5.50 & 0.0 & 6.50 & 4.0 & 2.50 & 3.0 \\ 8.50 & 5.50 & 5.0 & 0.0 & 1.50 & 3.50 & 2.0 \\ 6.0 & 4.0 & 4.50 & 4.0 & 0.0 & 2.0 & 4.50 \\ 4.50 & 3.50 & 4.50 & 3.50 & 1.50 & 0.0 & 1.0 \\ 5.0 & 2.0 & 5.0 & 2.50 & 4.0 & 2.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 5.0 & 7.0 & 4.50 & 3.0 & 3.50 & 2.0 \\ 4.0 & 0.0 & 5.50 & 2.50 & 2.50 & 3.0 & 3.50 \\ 10.0 & 4.0 & 0.0 & 3.50 & 9.50 & 8.0 & 5.0 \\ 6.0 & 5.0 & 6.50 & 0.0 & 4.50 & 1.50 & 3.0 \\ 4.0 & 3.50 & 6.0 & 3.0 & 0.0 & 3.50 & 4.50 \\ 4.50 & 2.0 & 8.50 & 1.0 & 4.0 & 0.0 & 2.0 \\ 3.0 & 1.50 & 4.50 & 2.0 & 6.0 & 4.0 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 3

Number of nodes: 7
Number of configurations: 2

Environment process generator:

$$Q_E = \begin{bmatrix} -4.0 & 4.0 \\ 6.0 & -6.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.60,\ 0.40]$$

Table D.24. Link Numbers and Capacities— Network 8, Parameter Set 3

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 30.0 & 36.0 & 28.0 & 20.0 & 22.0 & 14.0 \\ 24.0 & 0.0 & 20.0 & 16.0 & 12.0 & 12.0 & 10.0 \\ 32.0 & 22.0 & 0.0 & 26.0 & 16.0 & 10.0 & 12.0 \\ 34.0 & 22.0 & 20.0 & 0.0 & 6.0 & 14.0 & 8.0 \\ 24.0 & 16.0 & 18.0 & 16.0 & 0.0 & 8.0 & 18.0 \\ 18.0 & 14.0 & 18.0 & 14.0 & 6.0 & 0.0 & 4.0 \\ 20.0 & 8.0 & 20.0 & 10.0 & 16.0 & 8.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 20.0 & 28.0 & 18.0 & 12.0 & 14.0 & 8.0 \\ 16.0 & 0.0 & 22.0 & 10.0 & 10.0 & 12.0 & 14.0 \\ 40.0 & 16.0 & 0.0 & 14.0 & 38.0 & 32.0 & 20.0 \\ 24.0 & 20.0 & 26.0 & 0.0 & 18.0 & 6.0 & 12.0 \\ 16.0 & 14.0 & 24.0 & 12.0 & 0.0 & 14.0 & 18.0 \\ 18.0 & 8.0 & 34.0 & 4.0 & 16.0 & 0.0 & 8.0 \\ 12.0 & 6.0 & 18.0 & 8.0 & 24.0 & 16.0 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 4

Number of nodes: 7
Number of configurations: 2

Environment process generator:

$$Q_E = \begin{bmatrix} -4.0 & 4.0 \\ 6.0 & -6.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.60, 0.40]$$

Table D.25. Link Numbers and Capacities— Network 8, Parameter Set 4

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 15.0 & 18.0 & 14.0 & 10.0 & 11.0 & 7.0 \\ 12.0 & 0.0 & 10.0 & 8.0 & 6.0 & 6.0 & 5.0 \\ 16.0 & 11.0 & 0.0 & 13.0 & 8.0 & 5.0 & 6.0 \\ 17.0 & 11.0 & 10.0 & 0.0 & 3.0 & 7.0 & 4.0 \\ 12.0 & 8.0 & 9.0 & 8.0 & 0.0 & 4.0 & 9.0 \\ 9.0 & 7.0 & 9.0 & 7.0 & 3.0 & 0.0 & 2.0 \\ 10.0 & 4.0 & 10.0 & 5.0 & 8.0 & 4.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 5.0 & 7.0 & 4.50 & 3.0 & 3.50 & 2.0 \\ 4.0 & 0.0 & 5.50 & 2.50 & 2.50 & 3.0 & 3.50 \\ 10.0 & 4.0 & 0.0 & 3.50 & 9.50 & 8.0 & 5.0 \\ 6.0 & 5.0 & 6.50 & 0.0 & 4.50 & 1.50 & 3.0 \\ 4.0 & 3.50 & 6.0 & 3.0 & 0.0 & 3.50 & 4.50 \\ 4.50 & 2.0 & 8.50 & 1.0 & 4.0 & 0.0 & 2.0 \\ 3.0 & 1.50 & 4.50 & 2.0 & 6.0 & 4.0 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 5

Number of nodes: 7
Number of configurations: 2

Environment process generator:

$$Q_E = \begin{bmatrix} -4.0 & 4.0 \\ 6.0 & -6.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.60,\ 0.40]$$

Table D.26. Link Numbers and Capacities— Network 8, Parameter Set 5

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 7.50 & 9.0 & 7.0 & 5.0 & 5.50 & 3.50 \\ 6.0 & 0.0 & 5.0 & 4.0 & 3.0 & 3.0 & 2.50 \\ 8.0 & 5.50 & 0.0 & 6.50 & 4.0 & 2.50 & 3.0 \\ 8.50 & 5.50 & 5.0 & 0.0 & 1.50 & 3.50 & 2.0 \\ 6.0 & 4.0 & 4.50 & 4.0 & 0.0 & 2.0 & 4.50 \\ 4.50 & 3.50 & 4.50 & 3.50 & 1.50 & 0.0 & 1.0 \\ 5.0 & 2.0 & 5.0 & 2.50 & 4.0 & 2.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 2.50 & 3.50 & 2.250 & 1.50 & 1.750 & 1.0 \\ 2.0 & 0.0 & 2.750 & 1.250 & 1.250 & 1.50 & 1.750 \\ 5.0 & 2.0 & 0.0 & 1.750 & 4.750 & 4.0 & 2.50 \\ 3.0 & 2.50 & 3.250 & 0.0 & 2.250 & 0.750 & 1.50 \\ 2.0 & 1.750 & 3.0 & 1.50 & 0.0 & 1.750 & 2.250 \\ 2.250 & 1.0 & 4.250 & 0.50 & 2.0 & 0.0 & 1.0 \\ 1.50 & 0.750 & 2.250 & 1.0 & 3.0 & 2.0 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 6

Number of nodes: 7
Number of configurations: 2

Environment process generator:

$$Q_E = \begin{bmatrix} -4.0 & 4.0 \\ 6.0 & -6.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.60, 0.40]$$

Table D.27. Link Numbers and Capacities— Network 8, Parameter Set 6

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

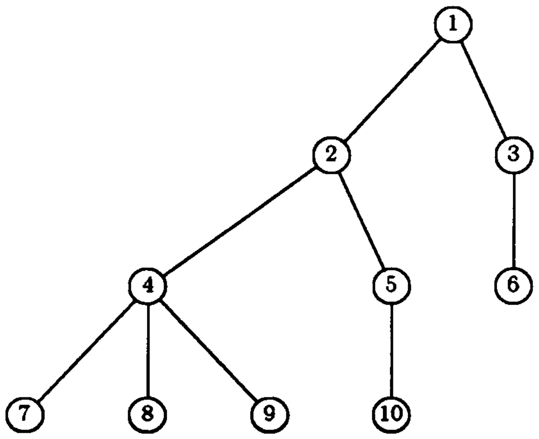
Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

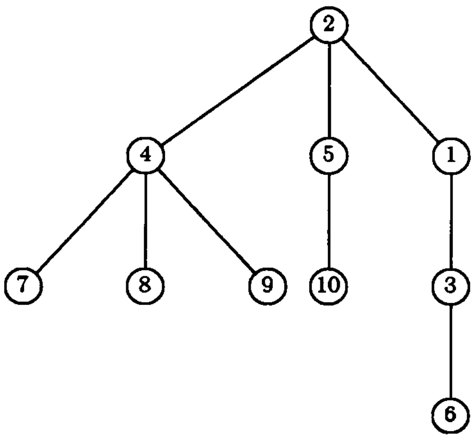
$$\Gamma^{(1)} = \begin{bmatrix} 0.0 & 30.0 & 36.0 & 28.0 & 20.0 & 22.0 & 14.0 \\ 24.0 & 0.0 & 20.0 & 16.0 & 12.0 & 12.0 & 10.0 \\ 32.0 & 22.0 & 0.0 & 26.0 & 16.0 & 10.0 & 12.0 \\ 34.0 & 22.0 & 20.0 & 0.0 & 6.0 & 14.0 & 8.0 \\ 24.0 & 16.0 & 18.0 & 16.0 & 0.0 & 8.0 & 18.0 \\ 18.0 & 14.0 & 18.0 & 14.0 & 6.0 & 0.0 & 4.0 \\ 20.0 & 8.0 & 20.0 & 10.0 & 16.0 & 8.0 & 0.0 \end{bmatrix}$$

$$\Gamma^{(2)} = \begin{bmatrix} 0.0 & 10.0 & 14.0 & 9.0 & 6.0 & 7.0 & 4.0 \\ 8.0 & 0.0 & 11.0 & 5.0 & 5.0 & 6.0 & 7.0 \\ 20.0 & 8.0 & 0.0 & 7.0 & 19.0 & 16.0 & 10.0 \\ 12.0 & 10.0 & 13.0 & 0.0 & 9.0 & 3.0 & 6.0 \\ 8.0 & 7.0 & 12.0 & 6.0 & 0.0 & 7.0 & 9.0 \\ 9.0 & 4.0 & 17.0 & 2.0 & 8.0 & 0.0 & 4.0 \\ 6.0 & 3.0 & 9.0 & 4.0 & 12.0 & 8.0 & 0.0 \end{bmatrix}$$

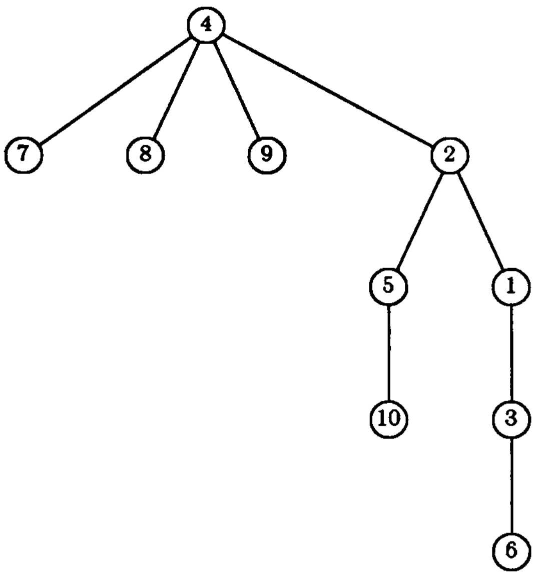
D.9. Network 9



Configuration 1



Configuration 2



Configuration 3

Figure D.9. Dynamic Hierarchy 9 Topology

Network 9, Parameter Set 1

Number of nodes: 10
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -7.0 & 4.0 & 3.0 \\ 8.0 & -10.0 & 2.0 \\ 6.0 & 3.0 & -9.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5030, 0.2695, 0.2275]$$

Table D.28. Link Numbers and Capacities— Network 9, Parameter Set 1

link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \begin{vmatrix} 0.0 & 8.0 & 8.0 & 5.0 & 5.0 & 6.0 & 4.0 & 3.0 & 4.0 & 2.0 \\ 9.0 & 0.0 & 3.0 & 10.0 & 4.0 & 2.0 & 9.0 & 10.0 & 5.0 & 6.0 \\ 7.0 & 3.0 & 0.0 & 2.0 & 2.0 & 11.0 & 1.0 & 1.0 & 2.0 & 1.0 \\ 4.0 & 12.0 & 2.0 & 0.0 & 4.0 & 2.0 & 18.0 & 15.0 & 17.0 & 9.0 \\ 3.0 & 7.0 & 1.0 & 3.0 & 0.0 & 2.0 & 8.0 & 6.0 & 10.0 & 11.0 \\ 4.0 & 3.0 & 13.0 & 3.0 & 2.0 & 0.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 3.0 & 7.0 & 2.0 & 20.0 & 10.0 & 1.0 & 0.0 & 10.0 & 10.0 & 2.0 \\ 4.0 & 10.0 & 1.0 & 18.0 & 12.0 & 1.0 & 11.0 & 0.0 & 13.0 & 2.0 \\ 3.0 & 8.0 & 2.0 & 19.0 & 8.0 & 1.0 & 9.0 & 12.0 & 0.0 & 2.0 \\ 2.0 & 5.0 & 1.0 & 7.0 & 12.0 & 1.0 & 3.0 & 3.0 & 3.0 & 0.0 \end{vmatrix}$$

$$\Gamma^{(2)} = \begin{vmatrix} 0.0 & 7.0 & 6.0 & 7.0 & 9.0 & 7.0 & 1.0 & 3.0 & 1.0 & 3.0 \\ 4.0 & 0.0 & 5.0 & 7.0 & 11.0 & 4.0 & 2.0 & 2.0 & 3.0 & 11.0 \\ 8.0 & 6.0 & 0.0 & 2.0 & 5.0 & 12.0 & 1.0 & 1.0 & 1.0 & 2.0 \\ 6.0 & 10.0 & 1.0 & 0.0 & 7.0 & 1.0 & 10.0 & 11.0 & 10.0 & 6.0 \\ 9.0 & 12.0 & 4.0 & 9.0 & 0.0 & 3.0 & 4.0 & 4.0 & 3.0 & 17.0 \\ 8.0 & 4.0 & 15.0 & 2.0 & 3.0 & 0.0 & 1.0 & 1.0 & 1.0 & 3.0 \\ 3.0 & 5.0 & 2.0 & 12.0 & 3.0 & 1.0 & 0.0 & 14.0 & 6.0 & 2.0 \\ 3.0 & 6.0 & 2.0 & 14.0 & 4.0 & 1.0 & 15.0 & 0.0 & 6.0 & 1.0 \\ 3.0 & 4.0 & 3.0 & 12.0 & 4.0 & 1.0 & 7.0 & 9.0 & 0.0 & 3.0 \\ 6.0 & 13.0 & 4.0 & 8.0 & 19.0 & 2.0 & 4.0 & 2.0 & 3.0 & 0.0 \end{vmatrix}$$

$$\Gamma^{(3)} = \begin{vmatrix} 0.0 & 10.0 & 14.0 & 8.0 & 4.0 & 10.0 & 2.0 & 4.0 & 4.0 & 3.0 \\ 8.0 & 0.0 & 10.0 & 9.0 & 6.0 & 7.0 & 7.0 & 8.0 & 6.0 & 9.0 \\ 15.0 & 11.0 & 0.0 & 6.0 & 3.0 & 20.0 & 1.0 & 2.0 & 2.0 & 2.0 \\ 6.0 & 11.0 & 5.0 & 0.0 & 3.0 & 5.0 & 9.0 & 8.0 & 8.0 & 4.0 \\ 4.0 & 9.0 & 4.0 & 6.0 & 0.0 & 2.0 & 2.0 & 2.0 & 3.0 & 13.0 \\ 12.0 & 8.0 & 20.0 & 5.0 & 2.0 & 0.0 & 2.0 & 1.0 & 1.0 & 3.0 \\ 4.0 & 6.0 & 2.0 & 10.0 & 3.0 & 1.0 & 0.0 & 10.0 & 8.0 & 3.0 \\ 5.0 & 8.0 & 3.0 & 12.0 & 2.0 & 1.0 & 11.0 & 0.0 & 7.0 & 2.0 \\ 4.0 & 9.0 & 3.0 & 11.0 & 4.0 & 1.0 & 10.0 & 6.0 & 0.0 & 2.0 \\ 2.0 & 7.0 & 4.0 & 4.0 & 14.0 & 2.0 & 2.0 & 2.0 & 3.0 & 0.0 \end{vmatrix}$$

Network 9, Parameter Set 2

Number of nodes: 10
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -35.0 & 20.0 & 15.0 \\ 40.0 & -50.0 & 10.0 \\ 30.0 & 15.0 & -45.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5030, 0.2695, 0.2275]$$

Table D.29. Link Numbers and Capacities— Network 9, Parameter Set 2

link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$\Gamma^{(1)} =$

$$\left|\begin{array}{cccccccccc}
0.0 & 8.0 & 8.0 & 5.0 & 5.0 & 6.0 & 4.0 & 3.0 & 4.0 & 2.0 \\
9.0 & 0.0 & 3.0 & 10.0 & 4.0 & 2.0 & 9.0 & 10.0 & 5.0 & 6.0 \\
7.0 & 3.0 & 0.0 & 2.0 & 2.0 & 11.0 & 1.0 & 1.0 & 2.0 & 1.0 \\
4.0 & 12.0 & 2.0 & 0.0 & 4.0 & 2.0 & 18.0 & 15.0 & 17.0 & 9.0 \\
3.0 & 7.0 & 1.0 & 3.0 & 0.0 & 2.0 & 8.0 & 6.0 & 10.0 & 11.0 \\
4.0 & 3.0 & 13.0 & 3.0 & 2.0 & 0.0 & 1.0 & 1.0 & 1.0 & 1.0 \\
3.0 & 7.0 & 2.0 & 20.0 & 10.0 & 1.0 & 0.0 & 10.0 & 10.0 & 2.0 \\
4.0 & 10.0 & 1.0 & 18.0 & 12.0 & 1.0 & 11.0 & 0.0 & 13.0 & 2.0 \\
3.0 & 8.0 & 2.0 & 19.0 & 8.0 & 1.0 & 9.0 & 12.0 & 0.0 & 2.0 \\
2.0 & 5.0 & 1.0 & 7.0 & 12.0 & 1.0 & 3.0 & 3.0 & 3.0 & 0.0
\end{array}\right|$$

$\Gamma^{(2)} =$

$$\left|\begin{array}{cccccccccc}
0.0 & 7.0 & 6.0 & 7.0 & 9.0 & 7.0 & 1.0 & 3.0 & 1.0 & 3.0 \\
4.0 & 0.0 & 5.0 & 7.0 & 11.0 & 4.0 & 2.0 & 2.0 & 3.0 & 11.0 \\
8.0 & 6.0 & 0.0 & 2.0 & 5.0 & 12.0 & 1.0 & 1.0 & 1.0 & 2.0 \\
6.0 & 10.0 & 1.0 & 0.0 & 7.0 & 1.0 & 10.0 & 11.0 & 10.0 & 6.0 \\
9.0 & 12.0 & 4.0 & 9.0 & 0.0 & 3.0 & 4.0 & 4.0 & 3.0 & 17.0 \\
8.0 & 4.0 & 15.0 & 2.0 & 3.0 & 0.0 & 1.0 & 1.0 & 1.0 & 3.0 \\
3.0 & 5.0 & 2.0 & 12.0 & 3.0 & 1.0 & 0.0 & 14.0 & 6.0 & 2.0 \\
3.0 & 6.0 & 2.0 & 14.0 & 4.0 & 1.0 & 15.0 & 0.0 & 6.0 & 1.0 \\
3.0 & 4.0 & 3.0 & 12.0 & 4.0 & 1.0 & 7.0 & 9.0 & 0.0 & 3.0 \\
6.0 & 13.0 & 4.0 & 8.0 & 19.0 & 2.0 & 4.0 & 2.0 & 3.0 & 0.0
\end{array}\right|$$

$\Gamma^{(3)} =$

$$\begin{vmatrix}
0.0 & 10.0 & 14.0 & 8.0 & 4.0 & 10.0 & 2.0 & 4.0 & 4.0 & 3.0 \\
8.0 & 0.0 & 10.0 & 9.0 & 6.0 & 7.0 & 7.0 & 8.0 & 6.0 & 9.0 \\
15.0 & 11.0 & 0.0 & 6.0 & 3.0 & 20.0 & 1.0 & 2.0 & 2.0 & 2.0 \\
6.0 & 11.0 & 5.0 & 0.0 & 3.0 & 5.0 & 9.0 & 8.0 & 8.0 & 4.0 \\
4.0 & 9.0 & 4.0 & 6.0 & 0.0 & 2.0 & 2.0 & 2.0 & 3.0 & 13.0 \\
12.0 & 8.0 & 20.0 & 5.0 & 2.0 & 0.0 & 2.0 & 1.0 & 1.0 & 3.0 \\
4.0 & 6.0 & 2.0 & 10.0 & 3.0 & 1.0 & 0.0 & 10.0 & 8.0 & 3.0 \\
5.0 & 8.0 & 3.0 & 12.0 & 2.0 & 1.0 & 11.0 & 0.0 & 7.0 & 2.0 \\
4.0 & 9.0 & 3.0 & 11.0 & 4.0 & 1.0 & 10.0 & 6.0 & 0.0 & 2.0 \\
2.0 & 7.0 & 4.0 & 4.0 & 14.0 & 2.0 & 2.0 & 2.0 & 3.0 & 0.0
\end{vmatrix}$$

Network 9, Parameter Set 3

Number of nodes: 10
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -7.0 & 4.0 & 3.0 \\ 8.0 & -10.0 & 2.0 \\ 6.0 & 3.0 & -9.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5030,\ 0.2695,\ 0.2275]$$

Table D.30. Link Numbers and Capacities— Network 9, Parameter Set 3			
link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} = \left|\begin{array}{cccccccccc}
0.0 & 16.0 & 16.0 & 10.0 & 10.0 & 12.0 & 8.0 & 6.0 & 8.0 & 4.0 \\
18.0 & 0.0 & 6.0 & 20.0 & 8.0 & 4.0 & 18.0 & 20.0 & 10.0 & 12.0 \\
14.0 & 6.0 & 0.0 & 4.0 & 4.0 & 22.0 & 2.0 & 2.0 & 4.0 & 2.0 \\
8.0 & 24.0 & 4.0 & 0.0 & 8.0 & 4.0 & 36.0 & 30.0 & 34.0 & 18.0 \\
6.0 & 14.0 & 2.0 & 6.0 & 0.0 & 4.0 & 16.0 & 12.0 & 20.0 & 22.0 \\
8.0 & 6.0 & 26.0 & 6.0 & 4.0 & 0.0 & 2.0 & 2.0 & 2.0 & 2.0 \\
6.0 & 14.0 & 4.0 & 40.0 & 20.0 & 2.0 & 0.0 & 20.0 & 20.0 & 4.0 \\
8.0 & 20.0 & 2.0 & 36.0 & 24.0 & 2.0 & 22.0 & 0.0 & 26.0 & 4.0 \\
6.0 & 16.0 & 4.0 & 38.0 & 16.0 & 2.0 & 18.0 & 24.0 & 0.0 & 4.0 \\
4.0 & 10.0 & 2.0 & 14.0 & 24.0 & 2.0 & 6.0 & 6.0 & 6.0 & 0.0
\end{array}\right|$$

$$\Gamma^{(2)} = \left|\begin{array}{cccccccccc}
0.0 & 14.0 & 12.0 & 14.0 & 18.0 & 14.0 & 2.0 & 6.0 & 2.0 & 6.0 \\
8.0 & 0.0 & 10.0 & 14.0 & 22.0 & 8.0 & 4.0 & 4.0 & 6.0 & 22.0 \\
16.0 & 12.0 & 0.0 & 4.0 & 10.0 & 24.0 & 2.0 & 2.0 & 2.0 & 4.0 \\
12.0 & 20.0 & 2.0 & 0.0 & 14.0 & 2.0 & 20.0 & 22.0 & 20.0 & 12.0 \\
18.0 & 24.0 & 8.0 & 18.0 & 0.0 & 6.0 & 8.0 & 8.0 & 6.0 & 34.0 \\
16.0 & 8.0 & 30.0 & 4.0 & 6.0 & 0.0 & 2.0 & 2.0 & 2.0 & 6.0 \\
6.0 & 10.0 & 4.0 & 24.0 & 6.0 & 2.0 & 0.0 & 28.0 & 12.0 & 4.0 \\
6.0 & 12.0 & 4.0 & 28.0 & 8.0 & 2.0 & 30.0 & 0.0 & 12.0 & 2.0 \\
6.0 & 8.0 & 6.0 & 24.0 & 8.0 & 2.0 & 14.0 & 18.0 & 0.0 & 6.0 \\
12.0 & 26.0 & 8.0 & 16.0 & 38.0 & 4.0 & 8.0 & 4.0 & 6.0 & 0.0
\end{array}\right|$$

$\Gamma^{(3)} =$

$$\begin{vmatrix}
0.0 & 20.0 & 28.0 & 16.0 & 8.0 & 20.0 & 4.0 & 8.0 & 8.0 & 6.0 \\
16.0 & 0.0 & 20.0 & 18.0 & 12.0 & 14.0 & 14.0 & 16.0 & 12.0 & 18.0 \\
30.0 & 22.0 & 0.0 & 12.0 & 6.0 & 40.0 & 2.0 & 4.0 & 4.0 & 4.0 \\
12.0 & 22.0 & 10.0 & 0.0 & 6.0 & 10.0 & 18.0 & 16.0 & 16.0 & 8.0 \\
8.0 & 18.0 & 8.0 & 12.0 & 0.0 & 4.0 & 4.0 & 4.0 & 6.0 & 26.0 \\
24.0 & 16.0 & 40.0 & 10.0 & 4.0 & 0.0 & 4.0 & 2.0 & 2.0 & 6.0 \\
8.0 & 12.0 & 4.0 & 20.0 & 6.0 & 2.0 & 0.0 & 20.0 & 16.0 & 6.0 \\
10.0 & 16.0 & 6.0 & 24.0 & 4.0 & 2.0 & 22.0 & 0.0 & 14.0 & 4.0 \\
8.0 & 18.0 & 6.0 & 22.0 & 8.0 & 2.0 & 20.0 & 12.0 & 0.0 & 4.0 \\
4.0 & 14.0 & 8.0 & 8.0 & 28.0 & 4.0 & 4.0 & 4.0 & 6.0 & 0.0
\end{vmatrix}$$

Network 9, Parameter Set 4

Number of nodes: 10
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -35.0 & 20.0 & 15.0 \\ 40.0 & -50.0 & 10.0 \\ 30.0 & 15.0 & -45.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.5030,\ 0.2695,\ 0.2275]$$

Table D.31. Link Numbers and Capacities— Network 9, Parameter Set 4			
link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

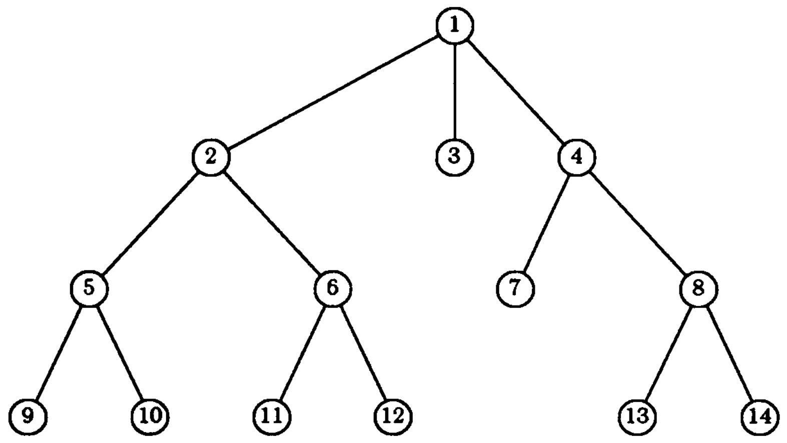
Arrival rate matrices:

$$\Gamma^{(1)} = \begin{vmatrix}
0.0 & 16.0 & 16.0 & 10.0 & 10.0 & 12.0 & 8.0 & 6.0 & 8.0 & 4.0 \\
18.0 & 0.0 & 6.0 & 20.0 & 8.0 & 4.0 & 18.0 & 20.0 & 10.0 & 12.0 \\
14.0 & 6.0 & 0.0 & 4.0 & 4.0 & 22.0 & 2.0 & 2.0 & 4.0 & 2.0 \\
8.0 & 24.0 & 4.0 & 0.0 & 8.0 & 4.0 & 36.0 & 30.0 & 34.0 & 18.0 \\
6.0 & 14.0 & 2.0 & 6.0 & 0.0 & 4.0 & 16.0 & 12.0 & 20.0 & 22.0 \\
8.0 & 6.0 & 26.0 & 6.0 & 4.0 & 0.0 & 2.0 & 2.0 & 2.0 & 2.0 \\
6.0 & 14.0 & 4.0 & 40.0 & 20.0 & 2.0 & 0.0 & 20.0 & 20.0 & 4.0 \\
8.0 & 20.0 & 2.0 & 36.0 & 24.0 & 2.0 & 22.0 & 0.0 & 26.0 & 4.0 \\
6.0 & 16.0 & 4.0 & 38.0 & 16.0 & 2.0 & 18.0 & 24.0 & 0.0 & 4.0 \\
4.0 & 10.0 & 2.0 & 14.0 & 24.0 & 2.0 & 6.0 & 6.0 & 6.0 & 0.0
\end{vmatrix}$$

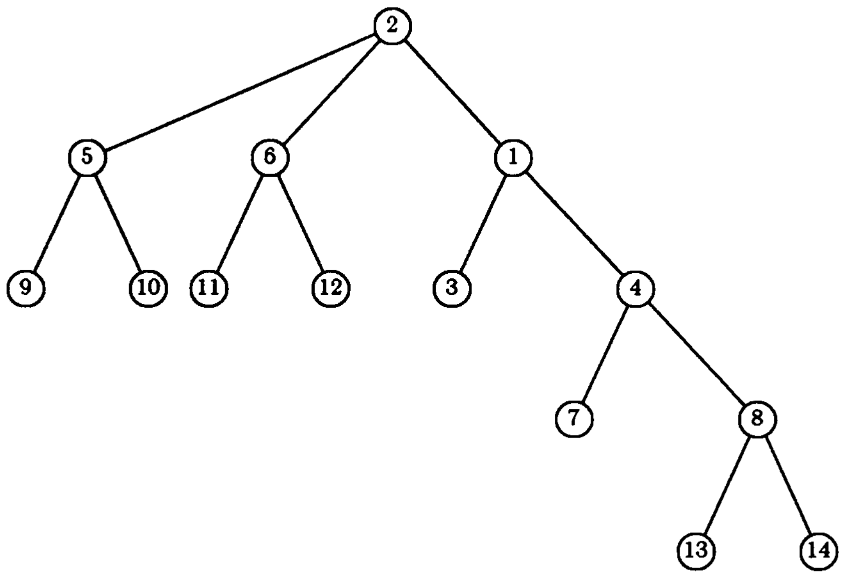
$$\Gamma^{(2)} = \begin{vmatrix}
0.0 & 14.0 & 12.0 & 14.0 & 18.0 & 14.0 & 2.0 & 6.0 & 2.0 & 6.0 \\
8.0 & 0.0 & 10.0 & 14.0 & 22.0 & 8.0 & 4.0 & 4.0 & 6.0 & 22.0 \\
16.0 & 12.0 & 0.0 & 4.0 & 10.0 & 24.0 & 2.0 & 2.0 & 2.0 & 4.0 \\
12.0 & 20.0 & 2.0 & 0.0 & 14.0 & 2.0 & 20.0 & 22.0 & 20.0 & 12.0 \\
18.0 & 24.0 & 8.0 & 18.0 & 0.0 & 6.0 & 8.0 & 8.0 & 6.0 & 34.0 \\
16.0 & 8.0 & 30.0 & 4.0 & 6.0 & 0.0 & 2.0 & 2.0 & 2.0 & 6.0 \\
6.0 & 10.0 & 4.0 & 24.0 & 6.0 & 2.0 & 0.0 & 28.0 & 12.0 & 4.0 \\
6.0 & 12.0 & 4.0 & 28.0 & 8.0 & 2.0 & 30.0 & 0.0 & 12.0 & 2.0 \\
6.0 & 8.0 & 6.0 & 24.0 & 8.0 & 2.0 & 14.0 & 18.0 & 0.0 & 6.0 \\
12.0 & 26.0 & 8.0 & 16.0 & 38.0 & 4.0 & 8.0 & 4.0 & 6.0 & 0.0
\end{vmatrix}$$

$$\Gamma^{(3)} = \begin{vmatrix}
0.0 & 20.0 & 28.0 & 16.0 & 8.0 & 20.0 & 4.0 & 8.0 & 8.0 & 6.0 \\
16.0 & 0.0 & 20.0 & 18.0 & 12.0 & 14.0 & 14.0 & 16.0 & 12.0 & 18.0 \\
30.0 & 22.0 & 0.0 & 12.0 & 6.0 & 40.0 & 2.0 & 4.0 & 4.0 & 4.0 \\
12.0 & 22.0 & 10.0 & 0.0 & 6.0 & 10.0 & 18.0 & 16.0 & 16.0 & 8.0 \\
8.0 & 18.0 & 8.0 & 12.0 & 0.0 & 4.0 & 4.0 & 4.0 & 6.0 & 26.0 \\
24.0 & 16.0 & 40.0 & 10.0 & 4.0 & 0.0 & 4.0 & 2.0 & 2.0 & 6.0 \\
8.0 & 12.0 & 4.0 & 20.0 & 6.0 & 2.0 & 0.0 & 20.0 & 16.0 & 6.0 \\
10.0 & 16.0 & 6.0 & 24.0 & 4.0 & 2.0 & 22.0 & 0.0 & 14.0 & 4.0 \\
8.0 & 18.0 & 6.0 & 22.0 & 8.0 & 2.0 & 20.0 & 12.0 & 0.0 & 4.0 \\
4.0 & 14.0 & 8.0 & 8.0 & 28.0 & 4.0 & 4.0 & 4.0 & 6.0 & 0.0
\end{vmatrix}$$

D.10. Network 10

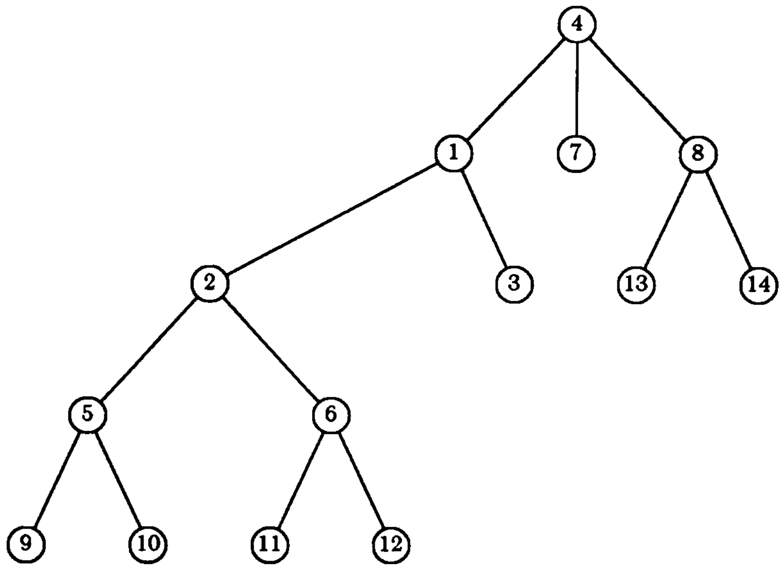


Configuration 1



Configuration 2

Figure D.10(a). Dynamic Hierarchy 10 Topology



Configuration 3

Figure D.10(b). Dynamic Hierarchy 10 Topology (continued)

Network 10, Parameter Set 1

Number of nodes: 14
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -20.0 & 8.0 & 12.0 \\ 10.0 & -19.0 & 9.0 \\ 7.0 & 15.0 & -22.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\pi = [0.3014, 0.3791, 0.3195]$$

Table D.32. Link Numbers and Capacities— Network 10, Parameter Set 1

link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$$\Gamma^{(1)} =$$

$$\left|\begin{array}{cccccccccccccc}
0.0 & 10.0 & 2.0 & 2.0 & 0.0 & 2.0 & 3.0 & 5.0 & 9.0 & 6.0 & 20.0 & 12.0 & 13.0 & 3.0 \\
12.0 & 0.0 & 5.0 & 7.0 & 6.0 & 11.0 & 17.0 & 7.0 & 13.0 & 4.0 & 0.0 & 16.0 & 4.0 & 1.0 \\
17.0 & 11.0 & 0.0 & 0.0 & 6.0 & 5.0 & 18.0 & 18.0 & 10.0 & 0.0 & 4.0 & 16.0 & 3.0 & 15.0 \\
19.0 & 2.0 & 10.0 & 0.0 & 14.0 & 4.0 & 1.0 & 5.0 & 7.0 & 4.0 & 13.0 & 15.0 & 7.0 & 4.0 \\
7.0 & 2.0 & 4.0 & 11.0 & 0.0 & 10.0 & 14.0 & 18.0 & 17.0 & 17.0 & 14.0 & 12.0 & 15.0 & 9.0 \\
12.0 & 9.0 & 12.0 & 17.0 & 10.0 & 0.0 & 11.0 & 12.0 & 15.0 & 14.0 & 7.0 & 0.0 & 1.0 & 1.0 \\
18.0 & 10.0 & 16.0 & 0.0 & 7.0 & 17.0 & 0.0 & 10.0 & 18.0 & 17.0 & 13.0 & 18.0 & 10.0 & 6.0 \\
3.0 & 17.0 & 16.0 & 18.0 & 7.0 & 15.0 & 17.0 & 0.0 & 13.0 & 19.0 & 8.0 & 15.0 & 7.0 & 9.0 \\
14.0 & 5.0 & 11.0 & 1.0 & 16.0 & 12.0 & 8.0 & 13.0 & 0.0 & 6.0 & 18.0 & 13.0 & 9.0 & 16.0 \\
10.0 & 13.0 & 7.0 & 2.0 & 10.0 & 13.0 & 11.0 & 6.0 & 17.0 & 0.0 & 3.0 & 14.0 & 5.0 & 15.0 \\
19.0 & 2.0 & 13.0 & 10.0 & 18.0 & 14.0 & 19.0 & 17.0 & 11.0 & 18.0 & 0.0 & 10.0 & 14.0 & 7.0 \\
5.0 & 7.0 & 4.0 & 17.0 & 11.0 & 16.0 & 14.0 & 1.0 & 18.0 & 11.0 & 7.0 & 0.0 & 5.0 & 2.0 \\
7.0 & 18.0 & 17.0 & 2.0 & 4.0 & 1.0 & 16.0 & 13.0 & 7.0 & 10.0 & 4.0 & 17.0 & 0.0 & 6.0 \\
0.0 & 18.0 & 16.0 & 17.0 & 8.0 & 14.0 & 7.0 & 7.0 & 17.0 & 4.0 & 11.0 & 18.0 & 13.0 & 0.0
\end{array}\right|$$

$$\Gamma^{(2)} =$$

$$\begin{vmatrix}
0.0 & 3.0 & 15.0 & 2.0 & 6.0 & 12.0 & 13.0 & 4.0 & 8.0 & 10.0 & 19.0 & 0.0 & 7.0 & 16.0 \\
15.0 & 0.0 & 18.0 & 3.0 & 11.0 & 14.0 & 13.0 & 9.0 & 15.0 & 7.0 & 8.0 & 9.0 & 13.0 & 6.0 \\
3.0 & 10.0 & 0.0 & 15.0 & 2.0 & 11.0 & 12.0 & 19.0 & 5.0 & 2.0 & 13.0 & 18.0 & 19.0 & 2.0 \\
13.0 & 15.0 & 1.0 & 0.0 & 0.0 & 11.0 & 9.0 & 3.0 & 11.0 & 15.0 & 15.0 & 4.0 & 10.0 & 6.0 \\
9.0 & 1.0 & 8.0 & 6.0 & 0.0 & 12.0 & 2.0 & 9.0 & 3.0 & 6.0 & 18.0 & 7.0 & 3.0 & 3.0 \\
19.0 & 4.0 & 7.0 & 0.0 & 5.0 & 0.0 & 15.0 & 10.0 & 10.0 & 2.0 & 3.0 & 16.0 & 2.0 & 15.0 \\
4.0 & 9.0 & 11.0 & 10.0 & 19.0 & 6.0 & 0.0 & 4.0 & 9.0 & 4.0 & 15.0 & 15.0 & 6.0 & 4.0 \\
3.0 & 0.0 & 11.0 & 5.0 & 3.0 & 10.0 & 5.0 & 0.0 & 14.0 & 14.0 & 6.0 & 5.0 & 1.0 & 18.0 \\
10.0 & 0.0 & 16.0 & 5.0 & 4.0 & 7.0 & 10.0 & 2.0 & 0.0 & 1.0 & 11.0 & 12.0 & 14.0 & 4.0 \\
17.0 & 17.0 & 1.0 & 16.0 & 15.0 & 19.0 & 17.0 & 3.0 & 0.0 & 0.0 & 6.0 & 16.0 & 1.0 & 16.0 \\
13.0 & 15.0 & 7.0 & 8.0 & 17.0 & 9.0 & 15.0 & 6.0 & 10.0 & 1.0 & 0.0 & 18.0 & 4.0 & 7.0 \\
4.0 & 12.0 & 5.0 & 3.0 & 0.0 & 13.0 & 18.0 & 12.0 & 10.0 & 12.0 & 2.0 & 0.0 & 18.0 & 7.0 \\
15.0 & 15.0 & 12.0 & 13.0 & 1.0 & 3.0 & 6.0 & 1.0 & 17.0 & 7.0 & 3.0 & 4.0 & 0.0 & 17.0 \\
18.0 & 3.0 & 15.0 & 18.0 & 2.0 & 10.0 & 13.0 & 1.0 & 2.0 & 8.0 & 15.0 & 13.0 & 11.0 & 0.0
\end{vmatrix}$$

$$\Gamma^{(3)} =$$

$$\begin{vmatrix}
0.0 & 5.0 & 15.0 & 14.0 & 14.0 & 17.0 & 15.0 & 19.0 & 17.0 & 10.0 & 9.0 & 17.0 & 14.0 & 7.0 \\
4.0 & 0.0 & 6.0 & 13.0 & 10.0 & 13.0 & 15.0 & 5.0 & 7.0 & 3.0 & 8.0 & 0.0 & 7.0 & 3.0 \\
9.0 & 8.0 & 0.0 & 18.0 & 4.0 & 2.0 & 0.0 & 2.0 & 5.0 & 4.0 & 8.0 & 0.0 & 11.0 & 4.0 \\
13.0 & 10.0 & 0.0 & 0.0 & 15.0 & 2.0 & 4.0 & 10.0 & 8.0 & 3.0 & 16.0 & 19.0 & 4.0 & 15.0 \\
6.0 & 7.0 & 1.0 & 18.0 & 0.0 & 6.0 & 12.0 & 12.0 & 10.0 & 16.0 & 13.0 & 11.0 & 18.0 & 18.0 \\
10.0 & 4.0 & 9.0 & 18.0 & 3.0 & 0.0 & 16.0 & 5.0 & 14.0 & 17.0 & 9.0 & 16.0 & 2.0 & 10.0 \\
1.0 & 6.0 & 3.0 & 8.0 & 18.0 & 1.0 & 0.0 & 13.0 & 15.0 & 9.0 & 12.0 & 6.0 & 18.0 & 3.0 \\
5.0 & 15.0 & 17.0 & 12.0 & 14.0 & 11.0 & 19.0 & 0.0 & 6.0 & 12.0 & 19.0 & 8.0 & 16.0 & 13.0 \\
18.0 & 3.0 & 16.0 & 17.0 & 11.0 & 5.0 & 8.0 & 9.0 & 0.0 & 8.0 & 18.0 & 2.0 & 11.0 & 19.0 \\
2.0 & 15.0 & 13.0 & 18.0 & 19.0 & 12.0 & 11.0 & 4.0 & 14.0 & 0.0 & 19.0 & 18.0 & 11.0 & 12.0 \\
16.0 & 14.0 & 8.0 & 3.0 & 15.0 & 14.0 & 13.0 & 7.0 & 14.0 & 2.0 & 0.0 & 19.0 & 7.0 & 0.0 \\
6.0 & 4.0 & 4.0 & 4.0 & 9.0 & 11.0 & 0.0 & 10.0 & 0.0 & 16.0 & 3.0 & 0.0 & 11.0 & 7.0 \\
19.0 & 12.0 & 16.0 & 4.0 & 13.0 & 2.0 & 8.0 & 11.0 & 9.0 & 10.0 & 15.0 & 5.0 & 0.0 & 12.0 \\
7.0 & 3.0 & 18.0 & 15.0 & 5.0 & 13.0 & 8.0 & 5.0 & 2.0 & 18.0 & 4.0 & 14.0 & 0.0 & 0.0
\end{vmatrix}$$

Network 10, Parameter Set 2

Number of nodes: 14
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -100.0 & 40.0 & 60.0 \\ 50.0 & -95.0 & 45.0 \\ 35.0 & 75.0 & -110.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.3014, 0.3791, 0.3195]$$

Table D.33. Link Numbers and Capacities— Network 10, Parameter Set 2			
link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$\Gamma^{(1)} =$

$$\left|\begin{array}{cccccccccccccc}
0.0 & 10.0 & 2.0 & 2.0 & 0.0 & 2.0 & 3.0 & 5.0 & 9.0 & 6.0 & 20.0 & 12.0 & 13.0 & 3.0 \\
12.0 & 0.0 & 5.0 & 7.0 & 6.0 & 11.0 & 17.0 & 7.0 & 13.0 & 4.0 & 0.0 & 16.0 & 4.0 & 1.0 \\
17.0 & 11.0 & 0.0 & 0.0 & 6.0 & 5.0 & 18.0 & 18.0 & 10.0 & 0.0 & 4.0 & 16.0 & 3.0 & 15.0 \\
19.0 & 2.0 & 10.0 & 0.0 & 14.0 & 4.0 & 1.0 & 5.0 & 7.0 & 4.0 & 13.0 & 15.0 & 7.0 & 4.0 \\
7.0 & 2.0 & 4.0 & 11.0 & 0.0 & 10.0 & 14.0 & 18.0 & 17.0 & 17.0 & 14.0 & 12.0 & 15.0 & 9.0 \\
12.0 & 9.0 & 12.0 & 17.0 & 10.0 & 0.0 & 11.0 & 12.0 & 15.0 & 14.0 & 7.0 & 0.0 & 1.0 & 1.0 \\
18.0 & 10.0 & 16.0 & 0.0 & 7.0 & 17.0 & 0.0 & 10.0 & 18.0 & 17.0 & 13.0 & 18.0 & 10.0 & 6.0 \\
3.0 & 17.0 & 16.0 & 18.0 & 7.0 & 15.0 & 17.0 & 0.0 & 13.0 & 19.0 & 8.0 & 15.0 & 7.0 & 9.0 \\
14.0 & 5.0 & 11.0 & 1.0 & 16.0 & 12.0 & 8.0 & 13.0 & 0.0 & 6.0 & 18.0 & 13.0 & 9.0 & 16.0 \\
10.0 & 13.0 & 7.0 & 2.0 & 10.0 & 13.0 & 11.0 & 6.0 & 17.0 & 0.0 & 3.0 & 14.0 & 5.0 & 15.0 \\
19.0 & 2.0 & 13.0 & 10.0 & 18.0 & 14.0 & 19.0 & 17.0 & 11.0 & 18.0 & 0.0 & 10.0 & 14.0 & 7.0 \\
5.0 & 7.0 & 4.0 & 17.0 & 11.0 & 16.0 & 14.0 & 1.0 & 18.0 & 11.0 & 7.0 & 0.0 & 5.0 & 2.0 \\
7.0 & 18.0 & 17.0 & 2.0 & 4.0 & 1.0 & 16.0 & 13.0 & 7.0 & 10.0 & 4.0 & 17.0 & 0.0 & 6.0 \\
0.0 & 18.0 & 16.0 & 17.0 & 8.0 & 14.0 & 7.0 & 7.0 & 17.0 & 4.0 & 11.0 & 18.0 & 13.0 & 0.0
\end{array}\right|$$

$$\Gamma^{(2)} =$$

$$\left|\begin{array}{cccccccccccccc}
0.0 & 3.0 & 15.0 & 2.0 & 6.0 & 12.0 & 13.0 & 4.0 & 8.0 & 10.0 & 19.0 & 0.0 & 7.0 & 16.0 \\
15.0 & 0.0 & 18.0 & 3.0 & 11.0 & 14.0 & 13.0 & 9.0 & 15.0 & 7.0 & 8.0 & 9.0 & 13.0 & 6.0 \\
3.0 & 10.0 & 0.0 & 15.0 & 2.0 & 11.0 & 12.0 & 19.0 & 5.0 & 2.0 & 13.0 & 18.0 & 19.0 & 2.0 \\
13.0 & 15.0 & 1.0 & 0.0 & 0.0 & 11.0 & 9.0 & 3.0 & 11.0 & 15.0 & 15.0 & 4.0 & 10.0 & 6.0 \\
9.0 & 1.0 & 8.0 & 6.0 & 0.0 & 12.0 & 2.0 & 9.0 & 3.0 & 6.0 & 18.0 & 7.0 & 3.0 & 3.0 \\
19.0 & 4.0 & 7.0 & 0.0 & 5.0 & 0.0 & 15.0 & 10.0 & 10.0 & 2.0 & 3.0 & 16.0 & 2.0 & 15.0 \\
4.0 & 9.0 & 11.0 & 10.0 & 19.0 & 6.0 & 0.0 & 4.0 & 9.0 & 4.0 & 15.0 & 15.0 & 6.0 & 4.0 \\
3.0 & 0.0 & 11.0 & 5.0 & 3.0 & 10.0 & 5.0 & 0.0 & 14.0 & 14.0 & 6.0 & 5.0 & 1.0 & 18.0 \\
10.0 & 0.0 & 16.0 & 5.0 & 4.0 & 7.0 & 10.0 & 2.0 & 0.0 & 1.0 & 11.0 & 12.0 & 14.0 & 4.0 \\
17.0 & 17.0 & 1.0 & 16.0 & 15.0 & 19.0 & 17.0 & 3.0 & 0.0 & 0.0 & 6.0 & 16.0 & 1.0 & 16.0 \\
13.0 & 15.0 & 7.0 & 8.0 & 17.0 & 9.0 & 15.0 & 6.0 & 10.0 & 1.0 & 0.0 & 18.0 & 4.0 & 7.0 \\
4.0 & 12.0 & 5.0 & 3.0 & 0.0 & 13.0 & 18.0 & 12.0 & 10.0 & 12.0 & 2.0 & 0.0 & 18.0 & 7.0 \\
15.0 & 15.0 & 12.0 & 13.0 & 1.0 & 3.0 & 6.0 & 1.0 & 17.0 & 7.0 & 3.0 & 4.0 & 0.0 & 17.0 \\
18.0 & 3.0 & 15.0 & 18.0 & 2.0 & 10.0 & 13.0 & 1.0 & 2.0 & 8.0 & 15.0 & 13.0 & 11.0 & 0.0
\end{array}\right|$$

$$\Gamma^{(3)} =$$

$$\left|\begin{array}{cccccccccccccc}
0.0 & 5.0 & 15.0 & 14.0 & 14.0 & 17.0 & 15.0 & 19.0 & 17.0 & 10.0 & 9.0 & 17.0 & 14.0 & 7.0 \\
4.0 & 0.0 & 6.0 & 13.0 & 10.0 & 13.0 & 15.0 & 5.0 & 7.0 & 3.0 & 8.0 & 0.0 & 7.0 & 3.0 \\
9.0 & 8.0 & 0.0 & 18.0 & 4.0 & 2.0 & 0.0 & 2.0 & 5.0 & 4.0 & 8.0 & 0.0 & 11.0 & 4.0 \\
13.0 & 10.0 & 0.0 & 0.0 & 15.0 & 2.0 & 4.0 & 10.0 & 8.0 & 3.0 & 16.0 & 19.0 & 4.0 & 15.0 \\
6.0 & 7.0 & 1.0 & 18.0 & 0.0 & 6.0 & 12.0 & 12.0 & 10.0 & 16.0 & 13.0 & 11.0 & 18.0 & 18.0 \\
10.0 & 4.0 & 9.0 & 18.0 & 3.0 & 0.0 & 16.0 & 5.0 & 14.0 & 17.0 & 9.0 & 16.0 & 2.0 & 10.0 \\
1.0 & 6.0 & 3.0 & 8.0 & 18.0 & 1.0 & 0.0 & 13.0 & 15.0 & 9.0 & 12.0 & 6.0 & 18.0 & 3.0 \\
5.0 & 15.0 & 17.0 & 12.0 & 14.0 & 11.0 & 19.0 & 0.0 & 6.0 & 12.0 & 19.0 & 8.0 & 16.0 & 13.0 \\
18.0 & 3.0 & 16.0 & 17.0 & 11.0 & 5.0 & 8.0 & 9.0 & 0.0 & 8.0 & 18.0 & 2.0 & 11.0 & 19.0 \\
2.0 & 15.0 & 13.0 & 18.0 & 19.0 & 12.0 & 11.0 & 4.0 & 14.0 & 0.0 & 19.0 & 18.0 & 11.0 & 12.0 \\
16.0 & 14.0 & 8.0 & 3.0 & 15.0 & 14.0 & 13.0 & 7.0 & 14.0 & 2.0 & 0.0 & 19.0 & 7.0 & 0.0 \\
6.0 & 4.0 & 4.0 & 4.0 & 9.0 & 11.0 & 0.0 & 10.0 & 0.0 & 16.0 & 3.0 & 0.0 & 11.0 & 7.0 \\
19.0 & 12.0 & 16.0 & 4.0 & 13.0 & 2.0 & 8.0 & 11.0 & 9.0 & 10.0 & 15.0 & 5.0 & 0.0 & 12.0 \\
7.0 & 3.0 & 18.0 & 15.0 & 5.0 & 13.0 & 8.0 & 5.0 & 2.0 & 18.0 & 4.0 & 14.0 & 0.0 & 0.0
\end{array}\right|$$

Network 10, Parameter Set 3

Number of nodes: 14
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -20.0 & 8.0 & 12.0 \\ 10.0 & -19.0 & 9.0 \\ 7.0 & 15.0 & -22.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.3014,\ 0.3791,\ 0.3195]$$

Table D.34. Link Numbers and Capacities— Network 10, Parameter Set 3

link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$\Gamma^{(1)} =$

$$\left|\begin{array}{cccccccccccccc}
0.0 & 20.0 & 4.0 & 4.0 & 0.0 & 4.0 & 6.0 & 10.0 & 18.0 & 12.0 & 40.0 & 24.0 & 26.0 & 6.0 \\
24.0 & 0.0 & 10.0 & 14.0 & 12.0 & 22.0 & 34.0 & 14.0 & 26.0 & 8.0 & 0.0 & 32.0 & 8.0 & 2.0 \\
34.0 & 22.0 & 0.0 & 0.0 & 12.0 & 10.0 & 36.0 & 36.0 & 20.0 & 0.0 & 8.0 & 32.0 & 6.0 & 30.0 \\
38.0 & 4.0 & 20.0 & 0.0 & 28.0 & 8.0 & 2.0 & 10.0 & 14.0 & 8.0 & 26.0 & 30.0 & 14.0 & 8.0 \\
14.0 & 4.0 & 8.0 & 22.0 & 0.0 & 20.0 & 28.0 & 36.0 & 34.0 & 34.0 & 28.0 & 24.0 & 30.0 & 18.0 \\
24.0 & 18.0 & 24.0 & 34.0 & 20.0 & 0.0 & 22.0 & 24.0 & 30.0 & 28.0 & 14.0 & 0.0 & 2.0 & 2.0 \\
36.0 & 20.0 & 32.0 & 0.0 & 14.0 & 34.0 & 0.0 & 20.0 & 36.0 & 34.0 & 26.0 & 36.0 & 20.0 & 12.0 \\
6.0 & 34.0 & 32.0 & 36.0 & 14.0 & 30.0 & 34.0 & 0.0 & 26.0 & 38.0 & 16.0 & 30.0 & 14.0 & 18.0 \\
28.0 & 10.0 & 22.0 & 2.0 & 32.0 & 24.0 & 16.0 & 26.0 & 0.0 & 12.0 & 36.0 & 26.0 & 18.0 & 32.0 \\
20.0 & 26.0 & 14.0 & 4.0 & 20.0 & 26.0 & 22.0 & 12.0 & 34.0 & 0.0 & 6.0 & 28.0 & 10.0 & 30.0 \\
38.0 & 4.0 & 26.0 & 20.0 & 36.0 & 28.0 & 38.0 & 34.0 & 22.0 & 36.0 & 0.0 & 20.0 & 28.0 & 14.0 \\
10.0 & 14.0 & 8.0 & 34.0 & 22.0 & 32.0 & 28.0 & 2.0 & 36.0 & 22.0 & 14.0 & 0.0 & 10.0 & 4.0 \\
14.0 & 36.0 & 34.0 & 4.0 & 8.0 & 2.0 & 32.0 & 26.0 & 14.0 & 20.0 & 8.0 & 34.0 & 0.0 & 12.0 \\
0.0 & 36.0 & 32.0 & 34.0 & 16.0 & 28.0 & 14.0 & 14.0 & 34.0 & 8.0 & 22.0 & 36.0 & 26.0 & 0.0
\end{array}\right|$$

$\Gamma^{(2)} =$

$$\begin{vmatrix}
0.0 & 6.0 & 30.0 & 4.0 & 12.0 & 24.0 & 26.0 & 8.0 & 16.0 & 20.0 & 38.0 & 0.0 & 14.0 & 32.0 \\
30.0 & 0.0 & 36.0 & 6.0 & 22.0 & 28.0 & 26.0 & 18.0 & 30.0 & 14.0 & 16.0 & 18.0 & 26.0 & 12.0 \\
6.0 & 20.0 & 0.0 & 30.0 & 4.0 & 22.0 & 24.0 & 38.0 & 10.0 & 4.0 & 26.0 & 36.0 & 38.0 & 4.0 \\
26.0 & 30.0 & 2.0 & 0.0 & 0.0 & 22.0 & 18.0 & 6.0 & 22.0 & 30.0 & 30.0 & 8.0 & 20.0 & 12.0 \\
18.0 & 2.0 & 16.0 & 12.0 & 0.0 & 24.0 & 4.0 & 18.0 & 6.0 & 12.0 & 36.0 & 14.0 & 6.0 & 6.0 \\
38.0 & 8.0 & 14.0 & 0.0 & 10.0 & 0.0 & 30.0 & 20.0 & 20.0 & 4.0 & 6.0 & 32.0 & 4.0 & 30.0 \\
8.0 & 18.0 & 22.0 & 20.0 & 38.0 & 12.0 & 0.0 & 8.0 & 18.0 & 8.0 & 30.0 & 30.0 & 12.0 & 8.0 \\
6.0 & 0.0 & 22.0 & 10.0 & 6.0 & 20.0 & 10.0 & 0.0 & 28.0 & 28.0 & 12.0 & 10.0 & 2.0 & 36.0 \\
20.0 & 0.0 & 32.0 & 10.0 & 8.0 & 14.0 & 20.0 & 4.0 & 0.0 & 2.0 & 22.0 & 24.0 & 28.0 & 8.0 \\
34.0 & 34.0 & 2.0 & 32.0 & 30.0 & 38.0 & 34.0 & 6.0 & 0.0 & 0.0 & 12.0 & 32.0 & 2.0 & 32.0 \\
26.0 & 30.0 & 14.0 & 16.0 & 34.0 & 18.0 & 30.0 & 12.0 & 20.0 & 2.0 & 0.0 & 36.0 & 8.0 & 14.0 \\
8.0 & 24.0 & 10.0 & 6.0 & 0.0 & 26.0 & 36.0 & 24.0 & 20.0 & 24.0 & 4.0 & 0.0 & 36.0 & 14.0 \\
30.0 & 30.0 & 24.0 & 26.0 & 2.0 & 6.0 & 12.0 & 2.0 & 34.0 & 14.0 & 6.0 & 8.0 & 0.0 & 34.0 \\
36.0 & 6.0 & 30.0 & 36.0 & 4.0 & 20.0 & 26.0 & 2.0 & 4.0 & 16.0 & 30.0 & 26.0 & 22.0 & 0.0
\end{vmatrix}$$

$\Gamma^{(3)} =$

$$\begin{vmatrix}
0.0 & 10.0 & 30.0 & 28.0 & 28.0 & 34.0 & 30.0 & 38.0 & 34.0 & 20.0 & 18.0 & 34.0 & 28.0 & 14.0 \\
8.0 & 0.0 & 12.0 & 26.0 & 20.0 & 26.0 & 30.0 & 10.0 & 14.0 & 6.0 & 16.0 & 0.0 & 14.0 & 6.0 \\
18.0 & 16.0 & 0.0 & 36.0 & 8.0 & 4.0 & 0.0 & 4.0 & 10.0 & 8.0 & 16.0 & 0.0 & 22.0 & 8.0 \\
26.0 & 20.0 & 0.0 & 0.0 & 30.0 & 4.0 & 8.0 & 20.0 & 16.0 & 6.0 & 32.0 & 38.0 & 8.0 & 30.0 \\
12.0 & 14.0 & 2.0 & 36.0 & 0.0 & 12.0 & 24.0 & 24.0 & 20.0 & 32.0 & 26.0 & 22.0 & 36.0 & 36.0 \\
20.0 & 8.0 & 18.0 & 36.0 & 6.0 & 0.0 & 32.0 & 10.0 & 28.0 & 34.0 & 18.0 & 32.0 & 4.0 & 20.0 \\
2.0 & 12.0 & 6.0 & 16.0 & 36.0 & 2.0 & 0.0 & 26.0 & 30.0 & 18.0 & 24.0 & 12.0 & 36.0 & 6.0 \\
10.0 & 30.0 & 34.0 & 24.0 & 28.0 & 22.0 & 38.0 & 0.0 & 12.0 & 24.0 & 38.0 & 16.0 & 32.0 & 26.0 \\
36.0 & 6.0 & 32.0 & 34.0 & 22.0 & 10.0 & 16.0 & 18.0 & 0.0 & 16.0 & 36.0 & 4.0 & 22.0 & 38.0 \\
4.0 & 30.0 & 26.0 & 36.0 & 38.0 & 24.0 & 22.0 & 8.0 & 28.0 & 0.0 & 38.0 & 36.0 & 22.0 & 24.0 \\
32.0 & 28.0 & 16.0 & 6.0 & 30.0 & 28.0 & 26.0 & 14.0 & 28.0 & 4.0 & 0.0 & 38.0 & 14.0 & 0.0 \\
12.0 & 8.0 & 8.0 & 8.0 & 18.0 & 22.0 & 0.0 & 20.0 & 0.0 & 32.0 & 6.0 & 0.0 & 22.0 & 14.0 \\
38.0 & 24.0 & 32.0 & 8.0 & 26.0 & 4.0 & 16.0 & 22.0 & 18.0 & 20.0 & 30.0 & 10.0 & 0.0 & 24.0 \\
14.0 & 6.0 & 36.0 & 30.0 & 10.0 & 26.0 & 16.0 & 10.0 & 4.0 & 36.0 & 8.0 & 28.0 & 0.0 & 0.0
\end{vmatrix}$$

Network 10, Parameter Set 4

Number of nodes: 14
Number of configurations: 3

Environment process generator:

$$Q_E = \begin{bmatrix} -100.0 & 40.0 & 60.0 \\ 50.0 & -95.0 & 45.0 \\ 35.0 & 75.0 & -110.0 \end{bmatrix}$$

Environment stationary probabilities:

$$\boldsymbol{\pi} = [0.3014, 0.3791, 0.3195]$$

Table D.35. Link Numbers and Capacities— Network 10, Parameter Set 4			
link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrices:

$\Gamma^{(1)} =$

$$\left|\begin{array}{cccccccccccccc}
0.0 & 20.0 & 4.0 & 4.0 & 0.0 & 4.0 & 6.0 & 10.0 & 18.0 & 12.0 & 40.0 & 24.0 & 26.0 & 6.0 \\
24.0 & 0.0 & 10.0 & 14.0 & 12.0 & 22.0 & 34.0 & 14.0 & 26.0 & 8.0 & 0.0 & 32.0 & 8.0 & 2.0 \\
34.0 & 22.0 & 0.0 & 0.0 & 12.0 & 10.0 & 36.0 & 36.0 & 20.0 & 0.0 & 8.0 & 32.0 & 6.0 & 30.0 \\
38.0 & 4.0 & 20.0 & 0.0 & 28.0 & 8.0 & 2.0 & 10.0 & 14.0 & 8.0 & 26.0 & 30.0 & 14.0 & 8.0 \\
14.0 & 4.0 & 8.0 & 22.0 & 0.0 & 20.0 & 28.0 & 36.0 & 34.0 & 34.0 & 28.0 & 24.0 & 30.0 & 18.0 \\
24.0 & 18.0 & 24.0 & 34.0 & 20.0 & 0.0 & 22.0 & 24.0 & 30.0 & 28.0 & 14.0 & 0.0 & 2.0 & 2.0 \\
36.0 & 20.0 & 32.0 & 0.0 & 14.0 & 34.0 & 0.0 & 20.0 & 36.0 & 34.0 & 26.0 & 36.0 & 20.0 & 12.0 \\
6.0 & 34.0 & 32.0 & 36.0 & 14.0 & 30.0 & 34.0 & 0.0 & 26.0 & 38.0 & 16.0 & 30.0 & 14.0 & 18.0 \\
28.0 & 10.0 & 22.0 & 2.0 & 32.0 & 24.0 & 16.0 & 26.0 & 0.0 & 12.0 & 36.0 & 26.0 & 18.0 & 32.0 \\
20.0 & 26.0 & 14.0 & 4.0 & 20.0 & 26.0 & 22.0 & 12.0 & 34.0 & 0.0 & 6.0 & 28.0 & 10.0 & 30.0 \\
38.0 & 4.0 & 26.0 & 20.0 & 36.0 & 28.0 & 38.0 & 34.0 & 22.0 & 36.0 & 0.0 & 20.0 & 28.0 & 14.0 \\
10.0 & 14.0 & 8.0 & 34.0 & 22.0 & 32.0 & 28.0 & 2.0 & 36.0 & 22.0 & 14.0 & 0.0 & 10.0 & 4.0 \\
14.0 & 36.0 & 34.0 & 4.0 & 8.0 & 2.0 & 32.0 & 26.0 & 14.0 & 20.0 & 8.0 & 34.0 & 0.0 & 12.0 \\
0.0 & 36.0 & 32.0 & 34.0 & 16.0 & 28.0 & 14.0 & 14.0 & 34.0 & 8.0 & 22.0 & 36.0 & 26.0 & 0.0
\end{array}\right|$$

$\Gamma^{(2)} =$

$$\begin{vmatrix}
0.0 & 6.0 & 30.0 & 4.0 & 12.0 & 24.0 & 26.0 & 8.0 & 16.0 & 20.0 & 38.0 & 0.0 & 14.0 & 32.0 \\
30.0 & 0.0 & 36.0 & 6.0 & 22.0 & 28.0 & 26.0 & 18.0 & 30.0 & 14.0 & 16.0 & 18.0 & 26.0 & 12.0 \\
6.0 & 20.0 & 0.0 & 30.0 & 4.0 & 22.0 & 24.0 & 38.0 & 10.0 & 4.0 & 26.0 & 36.0 & 38.0 & 4.0 \\
26.0 & 30.0 & 2.0 & 0.0 & 0.0 & 22.0 & 18.0 & 6.0 & 22.0 & 30.0 & 30.0 & 8.0 & 20.0 & 12.0 \\
18.0 & 2.0 & 16.0 & 12.0 & 0.0 & 24.0 & 4.0 & 18.0 & 6.0 & 12.0 & 36.0 & 14.0 & 6.0 & 6.0 \\
38.0 & 8.0 & 14.0 & 0.0 & 10.0 & 0.0 & 30.0 & 20.0 & 20.0 & 4.0 & 6.0 & 32.0 & 4.0 & 30.0 \\
8.0 & 18.0 & 22.0 & 20.0 & 38.0 & 12.0 & 0.0 & 8.0 & 18.0 & 8.0 & 30.0 & 30.0 & 12.0 & 8.0 \\
6.0 & 0.0 & 22.0 & 10.0 & 6.0 & 20.0 & 10.0 & 0.0 & 28.0 & 28.0 & 12.0 & 10.0 & 2.0 & 36.0 \\
20.0 & 0.0 & 32.0 & 10.0 & 8.0 & 14.0 & 20.0 & 4.0 & 0.0 & 2.0 & 22.0 & 24.0 & 28.0 & 8.0 \\
34.0 & 34.0 & 2.0 & 32.0 & 30.0 & 38.0 & 34.0 & 6.0 & 0.0 & 0.0 & 12.0 & 32.0 & 2.0 & 32.0 \\
26.0 & 30.0 & 14.0 & 16.0 & 34.0 & 18.0 & 30.0 & 12.0 & 20.0 & 2.0 & 0.0 & 36.0 & 8.0 & 14.0 \\
8.0 & 24.0 & 10.0 & 6.0 & 0.0 & 26.0 & 36.0 & 24.0 & 20.0 & 24.0 & 4.0 & 0.0 & 36.0 & 14.0 \\
30.0 & 30.0 & 24.0 & 26.0 & 2.0 & 6.0 & 12.0 & 2.0 & 34.0 & 14.0 & 6.0 & 8.0 & 0.0 & 34.0 \\
36.0 & 6.0 & 30.0 & 36.0 & 4.0 & 20.0 & 26.0 & 2.0 & 4.0 & 16.0 & 30.0 & 26.0 & 22.0 & 0.0
\end{vmatrix}$$

$\Gamma^{(3)} =$

$$\begin{vmatrix}
0.0 & 10.0 & 30.0 & 28.0 & 28.0 & 34.0 & 30.0 & 38.0 & 34.0 & 20.0 & 18.0 & 34.0 & 28.0 & 14.0 \\
8.0 & 0.0 & 12.0 & 26.0 & 20.0 & 26.0 & 30.0 & 10.0 & 14.0 & 6.0 & 16.0 & 0.0 & 14.0 & 6.0 \\
18.0 & 16.0 & 0.0 & 36.0 & 8.0 & 4.0 & 0.0 & 4.0 & 10.0 & 8.0 & 16.0 & 0.0 & 22.0 & 8.0 \\
26.0 & 20.0 & 0.0 & 0.0 & 30.0 & 4.0 & 8.0 & 20.0 & 16.0 & 6.0 & 32.0 & 38.0 & 8.0 & 30.0 \\
12.0 & 14.0 & 2.0 & 36.0 & 0.0 & 12.0 & 24.0 & 24.0 & 20.0 & 32.0 & 26.0 & 22.0 & 36.0 & 36.0 \\
20.0 & 8.0 & 18.0 & 36.0 & 6.0 & 0.0 & 32.0 & 10.0 & 28.0 & 34.0 & 18.0 & 32.0 & 4.0 & 20.0 \\
2.0 & 12.0 & 6.0 & 16.0 & 36.0 & 2.0 & 0.0 & 26.0 & 30.0 & 18.0 & 24.0 & 12.0 & 36.0 & 6.0 \\
10.0 & 30.0 & 34.0 & 24.0 & 28.0 & 22.0 & 38.0 & 0.0 & 12.0 & 24.0 & 38.0 & 16.0 & 32.0 & 26.0 \\
36.0 & 6.0 & 32.0 & 34.0 & 22.0 & 10.0 & 16.0 & 18.0 & 0.0 & 16.0 & 36.0 & 4.0 & 22.0 & 38.0 \\
4.0 & 30.0 & 26.0 & 36.0 & 38.0 & 24.0 & 22.0 & 8.0 & 28.0 & 0.0 & 38.0 & 36.0 & 22.0 & 24.0 \\
32.0 & 28.0 & 16.0 & 6.0 & 30.0 & 28.0 & 26.0 & 14.0 & 28.0 & 4.0 & 0.0 & 38.0 & 14.0 & 0.0 \\
12.0 & 8.0 & 8.0 & 8.0 & 18.0 & 22.0 & 0.0 & 20.0 & 0.0 & 32.0 & 6.0 & 0.0 & 22.0 & 14.0 \\
38.0 & 24.0 & 32.0 & 8.0 & 26.0 & 4.0 & 16.0 & 22.0 & 18.0 & 20.0 & 30.0 & 10.0 & 0.0 & 24.0 \\
14.0 & 6.0 & 36.0 & 30.0 & 10.0 & 26.0 & 16.0 & 10.0 & 4.0 & 36.0 & 8.0 & 28.0 & 0.0 & 0.0
\end{vmatrix}$$

APPENDIX E
COMPARATIVE RESULTS FOR INDIVIDUAL LINKS

Table E.1. Comparative Results for Links— Network 1, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.87978	0.8402	0.8295	0.8509	0.14433	0.1439	0.1426	0.1452	no	yes	4.7108	-
2	0.86512	0.8627	0.8546	0.8709	0.06665	0.0698	0.0693	0.0704	yes	no	-	4.5129
3	0.87978	0.8781	0.8658	0.8905	0.14433	0.1504	0.1484	0.1523	yes	no	-	4.0359
4	0.86152	0.8439	0.8348	0.8530	0.06665	0.0683	0.0677	0.0690	no	no	2.0879	2.4158

Table E.2. Comparative Results for Links— Network 1, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.29858	0.3007	0.2952	0.3062	0.09971	0.1040	0.1015	0.1065	yes	no	-	4.1250
2	0.29441	0.3052	0.2974	0.3131	0.04634	0.0545	0.0498	0.0592	no	no	3.5354	14.9725
3	0.29858	0.3048	0.2998	0.3097	0.09971	0.1055	0.1032	0.1078	no	no	2.0407	5.4882
4	0.29441	0.3048	0.2972	0.3123	0.04634	0.0542	0.0497	0.0587	no	no	3.4088	14.5019

Table E.3. Comparative Results for Links— Network 1, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	9.94226	9.2055	8.6247	9.7863	0.84017	0.7753	0.7302	0.8205	no	no	8.0035	8.3671
2	10.10408	10.2388	9.6231	10.8545	0.39756	0.4057	0.3831	0.4283	yes	yes	-	-
3	9.94226	10.0385	9.3860	10.6910	0.84017	0.8456	0.7942	0.8971	yes	yes	-	-
4	10.10408	9.2423	8.6277	9.8569	0.39756	0.3663	0.3437	0.3889	no	no	9.3243	8.5340

Table E.4. Comparative Results for Links— Network 2, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	% m.g. Q_i	% m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i$ ($\overline{W}_{Y,i}$)	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.85691	0.8334	0.8253	0.8415	0.03097	0.0327	0.0323	0.0330	no	no	2.8201	5.2905
2	0.82611	0.8336	0.8215	0.8458	0.05692	0.0594	0.0587	0.0601	yes	no	-	4.1751
3	0.85065	0.8486	0.8353	0.8620	0.05975	0.0628	0.0621	0.0636	yes	no	-	4.8567
4	0.82715	0.8358	0.8177	0.8540	0.10069	0.1040	0.1022	0.1059	yes	no	-	3.1827
5	0.83844	0.8338	0.8220	0.8457	0.06342	0.0659	0.0652	0.0666	yes	no	-	3.7633
6	0.89028	0.8838	0.8652	0.9023	0.13561	0.1426	0.1402	0.1450	yes	no	-	4.9018

Table E.5. Comparative Results for Links— Network 2, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	% m.g. Q_i	% m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i$ ($\overline{W}_{Y,i}$)	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.29293	0.2895	0.2875	0.2915	0.02156	0.0234	0.0232	0.0236	no	no	1.1848	7.8633
2	0.28914	0.2910	0.2871	0.2948	0.04018	0.0420	0.0416	0.0424	yes	no	-	4.3333
3	0.29261	0.2910	0.2883	0.2936	0.04173	0.0438	0.0434	0.0442	yes	no	-	4.7260
4	0.28982	0.2884	0.2848	0.2920	0.07108	0.0723	0.0716	0.0730	yes	no	-	1.6874
5	0.29121	0.2910	0.2889	0.2931	0.04454	0.0467	0.0464	0.0470	yes	no	-	4.6523
6	0.29922	0.2985	0.2938	0.3033	0.09321	0.0967	0.0956	0.0978	yes	no	-	3.6091

Table E.6. Comparative Results for Links— Network 2, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	% m.g. Q_i	% m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i$ ($\overline{W}_{Y,i}$)	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	10.69382	8.9213	8.1365	9.7070	0.19508	0.1676	0.1536	0.1817	no	no	19.8684	16.3664
2	8.90853	8.8774	8.2389	9.5159	0.30867	0.3092	0.2877	0.3307	yes	yes	-	-
3	9.82003	9.3622	8.6791	10.0454	0.34933	0.3399	0.3158	0.3641	yes	yes	-	-
4	8.71106	8.8006	8.2272	9.3740	0.53516	0.5413	0.5065	0.5760	yes	yes	-	-
5	9.32204	9.2355	8.4577	10.0134	0.35608	0.3571	0.3288	0.3854	yes	yes	-	-
6	10.37458	10.8445	9.0526	12.6365	0.81604	0.8642	0.7267	1.0018	yes	yes	-	-

Table E.7. Comparative Results for Links— Network 3, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.80744	0.8034	0.7962	0.8106	0.02413	0.0254	0.0250	0.0258	yes	no	-	5.0000
2	0.79680	0.8103	0.7996	0.8209	0.03478	0.0366	0.0362	0.0371	no	no	1.6661	5.1093
3	0.81782	0.8182	0.8065	0.8300	0.06732	0.0689	0.0679	0.0699	yes	no	-	2.2932
4	0.83036	0.8299	0.8217	0.8381	0.10950	0.1123	0.1113	0.1133	yes	no	-	2.4933
5	0.80497	0.7854	0.7719	0.7989	0.07745	0.0769	0.0757	0.0782	no	yes	2.4917	-
6	1.09095	1.0904	1.0522	1.1287	0.23651	0.2744	0.2670	0.2818	yes	no	-	13.8083
7	0.80600	0.7970	0.7826	0.8114	0.07238	0.0737	0.0727	0.0747	yes	no	-	1.7910
8	0.82604	0.8231	0.8087	0.8374	0.07101	0.0727	0.0717	0.0737	yes	no	-	2.3246

Table E.8. Comparative Results for Links— Network 3, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.28651	0.2868	0.2846	0.2889	0.01718	0.0188	0.0185	0.0190	yes	no	-	8.6170
2	0.28357	0.2875	0.2856	0.2893	0.02481	0.0267	0.0264	0.0269	no	no	1.3670	7.0787
3	0.28316	0.2847	0.2817	0.2877	0.04771	0.0488	0.0484	0.0493	no	no	1.2153	2.2336
4	0.28997	0.2879	0.2841	0.2917	0.07717	0.0789	0.0781	0.0797	yes	no	-	2.1927
5	0.28639	0.2876	0.2837	0.2915	0.05520	0.0566	0.0561	0.0570	yes	no	-	2.4735
6	0.32981	0.3360	0.3292	0.3429	0.15042	0.1671	0.1645	0.1698	yes	no	-	9.9821
7	0.28390	0.2841	0.2810	0.2872	0.05145	0.0530	0.0524	0.0535	yes	no	-	2.9245
8	0.28957	0.2890	0.2859	0.2920	0.05015	0.0518	0.0513	0.0522	yes	no	-	3.1853

Table E.9. Comparative Results for Links— Network 3, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.44210	8.0282	7.4636	8.5927	0.12606	0.1207	0.1125	0.1288	yes	yes	-	-
2	7.82406	7.8937	7.3349	8.4524	0.17068	0.1735	0.1619	0.1851	yes	yes	-	-
3	8.56319	9.4526	8.2243	10.6808	0.35416	0.3911	0.3424	0.4397	yes	yes	-	-
4	8.98270	8.7692	7.5836	9.9548	0.59721	0.5913	0.5156	0.6669	yes	yes	-	-
5	8.16555	8.2123	7.5987	8.8259	0.39327	0.3953	0.3666	0.4240	yes	yes	-	-
6	15.40817	15.1753	12.7842	17.5665	1.85537	1.8926	1.6089	2.1763	yes	yes	-	-
7	7.93466	8.0757	7.4809	8.6704	0.35807	0.3674	0.3406	0.3941	yes	yes	-	-
8	8.72178	7.8410	7.2542	8.4278	0.37804	0.3424	0.3184	0.3665	no	no	11.2324	10.4809

Table E.10. Comparative Results for Links— Network 4, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.81500	0.8065	0.7931	0.8199	0.01839	0.0192	0.0188	0.0197	yes	no	-	4.2188
2	0.81900	0.8184	0.8098	0.8270	0.01362	0.0147	0.0143	0.0151	yes	no	-	7.3469
3	0.80586	0.8063	0.7906	0.8221	0.06585	0.0668	0.0657	0.0679	yes	yes	-	-
4	0.84246	0.8466	0.8237	0.8696	0.06423	0.0668	0.0652	0.0683	yes	no	-	3.8473
5	0.99950	1.0252	0.9868	1.0635	0.11948	0.1372	0.1331	0.1412	yes	no	-	12.9155
6	0.83154	0.8185	0.7968	0.8383	0.06080	0.0616	0.0605	0.0628	yes	no	-	2.1104
7	0.83590	0.8374	0.8176	0.8573	0.06044	0.0626	0.0614	0.0637	yes	no	-	3.4505
8	0.81426	0.8111	0.7959	0.8263	0.04962	0.0506	0.0498	0.0514	yes	no	-	1.9368
9	0.87800	0.8952	0.8711	0.9193	0.07764	0.0836	0.0818	0.0853	yes	no	-	7.1292
10	0.81410	0.8269	0.8113	0.8426	0.04778	0.0494	0.0486	0.0502	yes	no	-	3.2794

Table E.11. Comparative Results for Links— Network 4, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.28744	0.2872	0.2849	0.2896	0.01304	0.0144	0.0141	0.0147	yes	no	-	9.4444
2	0.28756	0.2874	0.2853	0.2895	0.00964	0.0111	0.0108	0.0115	yes	no	-	13.1532
3	0.28646	0.2884	0.2838	0.2930	0.04691	0.0482	0.0475	0.0489	yes	no	-	2.6764
4	0.29137	0.2930	0.2879	0.2981	0.04501	0.0470	0.0463	0.0476	yes	no	-	4.2340
5	0.31556	0.3231	0.3167	0.3295	0.07861	0.0868	0.0854	0.0881	no	no	2.3336	9.4355
6	0.28991	0.2902	0.2864	0.2941	0.04247	0.0442	0.0438	0.0447	yes	no	-	3.9140
7	0.29069	0.2914	0.2879	0.2950	0.04249	0.0442	0.0438	0.0446	yes	no	-	3.9088
8	0.28768	0.2848	0.2814	0.2883	0.03522	0.0364	0.0359	0.0368	yes	no	-	3.2418
9	0.29669	0.2977	0.2920	0.3034	0.05631	0.0563	0.0555	0.0571	yes	yes	-	-
10	0.28746	0.2843	0.2806	0.2879	0.03391	0.0350	0.0347	0.0354	yes	no	-	3.1143

Table E.12. Comparative Results for Links— Network 4, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.76763	8.3932	7.8763	8.9100	0.09894	0.0955	0.0898	0.1012	yes	yes	-	-
2	9.65676	9.4654	8.8662	10.0646	0.07977	0.0799	0.0749	0.0848	yes	yes	-	-
3	8.22841	9.6069	7.9181	11.2956	0.33653	0.3914	0.3251	0.4577	yes	yes	-	-
4	9.63860	9.7102	8.5090	10.9115	0.37085	0.3774	0.3330	0.4219	yes	yes	-	-
5	13.30558	14.3754	12.6870	16.0637	0.85496	0.9582	0.8512	1.0653	yes	yes	-	-
6	9.14658	8.2984	7.4196	9.1773	0.33404	0.3084	0.2769	0.3399	yes	yes	-	-
7	9.17611	9.1674	8.2048	10.1300	0.33502	0.3404	0.3059	0.3748	yes	yes	-	-
8	8.49245	8.3616	7.3526	9.3705	0.25962	0.2585	0.2280	0.2890	yes	yes	-	-
9	10.62127	10.0644	8.9453	11.1835	0.48047	0.4634	0.4141	0.5126	yes	yes	-	-
10	8.62969	8.2327	7.1102	9.3552	0.25362	0.2433	0.2113	0.2753	yes	yes	-	-

Table E.13. Comparative Results for Links— Network 5, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.80726	0.8095	0.7982	0.8207	0.01793	0.0189	0.0185	0.0193	yes	no	-	5.1323
2	0.82917	0.8506	0.8406	0.8606	0.01436	0.0157	0.0153	0.0161	no	no	2.5194	8.5350
3	0.81010	0.8217	0.8057	0.8376	0.05612	0.0578	0.0569	0.0587	yes	no	-	2.9066
4	0.85846	0.8565	0.8384	0.8747	0.06058	0.0633	0.0622	0.0644	yes	no	-	4.2970
5	0.81922	0.8143	0.7821	0.8465	0.16909	0.1703	0.1642	0.1763	yes	yes	-	-
6	0.94425	0.9427	0.9180	0.9673	0.08432	0.0920	0.0898	0.0941	yes	no	-	8.3478
7	0.96670	0.9845	0.9609	1.0081	0.07743	0.0870	0.0853	0.0886	yes	no	-	11.0000
8	1.02222	1.0206	0.9802	1.0609	0.08853	0.1009	0.0975	0.1043	yes	no	-	12.2597
9	0.93234	0.9199	0.8957	0.9441	0.08498	0.0912	0.0892	0.0932	yes	no	-	6.8202
10	1.15366	1.3746	1.3319	1.4173	0.10843	0.1436	0.1397	0.1476	no	no	16.0730	24.4916
11	0.84206	0.8517	0.8120	0.8913	0.19087	0.1987	0.1914	0.2061	yes	no	-	3.9406
12	0.80168	0.8041	0.7911	0.8171	0.05823	0.0588	0.0580	0.0595	yes	yes	-	-

Table E.14. Comparative Results for Links— Network 5, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.28651	0.2866	0.2844	0.2888	0.01276	0.0142	0.0138	0.0145	yes	no	-	10.1409
2	0.28871	0.2904	0.2883	0.2925	0.01012	0.0117	0.0113	0.0120	yes	no	-	13.5043
3	0.28716	0.2888	0.2850	0.2926	0.03991	0.0412	0.0409	0.0416	yes	no	-	3.1311
4	0.29412	0.2922	0.2889	0.2955	0.04219	0.0441	0.0436	0.0446	yes	no	-	4.3311
5	0.28860	0.2908	0.2844	0.2973	0.11977	0.1217	0.1197	0.1237	yes	yes	-	-
6	0.30556	0.3074	0.3018	0.3131	0.05662	0.0610	0.0600	0.0619	yes	no	-	7.1808
7	0.30793	0.3033	0.2988	0.3079	0.05149	0.0550	0.0544	0.0555	no	no	1.5265	6.3818
8	0.31580	0.3195	0.3139	0.3252	0.05758	0.0630	0.0623	0.0638	yes	no	-	8.6032
9	0.30326	0.3047	0.2986	0.3108	0.05731	0.0606	0.0598	0.0615	yes	no	-	5.4290
10	0.33297	0.3293	0.3219	0.3368	0.06711	0.0750	0.0738	0.0761	yes	no	-	10.5200
11	0.29211	0.2893	0.2832	0.2953	0.13388	0.1356	0.1335	0.1376	yes	no	-	1.2684
12	0.28593	0.2848	0.2814	0.2881	0.04156	0.0425	0.0421	0.0429	yes	no	-	2.2118

Table E.15. Comparative Results for Links— Network 5, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.47516	8.4088	7.8644	8.9432	0.09400	0.0938	0.0880	0.0995	yes	yes	-	-
2	9.98245	10.1003	9.3841	10.8164	0.08622	0.0892	0.0830	0.0953	yes	yes	-	-
3	8.30875	7.8598	7.1110	8.6087	0.28845	0.2735	0.2488	0.2982	yes	yes	-	-
4	9.89749	9.8879	8.7933	10.9826	0.35523	0.3604	0.3223	0.3985	yes	yes	-	-
5	8.53283	7.7053	6.6401	8.7705	0.88598	0.8041	0.6995	0.9087	yes	yes	-	-
6	12.74286	13.2197	11.0966	15.3429	0.59596	0.6396	0.5405	0.7386	yes	yes	-	-
7	13.75858	14.9327	13.0135	16.8519	0.58084	0.6553	0.5738	0.7368	yes	yes	-	-
8	15.49807	15.6250	12.9431	18.3069	0.72223	0.7658	0.6375	0.8941	yes	yes	-	-
9	12.44970	14.8598	11.7876	17.9320	0.59145	0.7297	0.5821	0.8773	yes	yes	-	-
10	19.30082	17.5975	14.5857	20.6092	1.02206	0.9934	0.8321	1.1548	yes	yes	-	-
11	9.13726	9.0583	7.9057	10.2109	1.05089	1.0625	0.9252	1.1998	yes	yes	-	-
12	8.05837	8.1819	7.2940	9.0698	0.29277	0.2977	0.2667	0.3287	yes	yes	-	-

Table E.16. Comparative Results for Links— Network 6, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.88700	0.8397	0.8285	0.8509	0.08656	0.0868	0.0859	0.0878	no	yes	5.6330	-
2	0.80758	0.8058	0.7987	0.8130	0.05020	0.0513	0.0509	0.0516	yes	no	-	2.1443
3	0.88700	0.8456	0.8362	0.8568	0.08656	0.0875	0.0867	0.0883	no	no	4.8959	1.0743
4	0.80758	0.8047	0.7978	0.8116	0.05020	0.0512	0.0508	0.0516	yes	no	-	1.9531
5	0.88700	0.8873	0.8766	0.8980	0.08656	0.0917	0.0908	0.0925	yes	no	-	5.6052
6	0.80758	0.7968	0.7895	0.8042	0.05020	0.0507	0.0503	0.0511	no	no	1.3529	0.9862

Table E.17. Comparative Results for Links— Network 6, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.29909	0.2937	0.2913	0.2962	0.05959	0.0611	0.0608	0.0615	no	no	1.8352	2.4714
2	0.28672	0.2874	0.2854	0.2893	0.03574	0.0371	0.0369	0.0373	yes	no	-	3.6658
3	0.29909	0.2967	0.2940	0.2993	0.05959	0.0617	0.0613	0.0622	yes	no	-	3.4198
4	0.28672	0.2864	0.2842	0.2887	0.03574	0.0370	0.0367	0.0372	yes	no	-	3.4054
5	0.29909	0.3011	0.2976	0.3047	0.05959	0.0626	0.0621	0.0631	yes	no	-	4.8083
6	0.28672	0.2870	0.2850	0.2890	0.03574	0.0370	0.0368	0.0372	yes	no	-	3.4054

Table E.18. Comparative Results for Links— Network 6, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	10.24317	8.4968	7.8986	9.0949	0.51576	0.4344	0.4049	0.4639	no	no	20.5533	18.7293
2	8.27139	8.1442	7.7429	8.5455	0.25749	0.2521	0.2403	0.2640	yes	yes	-	-
3	10.24317	9.0434	8.4350	9.6519	0.51576	0.4623	0.4321	0.4926	no	no	13.2668	11.5639
4	8.27139	8.1236	7.7149	8.5322	0.25749	0.2515	0.2393	0.2637	yes	yes	-	-
5	10.24317	10.3526	9.5015	11.2034	0.51576	0.5290	0.4869	0.5712	yes	yes	-	-
6	8.27139	7.8248	7.5285	8.1211	0.25749	0.2424	0.2336	0.2512	no	yes	5.7034	-

Table E.19. Comparative Results for Links— Network 7, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	% m.g. Q_i	% m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.80524	0.7728	0.7671	0.7784	0.05208	0.0535	0.0532	0.0538	no	no	4.1977	2.6542
2	0.89957	0.9125	0.8961	0.9289	0.19651	0.2143	0.2112	0.2174	yes	no	-	8.3015
3	0.80524	0.7801	0.7719	0.7882	0.05208	0.0540	0.0535	0.0544	no	no	3.2227	3.5556
4	0.89957	0.8438	0.8308	0.8569	0.19651	0.1984	0.1960	0.2008	no	yes	6.6094	-
5	0.80524	0.7873	0.7789	0.7956	0.05208	0.0544	0.0539	0.0549	no	no	2.2787	4.2647
6	0.89957	0.8298	0.8125	0.8472	0.19651	0.1951	0.1916	0.1986	no	yes	8.4081	-
7	0.80524	0.8072	0.8010	0.8134	0.05208	0.0558	0.0554	0.0561	yes	no	-	6.6667
8	0.89957	0.8208	0.8084	0.8332	0.19651	0.1930	0.1906	0.1953	no	no	9.5967	1.8187

Table E.20. Comparative Results for Links— Network 7, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	% m.g. Q_i	% m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.28016	0.2789	0.2752	0.2827	0.03693	0.0444	0.0397	0.0492	yes	no	-	16.8243
2	0.29959	0.3048	0.3001	0.3095	0.13444	0.1443	0.1416	0.1470	no	no	1.7093	6.8330
3	0.28016	0.2809	0.2773	0.2844	0.03693	0.0447	0.0399	0.0495	yes	no	-	17.3826
4	0.29956	0.2957	0.2912	0.3001	0.13444	0.1402	0.1376	0.1427	yes	no	-	4.1084
5	0.28016	0.2820	0.2786	0.2854	0.03693	0.0449	0.0401	0.0496	yes	no	-	17.7506
6	0.29959	0.2956	0.2912	0.2999	0.13444	0.1401	0.1373	0.1428	yes	no	-	4.0400
7	0.28016	0.2826	0.2789	0.2862	0.03693	0.0450	0.0402	0.0499	yes	no	-	17.9833
8	0.29959	0.2934	0.2884	0.2984	0.13444	0.1391	0.1364	0.1418	no	no	2.1098	3.3501

Table E.21. Comparative Results for Links— Network 7, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	% m.g. Q_i	% m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	7.46566	6.0188	5.7701	6.2674	0.24425	0.2031	0.1954	0.2107	no	no	24.0390	20.2610
2	9.59461	10.0989	8.7636	11.4343	1.09602	1.1776	1.0282	1.3270	yes	no	-	6.9277
3	7.46566	6.1804	5.8781	6.4826	0.24425	0.2085	0.1989	0.2182	no	no	20.7957	17.1763
4	9.59461	7.7474	7.0070	8.4877	1.09602	0.9039	0.8226	0.9852	no	no	23.8430	21.2546
5	7.46566	6.5408	6.2287	6.8519	0.24425	0.2206	0.2106	0.2306	no	no	14.1486	10.7208
6	9.59461	7.8878	7.2289	8.5468	1.09602	0.9211	0.8474	0.9948	no	no	21.6386	18.9008
7	7.46566	7.4073	7.0521	7.7626	0.24425	0.2496	0.2382	0.2610	yes	yes	-	-
8	9.59461	7.5692	6.7670	8.3715	1.09602	0.8832	0.7939	0.9725	no	no	26.7586	24.0965

Table E.22. Comparative Results for Links— Network 8, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.80236	0.8164	0.7954	0.8373	0.01780	0.0182	0.0178	0.0186	yes	yes	-	-
2	0.81508	0.8210	0.8006	0.8414	0.01601	0.0164	0.0161	0.0167	yes	no	-	2.3781
3	0.81532	0.8208	0.8027	0.8380	0.00810	0.0083	0.0082	0.0085	yes	no	-	2.4096
4	0.80091	0.7997	0.7876	0.8118	0.00846	0.0086	0.0085	0.0087	yes	yes	-	-
5	0.80007	0.7899	0.7741	0.8056	0.01527	0.0152	0.0149	0.0154	yes	yes	-	-
6	0.84297	0.8389	0.8140	0.8638	0.01773	0.0182	0.0178	0.0187	yes	no	-	2.5824
7	0.80066	0.8005	0.7860	0.8150	0.01105	0.0111	0.0110	0.0113	yes	yes	-	-
8	0.83163	0.8305	0.8133	0.8478	0.01280	0.0132	0.0129	0.0134	yes	no	-	3.0303
9	0.80586	0.8147	0.7973	0.8320	0.02017	0.0206	0.0202	0.0209	yes	yes	-	-
10	0.81147	0.8138	0.7957	0.8318	0.01964	0.0200	0.0197	0.0204	yes	no	-	1.8000
11	0.80011	0.8081	0.7874	0.8187	0.01932	0.0195	0.0191	0.0198	yes	yes	-	-
12	0.80701	0.8043	0.7821	0.8265	0.02243	0.0225	0.0219	0.0231	yes	yes	-	-

Table E.23. Comparative Results for Links— Network 8, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.28597	0.2839	0.2809	0.2868	0.01270	0.0129	0.0127	0.0131	yes	yes	-	-
2	0.28735	0.2878	0.2842	0.2913	0.01185	0.0116	0.0114	0.0118	yes	yes	-	-
3	0.28727	0.2869	0.2826	0.2911	0.00574	0.0060	0.0058	0.0061	yes	no	-	4.3333
4	0.28581	0.2843	0.2810	0.2876	0.00604	0.0063	0.0061	0.0064	yes	yes	-	-
5	0.28572	0.2843	0.2806	0.2881	0.01091	0.0111	0.0109	0.0113	yes	yes	-	-
6	0.29043	0.2921	0.2861	0.2981	0.01241	0.0129	0.0126	0.0131	yes	no	-	3.7985
7	0.28578	0.2847	0.2816	0.2879	0.00789	0.0081	0.0079	0.0083	yes	yes	-	-
8	0.28898	0.2880	0.2842	0.2918	0.00901	0.0093	0.0091	0.0095	yes	no	-	3.1183
9	0.28636	0.2834	0.2791	0.2877	0.01436	0.0145	0.0143	0.0147	yes	yes	-	-
10	0.28696	0.2840	0.2794	0.2886	0.01395	0.0141	0.0139	0.0143	yes	yes	-	-
11	0.28573	0.2889	0.2851	0.2927	0.01380	0.0142	0.0140	0.0143	yes	no	-	2.8169
12	0.28650	0.2868	0.2828	0.2909	0.01597	0.0163	0.0161	0.0165	yes	no	-	2.0245

Table E.24. Comparative Results for Links— Network 8, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.15586	8.0065	7.1611	8.8518	0.09043	0.0892	0.0800	0.0984	yes	yes	-	-
2	9.06911	9.2640	8.1170	10.4111	0.08879	0.0922	0.0811	0.1034	yes	yes	-	-
3	9.61069	8.8670	8.2532	9.4809	0.04735	0.0446	0.0416	0.0476	no	yes	8.3872	-
4	8.09099	7.6574	7.1417	8.1732	0.04271	0.0405	0.0380	0.0430	yes	yes	-	-
5	8.00524	8.2198	7.1300	9.3097	0.07638	0.0781	0.0683	0.0878	yes	yes	-	-
6	10.96463	10.5346	9.2593	11.8100	0.11510	0.1140	0.1010	0.1269	yes	yes	-	-
7	8.05643	8.7431	7.8565	9.6297	0.05560	0.0604	0.0543	0.0664	yes	yes	-	-
8	10.56637	9.9599	8.9146	11.0052	0.08083	0.0784	0.0704	0.0864	yes	yes	-	-
9	8.36057	9.1586	7.9910	10.3262	0.10453	0.1151	0.1008	0.1294	yes	yes	-	-
10	8.71711	11.0569	6.1640	15.9498	0.10533	0.1332	0.0774	0.1890	yes	yes	-	-
11	8.00693	7.9893	7.0961	8.8824	0.09669	0.0964	0.0859	0.1069	yes	yes	-	-
12	8.40736	8.0959	6.9800	9.2119	0.11630	0.1131	0.0979	0.1284	yes	yes	-	-

Table E.25. Comparative Results for Links— Network 8, Parameter Set 4

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.61102	0.6111	0.6001	0.6221	0.01591	0.0168	0.0166	0.0170	yes	no	-	5.2976
2	0.65696	0.6507	0.6372	0.6642	0.01461	0.0156	0.0154	0.0159	yes	no	-	6.3462
3	0.65998	0.6528	0.6429	0.6628	0.00741	0.0080	0.0079	0.0081	yes	no	-	7.3750
4	0.60466	0.5994	0.5901	0.6088	0.00754	0.0080	0.0079	0.0081	yes	no	-	5.7500
5	0.58135	0.5809	0.5692	0.5926	0.01341	0.0140	0.0138	0.0142	yes	no	-	4.2143
6	0.71671	0.7048	0.6866	0.7230	0.01651	0.0180	0.0176	0.0183	yes	no	-	8.2778
7	0.60101	0.6024	0.5915	0.6132	0.00983	0.0104	0.0103	0.0106	yes	no	-	5.4808
8	0.51205	0.5045	0.4992	0.5099	0.01057	0.0106	0.0105	0.0107	no	yes	1.4965	-
9	0.54711	0.5359	0.5277	0.5440	0.01728	0.0174	0.0172	0.0177	no	yes	2.0918	-
10	0.53443	0.5309	0.5189	0.5429	0.01663	0.0170	0.0167	0.0173	yes	no	-	2.1765
11	0.57367	0.5740	0.5623	0.5856	0.01695	0.0175	0.0172	0.0178	yes	no	-	3.1429
12	0.54293	0.5367	0.5246	0.5489	0.01915	0.0194	0.0191	0.0198	yes	yes	-	-

Table E.26. Comparative Results for Links— Network 8, Parameter Set 5

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.22664	0.2271	0.2245	0.2297	0.01212	0.0130	0.0127	0.0133	yes	no	-	6.7692
2	0.23622	0.2337	0.2300	0.2373	0.01090	0.0117	0.0113	0.0120	yes	no	-	6.8376
3	0.23582	0.2357	0.2331	0.2383	0.00651	0.0063	0.0059	0.0066	yes	no	-	12.5397
4	0.22443	0.2237	0.2211	0.2263	0.00575	0.0064	0.0061	0.0068	yes	no	-	10.1563
5	0.21979	0.2183	0.2154	0.2211	0.01035	0.0110	0.0107	0.0113	yes	no	-	5.9091
6	0.24820	0.2506	0.2468	0.2544	0.01201	0.0132	0.0128	0.0135	yes	no	-	9.0152
7	0.22391	0.2240	0.2216	0.2263	0.00751	0.0082	0.0079	0.0086	yes	no	-	8.4146
8	0.20253	0.2020	0.2001	0.2039	0.00840	0.0090	0.0086	0.0093	yes	no	-	6.6667
9	0.21179	0.2116	0.2087	0.2145	0.01353	0.0142	0.0138	0.0145	yes	no	-	4.7183
10	0.20856	0.2100	0.2069	0.2131	0.01310	0.0138	0.0135	0.0141	yes	no	-	5.0725
11	0.21939	0.2182	0.2148	0.2216	0.01309	0.0138	0.0135	0.0142	yes	no	-	5.1449
12	0.21081	0.2125	0.2091	0.2158	0.01508	0.0158	0.0154	0.0161	yes	no	-	4.8734

Table E.27. Comparative Results for Links— Network 8, Parameter Set 6

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	3.69412	3.7256	3.4714	3.9798	0.04636	0.0510	0.0481	0.0540	yes	no	-	9.0984
2	4.30312	4.2777	3.9574	4.5981	0.05117	0.0508	0.0474	0.0543	yes	yes	-	-
3	5.68350	4.9826	4.6437	5.3216	0.02982	0.0300	0.0283	0.0317	no	yes	14.0670	-
4	3.94501	3.6279	3.4318	3.8240	0.02323	0.0238	0.0227	0.0250	no	yes	8.7409	-
5	3.19413	3.1726	2.9966	3.3486	0.03557	0.0379	0.0361	0.0397	yes	no	-	6.1478
6	6.35917	5.4157	4.9858	5.8456	0.07080	0.0683	0.0636	0.0730	no	yes	17.4210	-
7	3.71610	3.5754	3.3660	3.7849	0.02895	0.0306	0.0290	0.0322	yes	no	-	5.3922
8	2.18775	2.1667	2.0920	2.2414	0.02228	0.0227	0.0220	0.0234	yes	yes	-	-
9	2.59711	2.5843	2.4666	2.7020	0.04017	0.0417	0.0400	0.0434	yes	yes	-	-
10	2.43265	2.2859	2.1723	2.3996	0.03721	0.0362	0.0346	0.0378	no	yes	6.4198	-
11	3.07964	2.9424	2.7762	3.1086	0.04380	0.0446	0.0424	0.0468	yes	yes	-	-
12	2.52760	2.3682	2.2427	2.4937	0.04379	0.0430	0.0410	0.0450	no	yes	6.7309	-

Table E.28. Comparative Results for Links— Network 9, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.84116	0.8325	0.8141	0.8510	0.01312	0.0134	0.0132	0.0137	yes	no	-	2.0896
2	0.82977	0.8267	0.8044	0.8491	0.01270	0.0130	0.0127	0.0133	yes	yes	-	-
3	0.90413	0.8856	0.8560	0.9151	0.01759	0.0185	0.0180	0.0190	yes	no	-	4.9189
4	0.89129	0.8591	0.8276	0.8905	0.01766	0.0183	0.0177	0.0188	no	yes	3.7469	-
5	0.81401	0.8097	0.7984	0.8211	0.00802	0.0082	0.0081	0.0083	yes	no	-	2.1951
6	0.81758	0.8122	0.7968	0.8275	0.00865	0.0088	0.0087	0.0090	yes	no	-	1.7046
7	0.82855	0.8339	0.8141	0.8536	0.01159	0.0119	0.0117	0.0122	yes	no	-	2.6050
8	0.81124	0.8042	0.7839	0.8246	0.01159	0.0117	0.0115	0.0120	yes	yes	-	-
9	0.85597	0.8544	0.8274	0.8814	0.02228	0.0231	0.0225	0.0237	yes	no	-	3.7662
10	0.85314	0.8447	0.8139	0.8755	0.02469	0.0256	0.0248	0.0264	yes	no	-	3.5547
11	0.81884	0.8104	0.7920	0.8288	0.01435	0.0144	0.0141	0.0148	yes	yes	-	-
12	0.82222	0.8153	0.7985	0.8321	0.01478	0.0150	0.0148	0.0153	yes	yes	-	-
13	0.82208	0.8052	0.7899	0.8206	0.01310	0.0132	0.0130	0.0134	no	yes	2.0964	-
14	0.81762	0.8096	0.7931	0.8260	0.01520	0.0153	0.0150	0.0156	yes	yes	-	-
15	0.81660	0.8277	0.8082	0.8472	0.01443	0.0149	0.0146	0.0152	yes	no	-	3.1544
16	0.85561	0.8543	0.8290	0.8797	0.01604	0.0167	0.0163	0.0171	yes	no	-	3.9521
17	0.84334	0.8419	0.8185	0.8654	0.01856	0.0191	0.0186	0.0195	yes	no	-	2.8272
18	0.81105	0.8273	0.7974	0.8573	0.01994	0.0206	0.0200	0.0212	yes	no	-	3.2039

Table E.29. Comparative Results for Links— Network 9, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\,(\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.81970	0.8384	0.8127	0.8640	0.01297	0.0134	0.0131	0.0137	yes	no	-	3.2090
2	0.81434	0.8005	0.7816	0.8193	0.01260	0.0126	0.0123	0.0128	yes	yes	-	-
3	0.84310	0.8339	0.8106	0.8572	0.01708	0.0173	0.0169	0.0177	yes	yes	-	-
4	0.83782	0.8550	0.8260	0.8839	0.01716	0.0178	0.0173	0.0183	yes	no	-	3.5955
5	0.80797	0.8161	0.8005	0.8317	0.00800	0.0082	0.0080	0.0083	yes	yes	-	-
6	0.80987	0.8070	0.7919	0.8220	0.00861	0.0087	0.0086	0.0089	yes	yes	-	-
7	0.81400	0.8051	0.7909	0.8193	0.01149	0.0115	0.0113	0.0117	yes	yes	-	-
8	0.80628	0.8028	0.7866	0.8189	0.01155	0.0116	0.0114	0.0118	yes	yes	-	-
9	0.82227	0.8194	0.7919	0.8469	0.02182	0.0220	0.0214	0.0226	yes	yes	-	-
10	0.82040	0.8186	0.7902	0.8469	0.02426	0.0244	0.0238	0.0251	yes	yes	-	-
11	0.80887	0.8193	0.7980	0.8407	0.01427	0.0146	0.0143	0.0149	yes	no	-	2.2603
12	0.81032	0.8234	0.8040	0.8428	0.01469	0.0151	0.0148	0.0154	yes	no	-	2.7152
13	0.81070	0.8108	0.7960	0.8257	0.01302	0.0132	0.0130	0.0134	yes	yes	-	-
14	0.80825	0.8073	0.7899	0.8246	0.01513	0.0153	0.0150	0.0156	yes	yes	-	-
15	0.80776	0.8099	0.7856	0.8341	0.01436	0.0145	0.0141	0.0148	yes	yes	-	-
16	0.82514	0.8069	0.7854	0.8284	0.01578	0.0158	0.0154	0.0161	yes	yes	-	-
17	0.81760	0.8106	0.7920	0.8291	0.01830	0.0184	0.0181	0.0187	yes	yes	-	-
18	0.80454	0.8065	0.7936	0.8193	0.01987	0.0201	0.0198	0.0204	yes	yes	-	-

Table E.30. Comparative Results for Links— Network 9, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i$ ($\overline{W}_{V,i}$)	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	10.82969	10.7130	9.2830	12.1430	0.08431	0.0859	0.0747	0.0970	yes	yes	-	-
2	10.11330	7.3907	6.7428	8.0391	0.07715	0.0582	0.0531	0.0633	no	no	36.8382	32.5601
3	14.08112	12.1085	9.2097	15.0073	0.13889	0.1252	0.0966	0.1538	yes	yes	-	-
4	13.28062	10.9894	9.0720	12.9067	0.13336	0.1158	0.0963	0.1352	no	yes	20.9040	-
5	9.18753	8.0597	7.2107	8.9088	0.04505	0.0401	0.0360	0.0441	no	no	13.9935	12.3441
6	9.43669	10.2965	9.3256	11.2675	0.04965	0.0551	0.0502	0.0599	yes	no	-	9.8911
7	10.06743	10.1888	8.1567	12.2210	0.07012	0.0719	0.0578	0.0861	yes	yes	-	-
8	8.87588	9.8135	8.7698	10.8571	0.06317	0.0704	0.0633	0.0776	yes	no	-	10.2699
9	10.88077	10.9877	8.2505	13.7249	0.14289	0.1480	0.1132	0.1829	yes	yes	-	-
10	10.60947	9.6542	8.2493	11.0590	0.15470	0.1443	0.1249	0.1638	yes	yes	-	-
11	9.18063	9.4586	8.3085	10.6087	0.08033	0.0887	0.0738	0.0935	yes	yes	-	-
12	9.36508	8.8715	8.0326	9.7104	0.08409	0.0814	0.0741	0.0886	yes	yes	-	-
13	9.45777	9.0118	7.7367	10.2870	0.07517	0.0723	0.0626	0.0822	yes	yes	-	-
14	9.08483	9.7527	8.0738	11.4317	0.08436	0.0911	0.0758	0.1064	yes	yes	-	-
15	9.02082	10.1279	8.6190	11.6368	0.07958	0.0901	0.0774	0.1027	yes	yes	-	-
16	11.30901	11.4192	9.2058	13.6325	0.10641	0.1096	0.0895	0.1296	yes	yes	-	-
17	10.30720	9.9798	8.2418	11.7178	0.11382	0.1119	0.0938	0.1300	yes	yes	-	-
18	8.56451	8.6055	7.5402	9.6709	0.10629	0.1062	0.0936	0.1189	yes	yes	-	-

Table E.31. Comparative Results for Links— Network 9, Parameter Set 4

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i$ ($\overline{W}_{V,i}$)	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.65209	8.9834	7.7018	10.2650	0.06879	0.0717	0.0618	0.0816	yes	yes	-	-
2	8.48438	7.9247	6.8872	8.9621	0.06584	0.0616	0.0539	0.0693	yes	yes	-	-
3	9.31511	9.7738	8.0551	11.4924	0.09531	0.1004	0.0832	0.1175	yes	yes	-	-
4	9.15373	8.9931	7.3969	10.5894	0.09479	0.0932	0.0775	0.1089	yes	yes	-	-
5	8.30435	7.7671	7.1632	8.3709	0.04115	0.0386	0.0357	0.0415	yes	yes	-	-
6	8.36855	8.3127	7.3597	9.2657	0.04457	0.0444	0.0395	0.0493	yes	yes	-	-
7	8.47984	8.6874	7.1876	10.1872	0.06007	0.0615	0.0512	0.0718	yes	yes	-	-
8	8.22636	8.5445	7.8668	9.2223	0.05902	0.0610	0.0564	0.0656	yes	yes	-	-
9	8.63817	8.4437	7.4681	9.4294	0.11542	0.1130	0.1007	0.1253	yes	yes	-	-
10	8.57198	11.2342	7.8357	14.6327	0.12755	0.1650	0.1171	0.2130	yes	yes	-	-
11	8.28005	8.1590	7.2769	9.0410	0.07828	0.0720	0.0645	0.0796	yes	yes	-	-
12	8.32175	7.8159	6.8894	8.7423	0.07562	0.0713	0.0633	0.0793	yes	yes	-	-
13	8.34757	7.2111	6.5527	7.8694	0.06719	0.0586	0.0536	0.0636	no	no	15.7600	14.6587
14	8.26821	8.1169	6.8246	9.4092	0.07744	0.0760	0.0644	0.0875	yes	yes	-	-
15	8.24218	8.0181	7.2766	8.7596	0.07340	0.0713	0.0649	0.0776	yes	yes	-	-
16	8.76559	8.1804	7.1296	9.2312	0.08442	0.0791	0.0692	0.0890	yes	yes	-	-
17	8.51052	8.5757	7.5837	9.5677	0.09574	0.0962	0.0857	0.1067	yes	yes	-	-
18	8.12849	8.7066	7.7634	9.6498	0.10049	0.1067	0.0957	0.1177	yes	yes	-	-

Table E.32. Comparative Results for Links— Network 10, Parameter Set 1

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.80090	0.8046	0.7898	0.8194	0.00178	0.0018	0.0018	0.0018	yes	no	-	3.8889
2	0.80698	0.8079	0.7955	0.8204	0.00178	0.0018	0.0018	0.0018	yes	no	-	3.8889
3	0.83489	0.8259	0.7989	0.8530	0.00737	0.0076	0.0074	0.0078	yes	yes	-	-
4	0.80044	0.8009	0.7797	0.8221	0.00623	0.0063	0.0061	0.0064	yes	yes	-	-
5	0.81251	0.8136	0.8004	0.8268	0.00177	0.0018	0.0018	0.0019	yes	yes	-	-
6	0.80301	0.7967	0.7837	0.8098	0.00188	0.0019	0.0019	0.0020	yes	yes	-	-
7	0.81092	0.8017	0.7867	0.8167	0.00289	0.0024	0.0024	0.0025	yes	yes	-	-
8	0.81202	0.7969	0.7812	0.8126	0.00267	0.0027	0.0027	0.0028	yes	yes	-	-
9	0.81185	0.8070	0.7898	0.8241	0.00262	0.0027	0.0026	0.0027	yes	yes	-	-
10	0.80358	0.8105	0.7979	0.8230	0.00229	0.0024	0.0023	0.0024	yes	yes	-	-
11	0.82272	0.8223	0.8021	0.8425	0.00631	0.0065	0.0063	0.0066	yes	yes	-	-
12	0.80464	0.8089	0.7879	0.8298	0.00556	0.0056	0.0055	0.0058	yes	yes	-	-
13	0.81338	0.8119	0.7978	0.8265	0.00238	0.0025	0.0024	0.0025	yes	yes	-	-
14	0.80416	0.8074	0.7914	0.8235	0.00284	0.0029	0.0029	0.0030	yes	no	-	2.0690
15	0.82506	0.8298	0.8081	0.8516	0.00646	0.0066	0.0065	0.0068	yes	yes	-	-
16	0.83166	0.8304	0.8107	0.8501	0.00609	0.0063	0.0062	0.0064	yes	no	-	3.3333
17	0.81070	0.8118	0.7836	0.8399	0.00550	0.0056	0.0054	0.0057	yes	yes	-	-
18	0.82153	0.8274	0.8081	0.8467	0.00711	0.0073	0.0072	0.0075	yes	no	-	2.6027
19	0.81524	0.8082	0.7908	0.8255	0.00563	0.0057	0.0056	0.0058	yes	yes	-	-
20	0.80664	0.7969	0.7709	0.8228	0.00588	0.0059	0.0058	0.0061	yes	yes	-	-
21	0.81373	0.8118	0.7868	0.8367	0.00755	0.0076	0.0074	0.0078	yes	yes	-	-
22	0.81352	0.8072	0.7816	0.8328	0.00544	0.0055	0.0053	0.0056	yes	yes	-	-
23	0.80430	0.7971	0.7694	0.8249	0.00650	0.0066	0.0064	0.0068	yes	yes	-	-
24	0.80643	0.7856	0.7677	0.8035	0.00699	0.0069	0.0068	0.0070	no	yes	2.6515	-
25	0.81072	0.8165	0.8022	0.8309	0.00620	0.0063	0.0062	0.0064	yes	yes	-	-
26	0.80954	0.8305	0.8022	0.8588	0.00709	0.0073	0.0071	0.0075	yes	yes	-	-

Table E.33. Comparative Results for Links— Network 10, Parameter Set 2

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{Y,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	0.80063	0.7975	0.7872	0.8078	0.00173	0.0018	0.0017	0.0018	yes	yes	-	-
2	0.80491	0.8094	0.7980	0.8208	0.00172	0.0018	0.0018	0.0018	yes	no	-	4.4444
3	0.81609	0.8171	0.7952	0.8390	0.00729	0.0075	0.0073	0.0076	yes	yes	-	-
4	0.80021	0.7943	0.7732	0.8153	0.00623	0.0062	0.0061	0.0064	yes	yes	-	-
5	0.80866	0.7962	0.7851	0.8073	0.00176	0.0018	0.0018	0.0018	no	yes	1.5649	-
6	0.80200	0.8072	0.7929	0.8214	0.00188	0.0019	0.0019	0.0020	yes	yes	-	-
7	0.80679	0.8069	0.7920	0.8217	0.00239	0.0024	0.0024	0.0025	yes	yes	-	-
8	0.80760	0.7982	0.7803	0.8161	0.00266	0.0027	0.0026	0.0027	yes	yes	-	-
9	0.80729	0.8121	0.7977	0.8264	0.00261	0.0027	0.0026	0.0027	yes	yes	-	-
10	0.80231	0.8042	0.7916	0.8169	0.00229	0.0024	0.0023	0.0024	yes	yes	-	-
11	0.81052	0.8059	0.7806	0.8313	0.00627	0.0063	0.0061	0.0065	yes	yes	-	-
12	0.80230	0.7943	0.7748	0.8139	0.00555	0.0056	0.0054	0.0057	yes	yes	-	-
13	0.80873	0.8111	0.7961	0.8262	0.00237	0.0024	0.0024	0.0025	yes	yes	-	-
14	0.80246	0.8051	0.7951	0.8151	0.00284	0.0029	0.0029	0.0029	yes	no	-	2.0690
15	0.81236	0.8369	0.8093	0.8645	0.00642	0.0067	0.0065	0.0069	yes	no	-	4.1791
16	0.81477	0.8257	0.7979	0.8536	0.00603	0.0061	0.0059	0.0063	yes	yes	-	-
17	0.80526	0.8004	0.7849	0.8158	0.00549	0.0055	0.0054	0.0056	yes	yes	-	-
18	0.81007	0.7971	0.7777	0.8164	0.00707	0.0071	0.0069	0.0072	yes	yes	-	-
19	0.80739	0.8148	0.7994	0.8302	0.00561	0.0057	0.0056	0.0058	yes	yes	-	-
20	0.80319	0.8187	0.7969	0.8406	0.00587	0.0060	0.0059	0.0061	yes	yes	-	-
21	0.80623	0.7936	0.7731	0.8140	0.00752	0.0075	0.0073	0.0077	yes	yes	-	-
22	0.80663	0.7825	0.7622	0.8028	0.00542	0.0054	0.0052	0.0055	no	yes	3.0637	-
23	0.80197	0.7992	0.7780	0.8205	0.00649	0.0065	0.0063	0.0067	yes	yes	-	-
24	0.80286	0.7932	0.7744	0.8119	0.00698	0.0069	0.0068	0.0071	yes	yes	-	-
25	0.80510	0.8014	0.7872	0.8156	0.00618	0.0062	0.0061	0.0063	yes	yes	-	-
26	0.80458	0.8012	0.7821	0.8203	0.00707	0.0071	0.0069	0.0072	yes	yes	-	-

Table E.34. Comparative Results for Links— Network 10, Parameter Set 3

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.11663	8.0079	7.4642	8.5517	0.00878	0.0087	0.0081	0.0093	yes	yes	-	-
2	8.97386	8.3661	7.2749	9.4578	0.00953	0.0090	0.0078	0.0101	yes	yes	-	-
3	10.02497	10.9275	8.7546	13.1004	0.04426	0.0489	0.0394	0.0583	yes	yes	-	-
4	8.02829	7.7259	6.4981	8.9536	0.03123	0.0302	0.0256	0.0347	yes	yes	-	-
5	9.78609	8.8167	8.0320	9.6014	0.01051	0.0097	0.0089	0.0105	no	yes	10.9949	-
6	8.36651	8.5924	7.5007	9.6842	0.00979	0.0101	0.0088	0.0113	yes	yes	-	-
7	9.16380	8.7518	7.7560	9.7476	0.01343	0.0130	0.0115	0.0144	yes	yes	-	-
8	9.26383	8.7764	8.0220	9.5308	0.01510	0.0145	0.0133	0.0157	yes	yes	-	-
9	9.29530	8.1245	7.1467	9.1024	0.01488	0.0132	0.0117	0.0148	no	no	14.4107	12.7273
10	8.40587	8.2367	7.5456	8.9279	0.01197	0.0118	0.0108	0.0127	yes	yes	-	-
11	9.46289	8.6428	7.3751	9.9105	0.03624	0.0337	0.0290	0.0383	yes	yes	-	-
12	8.30796	8.2934	7.4714	9.1154	0.02866	0.0289	0.0260	0.0317	yes	yes	-	-
13	8.6772	9.51958	7.6124	9.7420	0.01380	0.0128	0.0113	0.0143	yes	yes	-	-
14	8.39519	8.0776	7.1031	9.0520	0.01481	0.0143	0.0126	0.0161	yes	yes	-	-
15	9.62398	10.2913	8.7663	11.8162	0.03762	0.0408	0.0349	0.0466	yes	yes	-	-
16	10.09396	9.4841	8.2129	10.7554	0.03687	0.0353	0.0308	0.0399	yes	yes	-	-
17	8.72067	8.7287	7.7469	9.7105	0.02954	0.0296	0.0264	0.0328	yes	yes	-	-
18	9.29923	9.4171	7.7302	11.1041	0.04022	0.0410	0.0340	0.0479	yes	yes	-	-
19	9.05086	8.6055	7.3127	9.8983	0.03117	0.0300	0.0257	0.0343	yes	yes	-	-
20	8.42843	8.3546	7.1563	9.5529	0.03071	0.0305	0.0263	0.0347	yes	yes	-	-
21	8.77513	8.5657	7.7084	9.4231	0.04072	0.0401	0.0363	0.0439	yes	yes	-	-
22	8.94750	8.0625	7.2472	8.8778	0.02985	0.0272	0.0246	0.0299	no	yes	10.9767	-
23	8.26577	7.7050	6.7507	8.6592	0.03333	0.0311	0.0273	0.0348	yes	yes	-	-
24	8.37001	8.6955	6.8732	10.5177	0.03627	0.0376	0.0299	0.0452	yes	yes	-	-
25	8.68513	7.6756	6.9407	8.4105	0.03314	0.0297	0.0271	0.0323	no	no	13.1525	11.5825
26	8.58769	8.1314	7.2023	9.0604	0.03754	0.0357	0.0317	0.0396	yes	yes	-	-

Table E.35. Comparative Results for Links— Network 10, Parameter Set 4

link	mat. geom.	simulat.	conf. interval		mat. geom.	simulat.	conf. interval		Q_i in	$\overline{T}_i$ in	%m.g. Q_i	%m.g. $\overline{T}_i$
(i)	$\overline{Q}_i$	$\overline{Q}_i$	left	right	$\overline{T}_i\ (\overline{W}_{V,i})$	$\overline{T}_i$	left	right	C.I.	C.I.	from sim.	from sim.
1	8.03341	8.9138	7.1516	10.6759	0.00870	0.0096	0.0077	0.0115	yes	yes	-	-
2	8.26942	8.1774	7.4564	8.8985	0.00885	0.0088	0.0080	0.0095	yes	yes	-	-
3	8.48433	8.4720	7.3287	9.6152	0.03808	0.0383	0.0333	0.0432	yes	yes	-	-
4	8.00677	7.6893	6.6164	8.7623	0.03115	0.0299	0.0259	0.0338	yes	yes	-	-
5	8.47028	8.5065	7.6070	9.4060	0.00923	0.0093	0.0084	0.0102	yes	yes	-	-
6	8.09828	8.2565	7.6977	8.8154	0.00961	0.0097	0.0090	0.0103	yes	yes	-	-
7	8.30090	8.5488	7.6990	9.3986	0.01229	0.0126	0.0114	0.0139	yes	yes	-	-
8	8.33657	7.8079	7.0705	8.5454	0.01374	0.0129	0.0118	0.0141	yes	yes	-	-
9	8.32401	8.1479	7.1417	9.1542	0.01348	0.0132	0.0116	0.0148	yes	yes	-	-
10	8.10824	7.9889	7.3135	8.6643	0.01159	0.0114	0.0105	0.0123	yes	yes	-	-
11	8.33610	9.0977	7.7479	10.4475	0.03234	0.0352	0.0302	0.0401	yes	yes	-	-
12	8.07534	7.6297	6.8246	8.4347	0.02794	0.0262	0.0235	0.0289	yes	yes	-	-
13	8.40827	8.0316	7.1308	8.9325	0.01234	0.0118	0.0105	0.0131	yes	yes	-	-
14	8.10060	8.1219	7.3587	8.8850	0.01434	0.0143	0.0130	0.0156	yes	yes	-	-
15	8.39113	8.7558	7.2296	10.2821	0.03325	0.0344	0.0286	0.0401	yes	yes	-	-
16	8.47828	8.1603	7.0683	9.2523	0.03150	0.0306	0.0266	0.0346	yes	yes	-	-
17	8.17330	8.3068	7.4033	9.2102	0.02788	0.0282	0.0253	0.0311	yes	yes	-	-
18	8.31040	7.6586	6.6183	8.6988	0.03636	0.0336	0.0294	0.0378	yes	yes	-	-
19	8.24535	9.1599	7.8717	10.4481	0.02867	0.0318	0.0275	0.0362	yes	yes	-	-
20	8.10243	8.8511	6.9310	10.7712	0.02965	0.0320	0.0254	0.0386	yes	yes	-	-
21	8.18553	8.8117	7.7412	9.8821	0.03826	0.0411	0.0362	0.0460	yes	yes	-	-
22	8.22241	8.5880	7.8025	9.3734	0.02767	0.0288	0.0262	0.0314	yes	yes	-	-
23	8.06022	7.9230	6.9786	8.8675	0.03262	0.0321	0.0283	0.0359	yes	yes	-	-
24	8.08596	8.5224	7.3567	9.6881	0.03517	0.0368	0.0319	0.0417	yes	yes	-	-
25	8.16280	7.6223	6.7263	8.5183	0.03135	0.0294	0.0261	0.0327	yes	yes	-	-
26	8.14344	8.2899	7.2337	9.3460	0.03580	0.0362	0.0318	0.0405	yes	yes	-	-

APPENDIX F
NETWORK TOPOLOGIES AND PARAMETER SETS FOR DYNAMIC AND STATIC HIERARCHY COMPARISON

F.1. Network 1

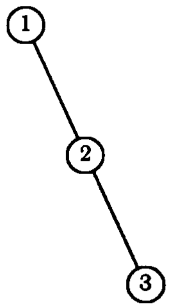


Figure F.1. Static Hierarchy 1 Topology

Network 1, Parameter Set 3

Number of nodes: 3

Table F.1. Link Numbers and Capacities— Network 1, Parameter Set 3

link	source node	destination node	capacity
1	2	1	1302.39001
2	1	2	2793.03760
3	3	2	1302.39001
4	2	3	2793.03760

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 0.0 & 24.8270 \\ 0.0 & 0.0 & 0.0 \\ 11.5768 & 0.0 & 0.0 \end{bmatrix}$$

F.2. Network 2

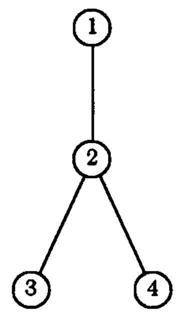


Figure F.2. Static Hierarchy 2 Topology

Network 2, Parameter Set 3

Number of nodes: 4

Table F.2. Link Numbers and Capacities— Network 2, Parameter Set 3

link	source node	destination node	capacity
1	2	1	5996.02490
2	1	2	3208.47754
3	3	2	3097.35010
4	2	3	1814.60254
5	4	2	2898.67505
6	2	4	1393.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 0.0 & 16.1298 & 12.390 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 27.5320 & 0.0 & 0.0 & 0.0 \\ 25.7660 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

F.3. Network 3

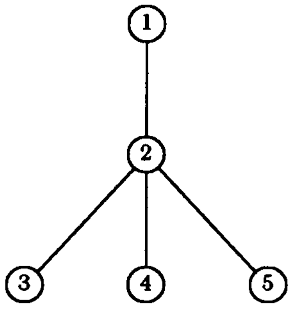


Figure F.3. Static Hierarchy 3 Topology

Network 3, Parameter Set 3

Number of nodes: 5

Table F.3. Link Numbers and Capacities— Network 3, Parameter Set 3

link	source node	destination node	capacity
1	2	1	7490.04736
2	1	2	5127.29980
3	3	2	2700.24756
4	2	3	1671.56995
5	4	2	2330.59497
6	2	4	884.09253
7	5	2	2496.20508
8	2	5	2571.63745

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 0.0 & 14.8584 & 7.8586 & 22.8590 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 24.0022 & 0.0 & 0.0 & 0.0 & 0.0 \\ 20.7164 & 0.0 & 0.0 & 0.0 & 0.0 \\ 21.8596 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

F.4. Network 4

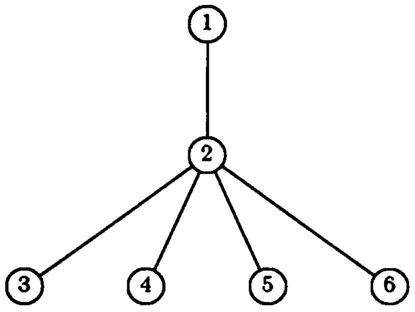


Figure F.4. Static Hierarchy 4 Topology

Network 4, Parameter Set 3

Number of nodes: 6

Table F.4. Link Numbers and Capacities— Network 4, Parameter Set 3

link	source node	destination node	capacity
1	2	1	9871.8750
2	1	2	13359.3750
3	3	2	2742.18750
4	2	3	2868.750
5	4	2	1673.43750
6	2	4	3037.50
7	5	2	3037.50
8	2	5	3656.250
9	6	2	2418.750
10	2	6	3796.8750

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 0.0 & 25.50 & 27.0 & 32.50 & 33.750 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 24.3750 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 14.8750 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 27.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 21.50 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

F.5. Network 5

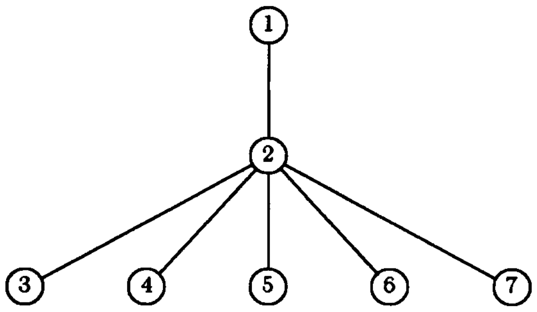


Figure F.5. Static Hierarchy 5 Topology

Network 5, Parameter Set 3

Number of nodes: 7

Table F.5. Link Numbers and Capacities— Network 5, Parameter Set 3			
link	source node	destination node	capacity
1	2	1	10080.42676
2	1	2	12738.19531
3	3	2	3225.39746
4	2	3	3067.71753
5	4	2	1075.90503
6	2	4	2305.91260
7	5	2	2540.04761
8	2	5	2284.31250
9	6	2	2274.00757
10	2	6	1986.20996
11	7	2	965.09253
12	2	7	3094.04248

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 0.0 & 27.2686 & 20.4970 & 20.3050 & 17.6552 & 27.5026 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 28.6702 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 9.5634 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 22.5782 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 20.2134 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 8.5786 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

F.7. Network 7

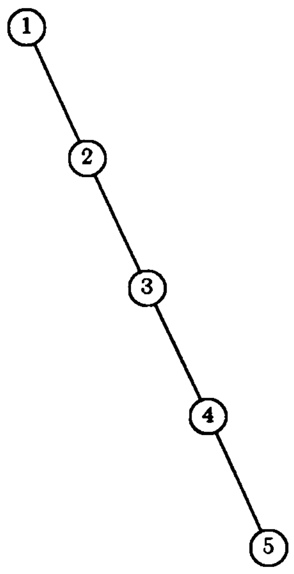


Figure F.6. Static Hierarchy 7 Topology

Network 7, Parameter Set 2

Number of nodes: 5

Table F.6. Link Numbers and Capacities— Network 7, Parameter Set 2

link	source node	destination node	capacity
1	2	1	3466.03491
2	1	2	966.64502
3	3	2	3466.03491
4	2	3	966.64502
5	4	3	3466.03491
6	3	4	966.64502
7	5	4	3466.03491
8	4	5	966.64502

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 2.1481 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 7.7023 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

F.8. Network 8

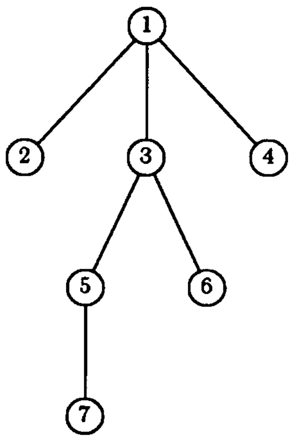


Figure F.7. Static Hierarchy 8 Topology

Network 8, Parameter Set 2

Number of nodes: 7

Table F.7. Link Numbers and Capacities— Network 8, Parameter Set 2

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 6.50 & 8.20 & 6.0 & 4.20 & 4.70 & 2.90 \\ 5.20 & 0.0 & 5.20 & 3.40 & 2.80 & 3.0 & 2.90 \\ 8.80 & 4.90 & 0.0 & 5.30 & 6.20 & 4.70 & 3.80 \\ 7.50 & 5.30 & 5.60 & 0.0 & 2.70 & 2.70 & 2.40 \\ 5.20 & 3.80 & 5.10 & 3.60 & 0.0 & 2.60 & 4.50 \\ 4.50 & 2.90 & 6.10 & 2.50 & 2.50 & 0.0 & 1.40 \\ 4.20 & 1.80 & 4.80 & 2.30 & 4.80 & 2.80 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 3

Number of nodes: 7

Table F.8. Link Numbers and Capacities— Network 8, Parameter Set 3

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 26.0 & 32.80 & 24.0 & 16.80 & 18.80 & 11.60 \\ 20.80 & 0.0 & 20.80 & 13.60 & 11.20 & 12.0 & 11.60 \\ 35.20 & 19.60 & 0.0 & 21.20 & 24.80 & 18.80 & 15.20 \\ 30.0 & 21.20 & 22.40 & 0.0 & 10.80 & 10.80 & 9.60 \\ 20.80 & 15.20 & 20.40 & 14.40 & 0.0 & 10.40 & 18.0 \\ 18.0 & 11.60 & 24.40 & 10.0 & 10.0 & 0.0 & 5.60 \\ 16.80 & 7.20 & 19.20 & 9.20 & 19.20 & 11.20 & 0.0 \end{bmatrix}$$

Network 8, Parameter Set 6

Number of nodes: 7

Table F.9. Link Numbers and Capacities— Network 8, Parameter Set 6

link	source node	destination node	capacity
1	2	1	10125.0
2	1	2	11340.0
3	3	1	22410.0
4	1	3	21285.0
5	4	1	11790.0
6	1	4	10395.0
7	5	3	16390.0
8	3	5	14310.0
9	6	3	8955.0
10	3	6	9225.0
11	7	5	9315.0
12	5	7	8055.0

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{bmatrix} 0.0 & 22.0 & 27.20 & 20.40 & 14.40 & 16.0 & 10.0 \\ 17.60 & 0.0 & 16.40 & 11.60 & 9.20 & 9.60 & 8.80 \\ 27.20 & 16.40 & 0.0 & 18.40 & 17.20 & 12.40 & 11.20 \\ 25.20 & 17.20 & 17.20 & 0.0 & 7.20 & 9.60 & 7.20 \\ 17.60 & 12.40 & 15.60 & 12.0 & 0.0 & 7.60 & 14.40 \\ 14.40 & 10.0 & 17.60 & 9.20 & 6.80 & 0.0 & 4.0 \\ 14.40 & 6.0 & 15.60 & 7.60 & 14.40 & 8.0 & 0.0 \end{bmatrix}$$

F.9. Network 9

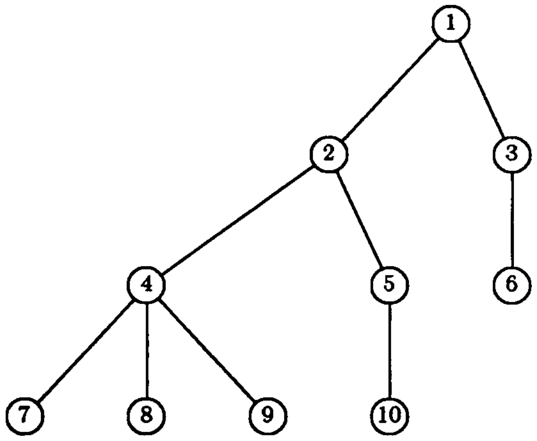


Figure F.8. Static Hierarchy 9 Topology

Network 9, Parameter Set 2

Number of nodes: 10

Table F.10. Link Numbers and Capacities— Network 9, Parameter Set 2

link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \left|\begin{array}{cccccccccc}
0.0 & 8.1855 & 8.8260 & 6.2215 & 5.8505 & 7.1795 & 2.7365 & 8.2275 & 3.1915 & 2.4970 \\
7.4250 & 0.0 & 5.1315 & 8.9640 & 6.3415 & 3.6765 & 6.6585 & 7.3890 & 4.6985 & 8.030 \\
9.0895 & 5.6285 & 0.0 & 2.910 & 3.0360 & 13.3170 & 1.0 & 1.2275 & 1.7305 & 1.4970 \\
4.9940 & 11.2835 & 2.4130 & 0.0 & 4.5810 & 2.4130 & 13.7965 & 12.3295 & 13.0660 & 7.0540 \\
4.3445 & 8.8025 & 2.4910 & 5.2995 & 0.0 & 2.2695 & 5.5570 & 4.5510 & 6.5210 & 13.0720 \\
6.8980 & 4.4070 & 15.1315 & 3.1855 & 2.2695 & 0.0 & 1.2275 & 1.0 & 1.0 & 1.9940 \\
3.2275 & 6.2835 & 2.0 & 15.5690 & 6.5210 & 1.0 & 0.0 & 11.0780 & 8.4670 & 2.2275 \\
3.9580 & 8.4670 & 1.7245 & 15.5570 & 7.5690 & 1.0 & 12.0780 & 0.0 & 9.7485 & 1.7305 \\
3.2275 & 7.1495 & 2.4970 & 15.2935 & 6.0120 & 1.0 & 8.6985 & 9.8265 & 0.0 & 2.2695 \\
3.0780 & 7.6110 & 2.4910 & 6.5870 & 14.3415 & 1.4970 & 3.0420 & 2.5030 & 3.0 & 0.0
\end{array}\right|$$

Network 9, Parameter Set 3

Number of nodes: 10

Table F.11. Link Numbers and Capacities— Network 9, Parameter Set 3

link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \left|\begin{array}{cccccccccc}
0.0 & 16.3710 & 17.6520 & 12.4430 & 11.7010 & 14.3590 & 5.4730 & 6.4550 & 6.3830 & 4.9940 \\
14.850 & 0.0 & 10.2630 & 17.9280 & 12.6830 & 7.3530 & 13.3170 & 14.7780 & 9.3770 & 16.060 \\
18.1790 & 11.2570 & 0.0 & 5.820 & 6.0720 & 26.6340 & 2.0 & 2.4550 & 3.4610 & 2.9940 \\
9.9880 & 22.4670 & 4.8260 & 0.0 & 9.1620 & 4.8260 & 27.5930 & 24.6590 & 26.1320 & 14.1080 \\
9.6890 & 17.6050 & 4.9820 & 10.5990 & 0.0 & 4.5390 & 11.1140 & 9.1020 & 13.0420 & 26.1440 \\
13.7960 & 8.8140 & 30.2630 & 6.3710 & 4.5390 & 0.0 & 2.4550 & 2.0 & 2.0 & 3.9880 \\
6.4550 & 12.4670 & 4.0 & 31.1380 & 13.0420 & 2.0 & 0.0 & 22.1560 & 16.9340 & 4.4550 \\
7.9160 & 16.9340 & 3.4490 & 31.1140 & 15.1380 & 2.0 & 24.1560 & 0.0 & 19.4970 & 3.4610 \\
6.4550 & 14.2990 & 4.9940 & 30.5870 & 12.0240 & 2.0 & 17.3770 & 19.6530 & 0.0 & 4.5390 \\
6.1560 & 15.2220 & 4.9820 & 13.1740 & 28.6830 & 2.9940 & 6.0840 & 5.0060 & 6.0 & 0.0
\end{array}\right|$$

Network 9, Parameter Set 4

Number of nodes: 10

Table F.12. Link Numbers and Capacities— Network 9, Parameter Set 4

link	source node	destination node	capacity
1	2	1	14030.66211
2	1	2	14405.17480
3	3	1	10822.61230
4	1	3	10712.13770
5	4	2	22613.06250
6	2	4	21019.16211
7	5	2	15782.6250
8	2	5	15633.0
9	6	3	8350.42480
10	3	6	7504.31250
11	7	4	12672.78711
12	4	7	12326.51270
13	8	4	13912.31250
14	4	8	11954.70020
15	9	4	12591.90039
16	4	9	11567.92480
17	10	5	9933.86230
18	5	10	9083.58789

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \left|\begin{array}{cccccccccc}
0.0 & 16.3710 & 17.6520 & 12.4430 & 11.7010 & 14.3590 & 5.4730 & 6.4550 & 6.3830 & 4.9940 \\
14.850 & 0.0 & 10.2630 & 17.9280 & 12.6830 & 7.3530 & 13.3170 & 14.7780 & 9.3770 & 16.060 \\
18.1790 & 11.2570 & 0.0 & 5.820 & 6.0720 & 26.6340 & 2.0 & 2.4550 & 3.4610 & 2.9940 \\
9.9880 & 22.4670 & 4.8260 & 0.0 & 9.1620 & 4.8260 & 27.5930 & 24.6590 & 26.1320 & 14.1060 \\
9.6890 & 17.6050 & 4.9620 & 10.5990 & 0.0 & 4.5390 & 11.1140 & 9.1020 & 13.0420 & 26.1440 \\
13.7960 & 8.8140 & 30.2630 & 6.3710 & 4.5390 & 0.0 & 2.4550 & 2.0 & 2.0 & 3.9880 \\
6.4550 & 12.4670 & 4.0 & 31.1380 & 13.0420 & 2.0 & 0.0 & 22.1560 & 16.9340 & 4.4550 \\
7.9160 & 16.9340 & 3.4490 & 31.1140 & 15.1380 & 2.0 & 24.1560 & 0.0 & 19.4970 & 3.4610 \\
6.4550 & 14.2990 & 4.9940 & 30.5870 & 12.0240 & 2.0 & 17.3770 & 19.6530 & 0.0 & 4.5390 \\
6.1560 & 15.2220 & 4.9620 & 13.1740 & 28.6830 & 2.9940 & 6.0840 & 5.0060 & 6.0 & 0.0
\end{array}\right|$$

F.10. Network 10

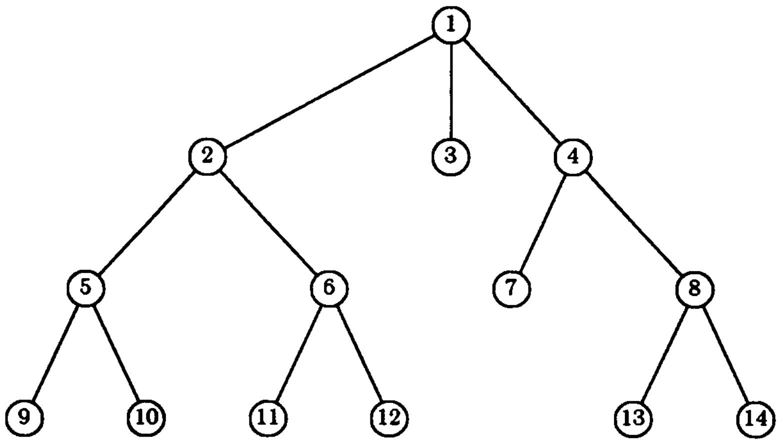


Figure F.9. Static Hierarchy 10 Topology

Network 10, Parameter Set 1

Number of nodes: 14

Table F.13. Link Numbers and Capacities— Network 10, Parameter Set 1

link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \left|\begin{array}{cccccccccccccc}
0.0 & 5.7488 & 11.0818 & 5.8340 & 6.7476 & 10.5835 & 10.6250 & 9.0939 & 11.1769 & 8.7944 & 16.1064 & 9.0483 & 11.0449 & 9.2063 \\
10.5813 & 0.0 & 10.2478 & 7.4006 & 9.1735 & 12.7763 & 14.8446 & 7.1192 & 11.8412 & 4.8178 & 5.5888 & 8.2343 & 8.3704 & 3.5345 \\
9.1366 & 9.6624 & 0.0 & 11.4375 & 3.8446 & 6.3161 & 9.9744 & 13.2671 & 6.5070 & 2.0362 & 8.6899 & 11.6462 & 11.6216 & 6.5572 \\
14.8064 & 9.4843 & 3.3931 & 0.0 & 9.0121 & 6.0147 & 4.9913 & 5.8393 & 8.8359 & 7.8506 & 14.7167 & 12.1079 & 7.1788 & 8.2727 \\
7.4387 & 3.2184 & 4.5579 & 11.3410 & 0.0 & 9.4902 & 8.8118 & 12.6711 & 9.4561 & 12.5104 & 15.1969 & 9.7850 & 11.4093 & 9.6009 \\
14.0147 & 5.5070 & 9.1460 & 10.8748 & 5.8680 & 0.0 & 14.1139 & 9.0063 & 12.7850 & 10.4093 & 6.1226 & 11.1776 & 1.6986 & 9.1829 \\
7.2611 & 8.3429 & 9.9510 & 6.3470 & 15.0637 & 7.7179 & 0.0 & 8.6839 & 13.6296 & 9.5157 & 13.4387 & 13.0287 & 11.0396 & 4.2833 \\
3.6390 & 9.9163 & 14.4240 & 11.1547 & 7.7201 & 11.8265 & 13.0898 & 0.0 & 11.1426 & 14.8680 & 10.7563 & 8.9725 & 7.6009 & 13.6899 \\
13.7616 & 2.4655 & 14.4930 & 7.6284 & 9.8533 & 7.8680 & 8.7582 & 7.5519 & 0.0 & 4.7435 & 15.3463 & 9.1064 & 11.5345 & 12.4093 \\
10.0977 & 15.1554 & 6.6424 & 12.4194 & 14.7710 & 14.9551 & 13.2746 & 4.2237 & 9.5968 & 0.0 & 9.2493 & 16.0362 & 5.4006 & 14.4206 \\
15.7669 & 10.7623 & 9.1279 & 7.0053 & 16.6624 & 12.1045 & 15.5666 & 9.6349 & 11.5794 & 6.4433 & 0.0 & 15.9083 & 7.9725 & 4.7635 \\
4.9404 & 7.9370 & 4.3791 & 7.5391 & 6.1909 & 13.2652 & 11.0434 & 8.0456 & 9.2162 & 12.9766 & 3.8265 & 0.0 & 11.8453 & 5.4930 \\
13.8668 & 14.9457 & 14.7860 & 6.8091 & 5.7882 & 2.0777 & 9.6530 & 7.8118 & 11.430 & 8.8627 & 7.1354 & 8.2377 & 0.0 & 12.0871 \\
9.0603 & 7.5210 & 16.2599 & 16.7401 & 4.7669 & 12.1641 & 9.5941 & 4.0864 & 6.5210 & 9.9694 & 10.2799 & 14.8265 & 8.0883 & 0.0
\end{array}\right|$$

Network 10, Parameter Set 2

Number of nodes: 14

Table F.14. Link Numbers and Capacities— Network 10, Parameter Set 2

link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$\Gamma =$

0.0	5.7498	11.0818	5.8840	6.7476	10.5835	10.6250	9.0939	11.1769	8.7944	16.1064	9.0493	11.0449	9.2063
10.5813	0.0	10.2478	7.4006	9.1735	12.7763	14.8446	7.1192	11.8412	4.8178	5.5898	8.2343	8.3704	3.5345
9.1366	9.6624	0.0	11.4375	3.8446	6.3161	9.9744	13.2671	6.5070	2.0362	8.6899	11.6462	11.6216	6.5572
14.8064	9.4843	3.3931	0.0	9.0121	6.0147	4.9913	5.8393	8.8359	7.8506	14.7167	12.1079	7.1788	8.2727
7.4387	3.2184	4.5579	11.8410	0.0	9.4802	8.8118	12.6711	9.4561	12.5104	15.1969	9.7850	11.4093	9.6009
14.0147	5.5070	9.1460	10.8748	5.8680	0.0	14.1139	9.0053	12.7850	10.4093	6.1226	11.1776	1.6986	9.1829
7.2611	8.3429	9.9510	6.3470	15.0637	7.7179	0.0	8.6839	13.6296	9.5157	13.4387	13.0287	11.0896	4.2833
3.6390	9.9163	14.4240	11.1547	7.7201	11.8265	13.0898	0.0	11.1426	14.8680	10.7563	8.9725	7.6009	13.6899
13.7616	2.4655	14.4930	7.6284	9.8533	7.8680	8.7582	7.5519	0.0	4.7435	15.8468	9.1064	11.5845	12.4093
10.0977	15.1554	6.6424	12.4194	14.7710	14.9551	13.2746	4.2237	9.5968	0.0	9.2493	16.0862	5.4006	14.4206
15.7669	10.7628	9.1279	7.0053	16.6624	12.1045	15.5666	9.6349	11.5794	6.4433	0.0	15.9083	7.9725	4.7635
4.9404	7.9370	4.3791	7.5391	6.1909	13.2652	11.0434	8.0456	9.2162	12.9766	3.8265	0.0	11.8453	5.4930
13.8668	14.9457	14.7850	6.8091	5.7382	2.0777	9.6530	7.8118	11.430	8.8627	7.1854	8.2377	0.0	12.0871
9.0603	7.5210	16.2599	16.7401	4.7669	12.1641	9.5941	4.0864	6.5210	9.9894	10.2799	14.8265	8.0883	0.0

Network 10, Parameter Set 3

Number of nodes: 14

Table F.15. Link Numbers and Capacities— Network 10, Parameter Set 3

link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$\Gamma =$

0.0	11.4976	22.1686	11.6680	13.4962	21.1670	21.250	18.1878	22.3538	17.5888	32.2128	18.0966	22.0698	18.4126
21.1626	0.0	20.4956	14.8012	18.3470	25.5526	29.6892	14.2384	23.6824	9.6356	11.1776	16.4686	16.7408	7.0690
18.2732	19.3248	0.0	22.8750	7.6892	12.6322	19.9488	26.5342	13.0140	4.0724	17.3798	23.2924	23.2432	13.1144
29.6168	18.9686	6.7862	0.0	18.0242	12.0294	9.9826	11.6786	17.6718	15.7012	29.4334	24.2168	14.3576	16.5454
14.8774	6.4368	9.1158	22.6820	0.0	18.9604	17.6286	25.3422	18.9122	25.0208	30.3938	19.570	22.8186	19.2018
23.0294	11.0140	18.2920	21.7496	11.7360	0.0	28.2278	18.0106	25.570	20.8186	12.2452	22.3552	8.3972	18.3658
14.5222	16.6858	19.9020	12.6940	30.1274	15.4358	0.0	17.3678	27.2592	19.0314	26.8774	26.0674	22.0792	8.5666
7.2780	19.8326	28.8480	22.3094	15.4402	23.6580	26.1796	0.0	22.2852	29.7360	21.5126	17.9450	15.2018	27.3798
27.5232	4.9310	28.9860	15.2568	19.7066	15.7360	17.5164	15.1038	0.0	9.4870	30.6926	18.2128	23.0690	24.8186
20.1954	30.3108	13.2848	24.8388	29.5420	29.9102	26.5492	8.4474	19.1936	0.0	18.4986	32.0724	10.8012	28.8412
31.5338	21.5246	18.2558	14.0106	33.3248	24.2090	31.1332	19.2698	23.1588	12.8866	0.0	31.8166	15.9450	9.5270
9.8808	15.8740	8.7582	15.0782	12.3818	26.5304	22.0868	16.0912	18.4324	25.9532	7.6530	0.0	23.6906	10.9860
27.7336	29.8914	29.570	13.6182	11.4764	4.1554	19.3060	15.6236	22.860	17.7254	14.2708	16.4754	0.0	24.1742
18.1206	15.0420	32.5198	33.4802	9.5338	24.3282	19.1882	8.1728	13.0420	19.9788	20.5598	29.6530	16.1766	0.0

Network 10, Parameter Set 4

Number of nodes: 14

Table F.16. Link Numbers and Capacities— Network 10, Parameter Set 4

link	source node	destination node	capacity
1	2	1	103883.54688
2	1	2	104707.39063
3	3	1	24906.77930
4	1	3	28910.00195
5	4	1	102578.96094
6	1	4	95663.52344
7	5	2	75669.59375
8	2	5	67953.78125
9	6	2	69186.84375
10	2	6	78595.74219
11	7	4	28868.19727
12	4	7	32476.65625
13	8	4	76224.21875
14	4	8	63444.64453
15	9	5	28241.97656
16	5	9	30086.48242
17	10	5	32904.62891
18	5	10	25609.02734
19	11	6	32242.00586
20	6	11	30702.08203
21	12	6	24007.11719
22	6	12	33326.01172
23	13	8	27774.04492
24	8	13	25831.19141
25	14	8	29227.02734
26	8	14	25537.76953

Mean message length: $\mu^{-1} = 100.0$ bits

Arrival rate matrix:

$$\Gamma = \begin{vmatrix}
0.0 & 11.4976 & 22.1636 & 11.6680 & 13.4952 & 21.1670 & 21.250 & 18.1878 & 22.3538 & 17.5888 & 32.2128 & 18.0966 & 22.0898 & 18.4126 \\
21.1626 & 0.0 & 20.4956 & 14.8012 & 18.3470 & 25.5526 & 29.6892 & 14.2384 & 23.6824 & 9.6356 & 11.1776 & 16.4686 & 16.7408 & 7.0690 \\
18.2732 & 19.3248 & 0.0 & 22.8750 & 7.6892 & 12.6322 & 19.9488 & 26.5342 & 13.0140 & 4.0724 & 17.3798 & 23.2924 & 23.2432 & 13.1144 \\
29.6168 & 18.9686 & 6.7862 & 0.0 & 18.0242 & 12.0294 & 9.9826 & 11.6786 & 17.6718 & 15.7012 & 29.4334 & 24.2158 & 14.3576 & 16.5454 \\
14.8774 & 6.4368 & 9.1158 & 22.6820 & 0.0 & 18.9604 & 17.6236 & 25.3422 & 18.9122 & 25.0208 & 30.3938 & 19.570 & 22.8186 & 19.2018 \\
28.0294 & 11.0140 & 18.2920 & 21.7496 & 11.7360 & 0.0 & 28.2278 & 18.0106 & 25.570 & 20.8186 & 12.2452 & 22.3552 & 3.3972 & 18.3658 \\
14.5222 & 16.6868 & 19.9020 & 12.6940 & 30.1274 & 15.4358 & 0.0 & 17.3678 & 27.2592 & 19.0314 & 26.8774 & 26.0574 & 22.0792 & 8.5666 \\
7.2780 & 19.8326 & 28.8480 & 22.3094 & 15.4402 & 23.6530 & 26.1796 & 0.0 & 22.2852 & 29.7360 & 21.5126 & 17.9450 & 15.2018 & 27.3798 \\
27.5232 & 4.9310 & 28.9860 & 15.2568 & 19.7066 & 15.7360 & 17.5164 & 15.1088 & 0.0 & 9.4870 & 30.6926 & 18.2128 & 28.0690 & 24.8186 \\
20.1954 & 30.3108 & 13.2848 & 24.8388 & 29.5420 & 29.9102 & 26.5492 & 8.4474 & 19.1936 & 0.0 & 18.4986 & 32.0724 & 10.8012 & 28.8412 \\
31.5338 & 21.5246 & 18.2558 & 14.0106 & 33.3248 & 24.2090 & 31.1332 & 19.2698 & 23.1588 & 12.8866 & 0.0 & 31.8166 & 15.9450 & 9.5270 \\
9.8806 & 15.8740 & 8.7582 & 15.0782 & 12.8818 & 26.5304 & 22.0868 & 16.0912 & 18.4324 & 25.9532 & 7.6530 & 0.0 & 28.6906 & 10.9860 \\
27.7336 & 29.8914 & 29.570 & 13.6182 & 11.4764 & 4.1554 & 19.3060 & 15.6236 & 22.860 & 17.7254 & 14.2706 & 16.4754 & 0.0 & 24.1742 \\
18.1206 & 15.0420 & 32.5198 & 33.4802 & 9.5338 & 24.3282 & 19.1882 & 8.1728 & 13.0420 & 19.9788 & 20.5598 & 29.6530 & 16.1766 & 0.0
\end{vmatrix}$$

APPENDIX G
SIMSCRIPT II.5 DYNAMIC HIERARCHY SIMULATION SOURCE CODE

The simulation experiments are performed as follows:

For each dynamic hierarchy topology/parameter set combination:

(1) Prepare a data set according to file format 1 (Figure G.1).

(2) Include the data set in a file with the SIMSCRIPT source code.

(3) Submit the file for compilation and execution.

(4) Receive the output file of results (and listing).

The output file contains a report of statistical observations for each batch and a report of statistical estimates accumulated over all batches.

The following notation is employed in all of the file formats:

- An item in angle brackets ("< . . . >") describes a required data item and indicates the position at which it is expected.
- An item in braces ("{ . . . }") provides additional description of a data item but does not appear in the input file.
- A numeric data item (for example, "1" or "2") is entered in the input file as shown.

```
<number of batches>
<number of observations>
<$t_{1-\alpha/2,n-1}$>  { $t$ statistic for confidence interval }
                        { construction, $n$ = number of batches }
<number of nodes>  { $N$ }
<number of environment states>    { $M$ }
<$\Gamma^{(1)}$>          { traffic matrix for environment state 1 }
<$\Gamma^{(2)}$>          { traffic matrix for environment state 2 }
.
.
.
<$\Gamma^{(M)}$>          { traffic matrix for environment state $M$ }
1 <source node> <destination node> <link 1 capacity>             { for link 1 }
2 <source node> <destination node> <link 2 capacity>             { for link 2 }
.
.
.
<L> <source node> <destination node> <link L capacity>          { for link L }
<$\mu^{-1}$>          { mean message length }
<environment process generator>
<environment stationary probabilities>
1
<node 1 routing table— nodes>
<node 1 routing table— links>
2
<node 2 routing table— nodes>
<node 2 routing table— links>
.
.
.
<N>
<node N routing table— nodes>
<node N routing table— links>
```

Figure G.1. File Format 1— Dynamic Hierarchy Simulation

Routing tables for node i are constructed as follows:

Node i routing table— nodes:

```
1      <next node>
2      <next node>
.
.
.
<$i-1$>  <next node>
<$i+1$>  <next node>
```

.
.
.
<N>   <next node>

Node i routing table— links:

1	<next link>
2	<next link>
.	
.	
.	
$<i-1>$	<next link>
$<i+1>$	<next link>
.	
.	
.	
<N>	<next link>

In the first table, a line of the form 'j <next node>' means that for a message at node i, if it is destined for node j, then the next node in its path is <next node>. In the second table, a line of the form 'k <next link>' means that for a message at node i, if the next node in its path is k, then the next link on which it must be transmitted is <next link>. If nodes i and k are not directly connected, then <next link> =-1.

```
//XXXXXXXX JOB XXXXX,RMOOSE,REGION=3072K,TIME=1
/*LONGKEY    XXXXX
/*JOBPARM    LINES=4
/*PRIORITY   IDLE
/*ROUTE PRINT  VM1.XXXXX
//STEP1 EXEC  SIM25CLG,
//        PARM.SIM='LOAD,ID,TRACE2,NOTERM,CHK,REN=NEW'
//SIM.SYSIN DD *
''
''
'' File: dynhier smscrpt
'' Modeler/Programmer: Robert L. Moose, Jr.
''             Dept. of Computer Science
''             Va. Tech
''             Blacksburg, VA  24060
'' Model: Network of M/M/1 queue in a random environment (RE-network)
''     as a model of the dynamic hierarchy.
''
'' Modeling History:
''   11-18-86  Robert L. Moose, Jr.
''         Initial implementation.
''   02-05-87  Robert L. Moose, Jr.
''         Beginning of conversion to SIMSCRIPT II.5.
''   05-08-87  Robert L. Moose, Jr.
''         Credibility assessment.
''   05-28-87  Robert L. Moose, Jr.
''         Conversion to batch means experimental design.
```

```
''
'' *******************************************************************
''
'' Purpose:
''   Simulate (in SIMSCRIPT II.5) a network of M/M/1 queues in a random
'' environment (RE-network), which forms the basic theoretical model of
'' the dynamic hierarchy. Collect estimates necessary to construct
'' confidence intervals for various statistics of interest. These
'' confidence intervals and the corresponding point estimates are used
'' to study the impact of approximations employed in a queueing model
'' of the dynamic hierarchy.
''
'' System description:
''   The dynamic hierarchy is a network architecture that represents a
'' generalization of the conventional tree-structured architecture in
'' which the network operates under a centralized, hierarchical mode of
'' control. A dynamic topology is enabled by allowing the apex of the
'' hierarchy to vary among a specified set of nodes. Under each of the
'' resultant tree-structured topologies, the root (apex) node exersizes
'' primary control. Secondary capabilities filter down through the
'' remainder of the network in a hierarchical manner.
''   External to the network, there exist multiple situations. An apex
'' node (and the corresponding topology) is designated as the one most
'' beneficiaal for each situation. The network always conforms to one of
'' the specified topologies. When a situation change occurs, the network
'' undergoes a transition, with the appropriate node becoming the apex
'' of the reconfigured topology.
''   The external situations are considered states of an environment,
'' which is included as part of the system of interest.
''   Message traffic in the dynamic hierarchy is handled in a
'' store-and-forward fashion. A message arrives to its source node from
'' outside the network. Then for each node and link in the path of the
'' message to its destination, the message is stored in the queue for the
'' designated outgoing link from the node. After reaching the front of
'' the queue, the message is transmitted to the next node in its path.
'' Fixed, shortest path message routing is used. This process continues
'' until the message reaches its destination node. The message then
'' departs the network.
''
'' Model description:
''   The dynamic hierarchy is modeled as a network of M/M/1 queues in
'' a random environment (an RE-network). The environment is assumed to
'' be an M state, irreducible Markov process E = (E(t): t >= 0). On
'' (E(t) = i)
''   - messages with source node j and destination node k (jk-messages)
''     arrive according to a Poisson processs wit rate gamma(i, j, k).
''   - the length of a message entering service at any link l is
''     (re)assignned according to an exponential distribution with mean
''     1/mu.
'' Other modeling assumptions include the following:
''   - given the state of the environment process, the arrival processes
''     from outside of the network (external arrival processes) are
''     mutually independent, the message length (service) processes are
''     mutually idependent, and the collections of external arrival
''     processes and message length processes are independent.
''   - nodal processing times are negligible.
''   - messages pass through the network in a store-and-forward fashion
''     over fixed paths (of links).
''   - first-come-first-served queue with unlimited buffer space for
''     each link.
''   - each link transmits in one direcction. Bidirectional transmission
''     is enabled by connecting nodes with link pairs.
''
'' Experimental design:
''   A single run consisting of multiple batches is run. The lengths
'' of the transient period and of each batch depend on the values
'' specified for number of transient period observations and batch size,
'' respectively. Observations during the transient period are deleted.
'' Statistics accumulated over all batches include: Mean and variance of
'' queue length for each link. Mean and variance of delay at each link.
'' Mean and variance of network message delay. Additionally, a
'' confidence interval is constructed for each of these statistics.
''   Observations of the above and other statistics are printed for each
'' batch. Lastly, for verification purposes, observations of mean
'' interarrival time and mean service time may be collected and printed.
''
'' Implementation summary:
'' The process and entity types in this implementation are:
```

```
''   - msggen: generate a Poisson process in a random environment of
''     jk-messages for a given source/destination pair jk.
''   - message: describe and execute the movement of a message
''     through the network.
''   - node: passive object type (defined as temporary entity)
''     representing a network node. Contains routing tables for the
''     node.
''   - tlink: describe and execute the actions of a link and the
''     attached queue.
''   - environment: generate a Markov process of states of the external
''     environment. Also, restart processses waiting for environment
''     transition (see notes).
'' Top level procedures are:
''   - gennet: create network links.
''   - inpnetparam: read network parameters.
''   - inpsimparam: read simulation parameters.
''   - reportbatch: print per batch statistical observations.
''   - finalanalysis: calculate and print statistics accumulated over
''     all batches.
''   - initstate: choose initial environment state.
''   - transition: choose next environment state and time of next
''     transition.
''   - batchreset: reset the per batch accumulation variables at the
''     end of the transient period and after each batch.
''
'' Implementation notes:
''   (1) A "msggen" object is created for each source/destination node
''        pair jk (j != k). These processes all draw from the same
''        random variate stream but the resultant arrival processes are
''        assumed to be independent.
''   (2) In general, the generation of a full interarrival interval(II)
''        requires the generation of multiple, unsuccessful interarrival
''        intervals. An unsuccessful II is one that would put the time
''        of the next arrival past the time of the next environment
''        transition. Upon generating an unsuccessful II, the
''        "msggen" enters a queue in the "environment" object and
''        then becomes passive. After an environment transition, the
''        "environment" activates all object waiting in its queues. A
''        successful II is one that puts the time of the next arrival
''        before the time of the next environment transition. A
''        "msggen" alternates between generating IIs and waiting in an
''        "environment" queue until it generates a successful II.
''        The arrival then occurs at the designated time.
''   (3) The "tlink" objects all draw (message lengths) from the same
''        random variate stream. However, the resultant service
''        processes are assumed to be independent.
''   (4) The generation of full service times follows a scheme similar
''        to the one described above for generating interarrival
''        intervals. "Tlink" objects wait in a separate queue in the
''        "environment".
''   (5) A "message" object selects the next node and link in its path
''        by inspecting the routing tables of the current "node" object.
''   (6) A "tlink" may be idle (suspended) because: (a) it is awaiting
''        arrrivals or (b) it is waiting in the "environment" queue. An
''        arriving "message" only activates an idle "tlink" if it is
''        idle for reason (a).
''   (7) To determine the next environment process state, a
''        conditional cumulative distribution is needed. Given that the
''        current state is i, the required CDF is computed by summing
''        normalized elements of row i of the infintesimal generator of
''        the environment process.
''   (8) All objects are created once and used for all batches.
''   (9) Output statements for printing an execution trace are
''        preceded by the comment "trace diags".
''   (10) Statements used in the verification process are preceded
''        by the comment "verif".
''   (11) Statements used to conduct pilot runs are preceded by the
''        comment "pilot".
''   (12) This simulation ogiginally used the method of replications
''        as the basis for the experimental design. The general program
''        structure and placement of some data structures may reflect
''        this fact.
''
'' ***********************************************************************


PREAMBLE
```

```
   DEFINE AS.INT TO MEAN AS INTEGER VARIABLE
   DEFINE AS.REAL TO MEAN AS REAL VARIABLE


PROCESSES

   EVERY MSGGEN
     HAS  A MGSOURCE, A MGDEST, AND A INTARRVFL
     AND MAY BELONG TO THE GTRANSQ
     DEFINE MGSOURCE AND MGDEST AS.INT
     '' VERIF
     DEFINE INTARRVFL AS.REAL

   EVERY MESSAGE
     HAS A MSGID, A MSGSOURCE, A MSGDEST, A NETATIME, A NETDTIME,
       AND A LINKATIME
     AND MAY BELONG TO A QUEUE
     DEFINE MSGID, MSGSOURCE, AND MSGDEST AS.INT
     DEFINE NETATIME, NETDTIME, AND LINKATIME AS.REAL

   EVERY TLINK
     HAS A LKID, A CAPACITY, A FROMNODE, A TONODE, A TRANSWAIT,
       A LINKDELAY, AND A SERVFL
     AND OWNS A QUEUE
     AND MAY BELONG TO THE LTRANSQ
     DEFINE LKID, FROMNODE, AND TONODE AS.INT
     DEFINE CAPACITY AND LINKDELAY AS.REAL
     DEFINE TRANSWAIT AS.INT '' USED AS BOOLEAN
     '' VERIF
     DEFINE SERVFL AS.REAL

   EVERY ENVIRONMENT
     HAS A GENERATOR, A STATPROB, A NOSTATE, A THISSTATE, AND
       A TRANSTIME
     AND OWNS A LTRANSQ AND A GTRANSQ
     DEFINE GENERATOR AS.INT '' USED AS REAL 2-DIM ARRAY
     DEFINE STATPROB AS.INT '' USED AS REAL 1-DIM ARRAY
     DEFINE NOSTATE AND THISSTATE AS.INT
     DEFINE TRANSTIME AS.REAL

TEMPORARY ENTITIES

   '' NODE IS ACTUALLY A PERMANENT ENTITY BUT IS DECLARED AS TEMPORARY
   '' BECAUSE OF MODELER USAGE PREFERENCE
   EVERY NODE
     HAS A NDID, A NROUTE, AND A LROUTE
     DEFINE NDID AS.INT
     DEFINE NROUTE AND LROUTE AS.INT '' USED AS 1-DIM INTEGER ARRAY

   '' THIS VERSION OF THE SIMULATION DOES NOT CONTAIN AN OBJECT TO
   '' REPRESENT THE OUTSIDE. NETDELAY IS A SYSTEM ATTRIBUTE


PRIORITY ORDER IS ENVIRONMENT, TLINK, MSGGEN, MESSAGE


THE SYSTEM
   HAS A NOENV, A NONODE, A NOLINK, A NODES, A LINKS, A ARRVPROCS, A
     ARRVRATE, A ZEROARRV, A NODEIDS, A CAPACITIES, A ENVGEN, A ENVPROB,
     A MEANMLEN, A NETDELAY, AND A MSGNO
   DEFINE NOENV, NONODE, AND NOLINK AS.INT
   DEFINE NODES AND LINKS AS INTEGER 1-DIM ARRAYS '' USED AS REF
   DEFINE ARRVPROCS AS A INTEGER 2-DIM ARRAY '' USED AS REF
   DEFINE ARRVRATE AS A REAL 3-DIM ARRAY
   DEFINE ZEROARRV AS A INTEGER 2-DIM ARRAY '' USED AS BOOLEAN
   DEFINE NODEIDS AS A INTEGER 2-DIM ARRAY
   DEFINE CAPACITIES AS A REAL 1-DIM ARRAY
   DEFINE ENVGEN AS A REAL 2-DIM ARRAY
   DEFINE ENVPROB AS A REAL 1-DIM ARRAY
   DEFINE MEANMLEN AS.REAL
   DEFINE NETDELAY AS.REAL
   DEFINE MSGNO AS.INT

DEFINE LINKINX, NODEINX1, NODEINX2, AND BATCHINX AS.REAL
DEFINE TRANSSEC, SSTSIMSEC, STARTTIME, AND TVAL AS.REAL
DEFINE NOBATCH, NOTRANSOBS, AND BATCHSIZE AS.INT
DEFINE SIMDONE AS.INT '' USED AS BOOLEAN
DEFINE ALINK AS.INT '' USED AS REF
DEFINE NMEANDELAY AND NVARDELAY AS.REAL
'' OVERALL ACCUMULATION VARIABLES FOR LINKS
DEFINE SLKMQLEN AND S2LKMQLEN AS REAL 1-DIM ARRAYS
DEFINE SLKVQLEN AND S2LKVQLEN AS REAL 1-DIM ARRAYS
DEFINE SLKMXQLEN AS A REAL 1-DIM ARRAY
DEFINE SLKMDELAY AND S2LKMDELAY AS REAL 1-DIM ARRAYS
DEFINE SLKVDELAY AND S2LKVDELAY AS REAL 1-DIM ARRAYS
'' PER BATCH ACCUMULATION VARIABLE FOR NETWORK
DEFINE NETNOMSG AS.INT MONITORED ON THE LEFT
'' OVER ACCUMULATION VARIABLE FOR NETWORK
DEFINE SNETNOMSG AS.REAL
'' VERIF
'' DEFINE IAT AND ERT AS.REAL
'' PILOT
'' DEFINE BATCHINX AS.INT


DEFINE INITSTATE AS A INTEGER FUNCTION
DEFINE DISCRETE AS A INTEGER FUNCTION
DEFINE TRANSITION AS A ROUTINE GIVEN 3 ARGUMENTS YIELDING 3 ARGUMENTS
DEFINE RTBUILD AS A ROUTINE GIVEN 0 ARGUMENTS
DEFINE GENNET AS A ROUTINE GIVEN 0 ARGUMENTS
DEFINE INPNETPARAM AS A ROUTINE YIELDING 5 ARGUMENTS
DEFINE INPSIMPARAM AS A ROUTINE YIELDING 4 ARGUMENTS
DEFINE REPORTBATCH AS A ROUTINE GIVEN 0 ARGUMENTS
DEFINE BATCHRESET AS A ROUTINE GIVEN 0 ARGUMENTS
DEFINE FINALANALYSIS AS A ROUTINE GIVEN 1 ARGUMENT


'' PER BATCH ACCUMULATION VARIABLES
ACCUMULATE LKMQLEN AS THE PERBATCH MEAN, LKVQLEN AS THE PERBATCH
  VARIANCE, AND LKMAXQLEN AS THE PERBATCH MAXIMUM OF N.QUEUE
TALLY LKNOMSG AS THE PERBATCH NUMBER, LKMDELAY AS THE PERBATCH MEAN, AND
  LKVDELAY AS THE PERBATCH VARIANCE OF LINKDELAY
'' NETNOMSG IS THE PERBATCH NUMBER OF NETDELAY BUT IS CALCULATED
'' EXPLICITLY SO THAT IT CAN BE MONITORED (ON THE LEFT).
TALLY NETMDELAY AS THE PERBATCH MEAN AND NETS2DELAY AS THE PERBATCH
  SUM.OF.SQUARES OF NETDELAY
'' OVERALL ACCUMULATION VARIABLES FOR NETWORK
TALLY MNETMDELAY AS THE GRAND MEAN AND S2NETMDELAY AS THE GRAND
  SUM.OF.SQUARES OF NMEANDELAY
TALLY MNETVDELAY AS THE GRAND MEAN AND S2NETVDELAY AS THE GRAND
 SUM.OF.SQUARES OF NVARDELAY
'' VERIF
'' TALLY MEANARRV AS THE PERBATCH MEAN OF INTARRVFL
'' TALLY MEANSERV AS THE PERBATCH MEAN OF SERVFL


DEFINE .SOURCE TO MEAN 1
DEFINE .DEST TO MEAN 2
DEFINE .FALSE TO MEAN 0
DEFINE .TRUE TO MEAN 1
DEFINE .PASSIVE TO MEAN 0
DEFINE .ACTIVE TO MEAN 1
DEFINE .SUSPENDED TO MEAN 2
DEFINE .INTERRUPTED TO MEAN 3
DEFINE .MAXNODES TO MEAN 50
DEFINE .MAXLINKS TO MEAN 2 * (.MAXNODES - 1)
DEFINE .MAXENV TO MEAN 10
DEFINE .SECONDS TO MEAN DAYS
DEFINE .NONE TO MEAN 0
'' FOLLOWING DEFINES USED TO MAKE STREAM CHANGES EASIER
DEFINE STREAM1 TO MEAN 1
DEFINE STREAM2 TO MEAN 2
DEFINE STREAM3 TO MEAN 3
DEFINE STREAM4 TO MEAN 4


END '' PREAMBLE


MAIN

  '' READ SIMULATION AND NETWORK PARAMETERS
  CALL INPSIMPARAM YIELDING NOBATCH, BATCHSIZE, NOTRANSOBS, AND TVAL
  PRINT 1 LINE WITH NOBATCH, BATCHSIZE, AND NOTRANSOBS THUS
NOBATCH= *** BATCHSIZE= **** NOTRANSOBS= ***
  '' SPACE FOR GLOBALS ARRVRATE, ZEROARRV, NODEIDS, AND CAPACITIES
  '' RESERVED IN INPNETPARAM.
  '' NONODE AND NOLINK ARE GLOBAL EVERYWHERE EXCEPT INPNETPARAM
  CALL INPNETPARAM YIELDING NONODE, NOLINK, ENVGEN(*,*), ENVPROB(*),
    AND NOENV
  '' VERIF
  '' READ IAT AND ERT
  '' INTERARRIVAL TIME AND ENVIRONMENT STATE RESIDENCE TIME

  '' RESERVE SPACE FOR ARRAYS OF POINTERS TO PROCESSES
  RESERVE NODES AS NONODE
  RESERVE LINKS AS NOLINK
  RESERVE ARRVPROCS AS NONODE BY NONODE

  '' READ ROUTING TABLES AND GENERATE NODES
  CALL RTBUILD

  '' RESERVE SPACE FOR AND INITIALIZE OVERALL ACCUMULATION VARIABLES
  RESERVE SLKMQLEN, S2LKMQLEN, SLKVQLEN, AND S2LKVQLEN AS NOLINK
  RESERVE SLKMXQLEN AS NOLINK
  RESERVE SLKMDELAY, S2LKMDELAY, SLKVDELAY, AND S2LKVDELAY AS NOLINK
  FOR LINKINX = 1 TO NOLINK
    DO
      LET SLKMQLEN(LINKINX) = 0.0
      LET S2LKMQLEN(LINKINX) = 0.0
      LET SLKVQLEN(LINKINX) = 0.0
      LET S2LKVQLEN(LINKINX) = 0.0
      LET SLKMXQLEN(LINKINX) = 0.0
      LET SLKMDELAY(LINKINX) = 0.0
      LET S2LKMDELAY(LINKINX) = 0.0
      LET SLKVDELAY(LINKINX) = 0.0
      LET S2LKVDELAY(LINKINX) = 0.0
    LOOP
  RESET THE GRAND TOTALS OF NMEANDELAY
  RESET THE GRAND TOTALS OF NVARDELAY
  LET SNETNOMSG = 0.0

  LET BATCHINX = 0
  LET SIMDONE = .FALSE

  LET NETNOMSG = 0
  LET MSGNO = 0
  '' CREATE AND ACTIVATE ENVIRONMENT.
  ACTIVATE AN ENVIRONMENT GIVING ENVGEN(*, *), ENVPROB(*), NOENV,
    0, AND 0.0 NOW

  '' CREATE AND ACTIVATE LINKS.
  '' LINKS CONTAIN PER BATCH ACCUMULATION VARIABLES.
  CALL GENNET
  FOR NODEINX1 = 1 TO NONODE
    FOR NODEINX2 = 1 TO NONODE
      DO
        IF (ZEROARRV(NODEINX1, NODEINX2) = .FALSE)
            ACTIVATE A MSGGEN GIVING NODEINX1 AND NODEINX2
              NOW
            LET ARRVPROCS(NODEINX1, NODEINX2) = MSGGEN
        ELSE
          LET ARRVPROCS(NODEINX1, NODEINX2) = .NONE
        ALWAYS
      LOOP  '' FOR

  '' START, WAIT, AND THEN WRITE STATISTICS
  START SIMULATION

  CALL FINALANALYSIS(NOBATCH)
END


PROCESS
MSGGEN

  DEFINE INTARRVTIME AS.REAL
  DEFINE ARRVTIME, NARRVRATE AS.REAL
  DEFINE ARRVDONE AS.INT  '' USED AS BOOLEAN
  '' VERIF
  '' DEFINE LASTARRV AS.REAL

  '' VERIF
  '' LET LASTARRV = TIME.V
  WHILE (SIMDONE = .FALSE)
    DO
      IF (ARRVRATE(THISSTATE(ENVIRONMENT), MGSOURCE, MGDEST) ^= 0.0)
          '' TRACE DIAGS
          '' PRINT 1 LINE WITH TIME.V, MSGNO+1, MGSOURCE AND MGDEST
            '' THUS
'' (********.**) MSG ACT: MSG ID: **** SRC: ** DEST: **
          '' VERIF
          '' LET LASTARRV = TIME.V
          LET MSGNO = MSGNO + 1
          ACTIVATE A MESSAGE GIVING MSGNO, MGSOURCE, MGDEST,
            0.0, 0.0, AND 0.0 NOW
      ALWAYS

      LET ARRVDONE = .FALSE
      WHILE (ARRVDONE = .FALSE)
        DO
          '' INTARRVTIME = IAT
          LET NARRVRATE = ARRVRATE(THISSTATE(ENVIRONMENT), MGSOURCE,
                          MGDEST)
          IF (NARRVRATE ^= 0.0)
            LET INTARRVTIME = EXPONENTIAL.F(1.0 / NARRVRATE,
                              STREAM3)
            LET ARRVTIME = TIME.V + INTARRVTIME
            IF (ARRVTIME <= TRANSTIME(ENVIRONMENT))
              LET ARRVDONE = .TRUE
            ELSE
              LET ARRVDONE = .FALSE
            ALWAYS
          ELSE
            LET ARRVDONE = .FALSE
          ALWAYS
          '' TRACE DIAGS
          '' IF (ARRVDONE = .FALSE)
            '' PRINT 2 LINES WITH TIME.V, MGSOURCE, MGDEST
              '' ARRVTIME, AND TRANSTIME(ENVIRONMENT) THUS
'' (********.**) PART ARRV: MSG ID: (NA) SRC: ** DEST: **
''                ARRV TIME: ******* *.** TRAN TIME: ********.**
          '' ALWAYS
          IF (ARRVDONE = .TRUE)
            WAIT INTARRVTIME .SECONDS
          ELSE
            '' WAIT IN ENVIRONMENT TRANSITION QUEUE
            FILE MSGGEN IN GTRANSQ(ENVIRONMENT)
            SUSPEND
          ALWAYS
        LOOP '' WHILE 2

      '' VERIF
      '' LET INTARRVFL = TIME.V - LASTARRV
    LOOP '' WHILE 1

  '' SUSPEND. OTHERWISE SIMSCRIPT WILL DESTROY THIS PROCESS.
  SUSPEND

  RETURN
END '' PROCESS MSGGEN


PROCESS
MESSAGE

  DEFINE THISNODE, NEXTNODE AS.INT
  DEFINE ALINK AS.INT '' USED AS REF
  DEFINE THISLINK AS.INT
  DEFINE MSGLEN AS.REAL
  DEFINE ROUTETAB AS A INTEGER 1-DIM ARRAY

  LET NETATIME = TIME.V
  LET THISNODE = MSGSOURCE
  '' TRACE DIAGS
  '' PRINT 1 LINE WITH TIME.V, THISNODE, AND MSGID THUS
'' (********.**) MSG ARRV: NODE: ** MSG ID: ****
  WHILE (THISNODE ^= MSGDEST)
    DO
      LET ROUTETAB(*) = NROUTE(NODES(THISNODE))
      LET NEXTNODE = ROUTETAB(MSGDEST)
      LET ROUTETAB(*) = LROUTE(NODES(THISNODE))
      LET THISLINK = ROUTETAB(NEXTNODE)
      '' TRACE DIAGS
      '' PRINT 1 LINE WITH TIME.V, THISLINK, AND MSGID THUS
'' (********.**) MSG ARRV: LINK: ** MSG ID: ****
      LET LINKATIME = TIME.V
      LET ALINK = LINKS(THISLINK)

      FILE MESSAGE IN QUEUE(ALINK)
      IF ((STA.A(ALINK) = .SUSPENDED) AND (TRANSWAIT(ALINK) = .FALSE))
        REACTIVATE THE TLINK CALLED ALINK NOW
      ALWAYS
      SUSPEND

      LET THISNODE = NEXTNODE
      '' TRACE DIAGS
      '' PRINT 1 LINE WITH TIME.V, THISNODE, AND MSGID THUS
'' (********.**) MSG ARRV: NODE: ** MSG ID: ****

      LET ROUTETAB(*) = 0
    LOOP '' WHILE

  LET NETDTIME = TIME.V
  LET NETDELAY = NETDTIME - NETATIME
  '' TRACE DIAGS
  '' PRINT 1 LINE WITH TIME.V, NETDELAY, AND MSGID THUS
'' (********.**) NET DEP: DELAY: *******.***** MSG ID: ****

  '' PILOT
  '' IF (NETNOMSG = BATCHSIZE)
    '' LET BATCHINX = BATCHINX + 1
    '' CALL REPORTAVG
  '' ALWAYS

  '' INCREMENT NETNOMSG. MAY CAUSE (THROUGH LEFT MONITORING) DELETION
  '' OF TRANSIENT PERIOD OBSERVATIONS OR BATCH RESET.
  LET NETNOMSG = NETNOMSG + 1

  RETURN
END '' PROCESS MESSAGE


   '' THIS VERSION OF TLINK DOES NOT ALLOW SERVER INTERRUPTIONS.
PROCESS
TLINK

  DEFINE AMSG AS.INT '' USED AS REF
  DEFINE MSGLEN AS.REAL
  DEFINE SERVTIME, DEPTIME AS.REAL
  DEFINE SERVDONE AS.INT '' USED AS BOOLEAN
  '' VERIF
  '' DEFINE LASTSTART AS.REAL

  WHILE (SIMDONE = .FALSE)
    DO
      IF (QUEUE IS EMPTY)
        SUSPEND
      ALWAYS

      LET AMSG = F.QUEUE
      LET SERVDONE = .FALSE
      '' VERIF
      '' LET LASTSTART = TIME.V
      WHILE (SERVDONE = .FALSE)
        DO
          '' LET MSGLEN = MEANMLEN
          LET MSGLEN = EXPONENTIAL.F(MEANMLEN, STREAM4)
          LET SERVTIME = MSGLEN / CAPACITY
          LET DEPTIME = TIME.V + SERVTIME
          IF (DEPTIME <= TRANSTIME(ENVIRONMENT))
            LET SERVDONE = .TRUE
          ELSE
            LET SERVDONE = .FALSE
          ALWAYS
          IF (SERVDONE = .TRUE)
            WORK SERVTIME .SECONDS
          ELSE
            LET TRANSWAIT = .TRUE
            FILE TLINK IN LTRANSQ(ENVIRONMENT)
          SUSPEND
          LET TRANSWAIT = .FALSE
        ALWAYS
      LOOP '' WHILE 2
    LET LINKDELAY = TIME.V - LINKATIME(AMSG)

    '' TRACE DIAGS
    '' PRINT 1 LINE WITH TIME.V, LKID, MSGID(AMSG), AND
      '' LASTSTART THUS
'' (********.**) SERV COMP: LINK: ** MSG ID: **** START: ********.**

    '' VERIF
    '' LET SERVFL = TIME.V - LASTSTART

    REMOVE FIRST MESSAGE FROM QUEUE
    REACTIVATE THIS MESSAGE NOW
  LOOP  '' WHILE 1

  '' SUSPEND. OTHERWISE SIMSCRIPT WILL DESTROY THIS PROCESS. ERROR
  '' IF QUEUE IS NOT EMPTY. SET TRANSWAIT TO TRUE SO MESSAGES WILL
  '' NOT TRY TO ACTIVATE THIS PROCESS.
  LET TRANSWAIT = .TRUE
  SUSPEND

  RETURN
END  '' PROCESS TLINK


PROCESS
ENVIRONMENT

  DEFINE NEXTSTATE AS.INT
  DEFINE RESTIME AS.REAL
  DEFINE STATECDF AS A REAL 1-DIM ARRAY
  DEFINE CDFVAL AS.REAL
  DEFINE ENVINX AS.INT
  DEFINE LOCGEN AS A REAL 2-DIM ARRAY
  DEFINE LOCSPROB AS A REAL 1-DIM ARRAY

  LET LOCGEN(*,*) = GENERATOR
  LET LOCSPROB(*) = STATPROB

  '' CREATE ARRAY OF CUMULATIVE DISTRIBUTION FUNCTION VALUES FOR
  '' ENVIRONMENT STATIONARY PROBABITIES
  RESERVE STATECDF AS NOSTATE
  LET CDFVAL = 0.0
  FOR ENVINX = 1 TO NOSTATE - 1
    DO
      LET CDFVAL = CDFVAL + LOCSPROB(ENVINX)
      LET STATECDF(ENVINX) = CDFVAL
    LOOP  '' FOR
  '' LAST CDF VALUE MUST BE EXACTLY 1.0
  LET STATECDF(NOSTATE) = 1.0

  '' PICK INITIAL STATE. THEN CONTINUE WITH STATE TRANSITIONS UNTIL
  '' STOPTIME
  LET THISSTATE = INITSTATE(STATECDF(*), NOSTATE)
  '' TRACE DIAGS
  '' PRINT 1 LINE WITH TIME.V AND THISSTATE THUS
'' (********.**) ENV ACT: INIT STATE: **
  WHILE (SIMDONE = .FALSE)
    DO
      CALL TRANSITION(THISSTATE, LOCGEN(*,*), NOSTATE) YIELDING
        NEXTSTATE, RESTIME, AND TRANSTIME
      WAIT RESTIME .SECONDS
      LET THISSTATE = NEXTSTATE
      '' REACTIVATE ALL ARRIVAL AND SERVICE PROCESSES IN THE
      '' TRANSITION WAIT QUEUES
      WHILE (LTRANSQ IS NOT EMPTY)
        DO
          REMOVE FIRST TLINK FROM LTRANSQ
          REACTIVATE THIS TLINK NOW
        LOOP  '' WHILE 2
      WHILE (GTRANSQ IS NOT EMPTY)
        DO
          REMOVE FIRST MSGGEN FROM GTRANSQ
          REACTIVATE THIS MSGGEN NOW
        LOOP  '' WHILE 2
      '' TRACE DIAGS
      '' PRINT 1 LINE WITH TIME.V AND THISSTATE THUS
'' (********.**) ENV TRANS: NEW STATE: **
    LOOP  '' WHILE 1

  LET LOCGEN(*, *) = 0
  LET LOCSPROB(*) = 0
  RELEASE STATECDF

  '' SUSPEND. OTHERWISE SIMSCRIPT WILL DESTROY THIS PROCESS. ERROR
  '' IF QUEUES ARE NOT EMPTY.
  SUSPEND

  RETURN
END  '' PROCESS ENVIRONMENT


ROUTINE
BATCHRESET

  DEFINE LINKINX AS.INT
  DEFINE NODEINX1 AND NODEINX2 AS.INT

  FOR LINKINX = 1 TO NOLINK
    DO
      RESET THE PERBATCH TOTALS OF N.QUEUE(LINKS(LINKINX))
      RESET THE PERBATCH TOTALS OF LINKDELAY(LINKS(LINKINX))
    LOOP  '' FOR

  '' VERIF
  '' FOR LINKINX = 1 TO NOLINK
    '' RESET THE PERBATCH TOTALS OF SERVFL(LINKS(LINKINX))
  '' FOR NODEINX1 = 1 TO NONODE
    '' FOR NODEINX2 = 1 TO NONODE
      '' DO
        '' IF (ZEROARRV(NODEINX1, NODEINX2) = .FALSE)
          '' RESET THE PERBATCH TOTALS OF
            '' INTARRVFL(ARRVPROCS(NODEINX1, NODEINX2))
        '' ALWAYS
      '' LOOP

  RESET THE PERBATCH TOTALS OF NETDELAY
END  '' PROCESS RESETBATCH


'' THIS ROUTINE ASSUMES THAT DISTFUNCT(MAXVAL) = 1.0 (EXACTLY).
ROUTINE
DISCRETE(DISTFUNCT, MAXVAL, STREAM)
  DEFINE DISTFUNCT AS A REAL 1-DIM ARRAY
  DEFINE MAXVAL AS.INT
  DEFINE STREAM AS.INT

  DEFINE VARIATE AS.REAL
  DEFINE VALINX AS.INT
  DEFINE DONE AS.INT  '' USED AS BOOLEAN

  LET VARIATE = UNIFORM.F(0.0, 1.0, STREAM)
  LET DONE = .FALSE
  LET VALINX = 1
  WHILE (DONE = .FALSE)
    DO
      IF (VARIATE <= DISTFUNCT(VALINX))
        LET DONE = .TRUE
      ELSE
        LET VALINX = VALINX + 1
      ALWAYS
    LOOP  '' ALWAYS

  RETURN( VALINX )
END  '' DISCRETE


ROUTINE
INITSTATE(STATECDF, NOSTATE)
  DEFINE STATECDF AS A REAL 1-DIM ARRAY
  DEFINE NOSTATE AS.INT

  '' CHOOSE INITIAL STATE ACCORDING TO CDF OF STATIONARY DISTRIBUTION
  RETURN( DISCRETE(STATECDF(*), NOSTATE, STREAM1) )
END '' INITSTATE

ROUTINE
TRANSITION(THISSTATE, GENERATOR, NOSTATE) YIELDING NEXTSTATE, RESTIME,
       AND TRANSTIME
  DEFINE THISSTATE AS.INT
  DEFINE GENERATOR AS A REAL 2-DIM ARRAY
  DEFINE NOSTATE AS.INT
  DEFINE NEXTSTATE AS.INT '' OUTPUT
  DEFINE RESTIME AS.REAL ''OUTPUT
  DEFINE TRANSTIME AS.REAL '' OUTPUT

  DEFINE ENVINX AS.INT
  DEFINE TEMPCDF AS A REAL 1-DIM ARRAY
  DEFINE LAMBDA AND CDFVAL AS.REAL

  '' LAMBDA IS THE INVERSE OF THE MEAN TIME IN THE CURRENT STATE
  LET LAMBDA = -GENERATOR(THISSTATE, THISSTATE)
  '' CREATE CUMULATIVE DISTRIBUTION FUNCTION FOR NEXT STATE
  '' PROBABILITIES FROM A ROW OF THE ENVIRONMENT GENERATOR
  RESERVE TEMPCDF AS NOSTATE
  LET CDFVAL = 0.0
  FOR ENVINX = 1 TO NOSTATE - 1
    DO
      IF (ENVINX ^= THISSTATE)
        LET CDFVAL = CDFVAL + GENERATOR(THISSTATE, ENVINX) / LAMBDA
      ALWAYS
      LET TEMPCDF(ENVINX) = CDFVAL
    LOOP '' FOR
  '' LAST CDF VALUE MUST BE EXACTLY 1.0
  LET TEMPCDF(NOSTATE) = 1.0

  '' CHOOSE NEXT STATE AND TIME UNTIL NEXT TRANSTION
  LET NEXTSTATE = DISCRETE(TEMPCDF(*), NOSTATE, STREAM1)
  '' RESTIME = ERT
  LET RESTIME = EXPONENTIAL.F(1.0 / LAMBDA, STREAM2)
  LET TRANSTIME = TIME.V + RESTIME

  RELEASE TEMPCDF
  RETURN
END '' TRANSITION


ROUTINE
RTBUILD

  DEFINE THISNODE, NEXTNODE, AND DESTNODE AS.INT
  DEFINE NODEINX1 AND NODEINX2 AS.INT
  DEFINE TEMPNR AND TEMPLR AS INTEGER 1-DIM ARRAYS

  FOR NODEINX1 = 1 TO NONODE
    DO
      READ THISNODE
      CREATE A NODE
      LET NDID(NODE) = THISNODE

      RESERVE TEMPNR AS NONODE
      RESERVE TEMPLR AS NONODE
      FOR NODEINX2 = 1 TO NONODE - 1
        DO
          READ DESTNODE
          READ TEMPNR(DESTNODE) '' = NEXTNODE
        LOOP '' FOR 2
      LET TEMPNR(THISNODE) = -1
      FOR NODEINX2 = 1 TO NONODE - 1
        DO
          READ NEXTNODE
          '' LINKID IN INPUT MUST BE -1 FOR TWO UNCONNECTED
          '' NODES
          READ TEMPLR(NEXTNODE) '' = LINKID
        LOOP '' FOR 3
      LET TEMPLR(THISNODE) = -1

      LET NROUTE(NODE) = TEMPNR(*)
      LET LROUTE(NODE) = TEMPLR(*)
      LET NODES(THISNODE) = NODE
      '' THESE ASSIGNMENTS TO ZERO ARE NECESSARY SO TEMPNR AND TEMPLR
      '' CAN BE RESERVED AGAIN
      LET TEMPNR(*) = 0
      LET TEMPLR(*) = 0
    LOOP  '' FOR 1

  RETURN
END '' RTBUILD


ROUTINE
GENNET

  DEFINE LINKINX AS.INT

  FOR LINKINX = 1 TO NOLINK
    DO
      ACTIVATE A TLINK GIVING LINKINX, CAPACITIES(LINKINX),
        NODEIDS(LINKINX, .SOURCE), NODEIDS(LINKINX, .DEST), .FALSE,
        AND 0.0 NOW
      LET LINKS(LINKINX) = TLINK
    LOOP  '' FOR

  RETURN
END '' GENNET


ROUTINE
INPNETPARAM YIELDING NONODE, NOLINK, GENERATOR, STATPROB, AND NOENV
  DEFINE NONODE AS.INT  '' OUTPUT
  DEFINE NOLINK AS.INT  '' OUTPUT
  DEFINE GENERATOR AS A REAL 2-DIM ARRAY  '' OUTPUT
  DEFINE STATPROB AS A REAL 1-DIM ARRAY  '' OUTPUT
  DEFINE NOENV AS.INT  '' OUTPUT

  DEFINE NODEINX1 AND NODEINX2 AS.INT
  DEFINE LINKINX AND LINKID AS.INT
  DEFINE ENVINX1 AND ENVINX2 AS.INT
  DEFINE NARRVRATE AS.REAL

  READ NONODE
  READ NOENV
  LET NOLINK = 2 * (NONODE - 1)

  RESERVE ARRVRATE AS NOENV BY NONODE BY NONODE
  RESERVE ZEROARRV AS NONODE BY NONODE
  RESERVE NODEIDS AS NOLINK BY 2
  RESERVE CAPACITIES AS NOLINK

  FOR ENVINX1 = 1 TO NOENV
    FOR NODEINX1 = 1 TO NONODE
      FOR NODEINX2 = 1 TO NONODE
        READ ARRVRATE(ENVINX1, NODEINX1, NODEINX2)

  '' IDENTIFY NODE PAIRS FOR WHICH ARRIVAL RATES ARE ZERO FOR ALL
  '' CONFIGURATIONS. (NO ARRIVAL PROCESSES NEEDED FOR THESE PAIRS).
  FOR NODEINX1 = 1 TO NONODE
    FOR NODEINX2 = 1 TO NONODE
      DO
        LET NARRVRATE = 0.0
        LET ENVINX1 = 0
        WHILE ((NARRVRATE = 0.0) AND (ENVINX1 < NOENV))
          DO
            LET ENVINX1 = ENVINX1 + 1
            LET NARRVRATE = ARRVRATE(ENVINX1, NODEINX1, NODEINX2)
          LOOP  '' WHILE
        IF (NARRVRATE = 0.0)
          LET ZEROARRV(NODEINX1, NODEINX2) = .TRUE
        ELSE
          LET ZEROARRV(NODEINX1, NODEINX2) = .FALSE
        ALWAYS
      LOOP  '' FOR

  FOR LINKINX = 1 TO NOLINK
    DO
      READ LINKID
      READ NODEIDS(LINKID, .SOURCE), NODEIDS(LINKID, .DEST), AND
        CAPACITIES(LINKID)
    LOOP '' FOR
  READ MEANMLEN

  RESERVE GENERATOR AS NOENV BY NOENV
  RESERVE STATPROB AS NOENV
  FOR ENVINX1 = 1 TO NOENV
    FOR ENVINX2 = 1 TO NOENV
      READ GENERATOR(ENVINX1, ENVINX2)
  FOR ENVINX1 = 1 TO NOENV
    READ STATPROB(ENVINX1)

  RETURN
END '' INPNETPARAM


ROUTINE
INPSIMPARAM YIELDING NOBATCH, BATCHSIZE, NOTRANSOBS, AND TVAL
  DEFINE NOBATCH AS.INT '' OUTPUT
  DEFINE BATCHSIZE AS.INT '' OUTPUT
  DEFINE NOTRANSOBS AS.INT '' OUTPUT
  DEFINE TVAL AS.REAL '' OUTPUT

  READ NOBATCH
  READ NOTRANSOBS AND BATCHSIZE
  READ TVAL

  RETURN
END '' INPSIMPARAM


ROUTINE
REPORTBATCH

  DEFINE LINKINX AS.INT
  DEFINE MEANQLEN, VARQLEN, MEANDELAY, AND VARDELAY AS.REAL
  DEFINE ALINK AS.INT '' USED AS REF

  '' VERIF VAR
  '' DEFINE NODEINX1 AND NODEINX2 AS.INT
  '' DEFINE AMSGGEN AS.INT

  SKIP 3 LINES
  PRINT 1 LINE WITH BATCHINX THUS
STATISTICS FOR BATCH ***
  LET SSTSIMSEC = TIME.V - STARTTIME
  PRINT 1 LINE WITH SSTSIMSEC THUS
RUN LENGTH: *******.**** SEC
  SKIP 3 LINES
  PRINT 3 LINES THUS
            QUEUE LENGTH                DELAY       NUMBER OF
LINK    MEAN     VARIANCE   MAXIMUM    MEAN     VARIANCE   MESSAGES
--------------------------------------------------------------------------------
  FOR LINKINX = 1 TO NOLINK
    DO
      LET ALINK = LINKS(LINKINX)
      LET MEANQLEN = LKMQLEN(ALINK)
      LET VARQLEN = LKVQLEN(ALINK)
      LET MEANDELAY = LKMDELAY(ALINK)
      LET VARDELAY = LKVDELAY(ALINK)
      PRINT 1 LINE WITH LINKINX, MEANQLEN, VARQLEN, LKMAXQLEN(ALINK),
        MEANDELAY, VARDELAY AND LKNOMSG(ALINK) THUS
***  ******.****  ******.****   ****   ******.****  ******.****   ******

      '' UPDATE OVERALL ACCUMULATION VARIABLES FOR LINK.
      LET SLKMQLEN(LINKINX) = SLKMQLEN(LINKINX) + MEANQLEN
      LET S2LKMQLEN(LINKINX) = S2LKMQLEN(LINKINX) + (MEANQLEN ** 2.0)
      LET SLKVQLEN(LINKINX) = SLKVQLEN(LINKINX) + VARQLEN
      LET S2LKVQLEN(LINKINX) = S2LKVQLEN(LINKINX) + (VARQLEN ** 2.0)
      LET SLKMXQLEN(LINKINX) = SLKMXQLEN(LINKINX) + LKMAXQLEN(ALINK)
      LET SLKMDELAY(LINKINX) = SLKMDELAY(LINKINX) + MEANDELAY
      LET S2LKMDELAY(LINKINX) = S2LKMDELAY(LINKINX)
                      + (MEANDELAY ** 2.0)
      LET SLKVDELAY(LINKINX) = SLKVDELAY(LINKINX) + VARDELAY
      LET S2LKVDELAY(LINKINX) = S2LKVDELAY(LINKINX)
                      + (VARDELAY ** 2.0)
    LOOP '' FOR

  '' VERIF
```

```
   '' SKIP 1 LINE
   '' PRINT 3 LINES THUS
'' SOURCE  DEST  MEAN INTERARRIVAL
'' NODE    NODE        TIME
'' --------------------------------
   '' FOR NODEINX1 = 1 TO NONODE
     '' FOR NODEINX2 = 1 TO NONODE
       '' DO
         '' IF (ZEROARRV(NODEINX1, NODEINX2) = .FALSE)
           '' LET AMSGGEN = ARRVPROCS(NODEINX1, NODEINX2)
           '' PRINT 1 LINE WITH NODEINX1, NODEINX2, AND
             '' MEANARRV(AMSGGEN) THUS
  '' ***    ***    ******.****
         '' ALWAYS
       ''  LOOP

   '' SKIP 1 LINE
   '' PRINT 3 LINES THUS
     '' MEAN SERVICE
'' LINK     TIME
'' -----------------
   '' FOR LINKINX = 1 TO NOLINK
     '' DO
       '' LET ALINK = LINKS(LINKINX)
       '' PRINT 1 LINE WITH LINKINX AND MEANSERV(ALINK) THUS
 '' ***   ******.****
     '' LOOP

   SKIP 1 LINE
   LET NMEANDELAY = NETMDELAY
   LET NVARDELAY = (NETS2DELAY - (NETNOMSG * (NETMDELAY ** 2.0)))
             / (NETNOMSG - 1)
   LET SNETNOMSG = SNETNOMSG + NETNOMSG
   PRINT 3 LINES WITH NMEANDELAY AND NVARDELAY THUS
NETWORK DELAY:
    AVERAGE: *******.****
    VARIANCE: *******.****

   '' DIAGS
   '' FOR LINKINX = 1 TO NOLINK
     '' DO
       '' PRINT 1 LINE WITH LINKINX AND N.QUEUE(LINKS(LINKINX)) THUS
'' LINK: *** NUMBER IN QUEUE: ********
     '' LOOP

   '' PER BATCH ACCUMULATION VARIABLES RESET IN BATCHRESET BY CALL FROM
   '' MONITOR ROUTINE FOR NETNOMSG.

  RETURN
END '' REPORTBATCH


'' PILOT
ROUTINE
REPORTAVG

  DEFINE LINKINX AS.INT
  DEFINE GMEANQLEN AND GMEANDELAY AS.REAL
  DEFINE ALINK AS.INT '' USED AS REF

  LET SNETNOMSG = SNETNOMSG + NETNOMSG
  SKIP 1 LINE
  PRINT 1 LINE WITH TIME.V AND SNETNOMSG THUS
(********.****) STATISTICS AT NET MSG COUNT: ********
  SKIP 1 LINE
  PRINT 2 LINES THUS
LINK     MEAN QUEUE LENGTH   MEAN DELAY
-----------------------------------------
  FOR LINKINX = 1 TO NOLINK
    DO
      LET ALINK = LINKS(LINKINX)
      '' UPDATE OVERALL ACCUMULATION VARIABLES FOR LINK.
      LET SLKMQLEN(LINKINX) = SLKMQLEN(LINKINX) + LKMQLEN(ALINK)
      LET SLKMDELAY(LINKINX) = SLKMDELAY(LINKINX) + LKMDELAY(ALINK)
      LET GMEANQLEN = SLKMQLEN(LINKINX) / BATCHINX
      LET GMEANDELAY = SLKMDELAY(LINKINX) / BATCHINX
      PRINT 1 LINE WITH LINKINX, GMEANQLEN, AND GMEANDELAY THUS
***        ******.****      ******.****
```

```
    LOOP

  SKIP 1 LINE
  LET NMEANDELAY = NETMDELAY
  PRINT 1 LINE WITH MNETMDELAY THUS
MEAN NETWORK DELAY: *******.****

  '' RESETS ARE NECESSARY SINCE DELETEOBS IS NOT CALLED
  FOR LINKINX = 1 TO NOLINK
    DO
      ALINK = LINKS(LINKINX)
      RESET THE PERBATCH TOTALS OF N.QUEUE(ALINK)
      RESET THE PERBATCH TOTALS OF LINKDELAY(ALINK)
    LOOP '' FOR
  RESET THE PERBATCH TOTALS OF NETDELAY

  RETURN
END '' REPORTAVG


ROUTINE
FINALANALYSIS(NOBATCH)
  DEFINE NOBATCH AS.INT

  DEFINE LINKINX AS.INT
  DEFINE MEANQLEN, VARQLEN, MAXQLEN, MEANDELAY, AND VARDELAY AS.REAL
  DEFINE MEANNOMSG AS.REAL '' FOR NETWORK
  DEFINE MSAMPMEAN, VSAMPMEAN, SAMPVAR AS.REAL
  DEFINE MHALFLEN AND VHALFLEN AS.REAL

  SKIP 2 LINES
  PRINT 1 LINE WITH NOBATCH THUS
STATISTICS OVER ALL (***) BATCHES
  SKIP 1 LINE
  PRINT 3 LINES THUS
           QUEUE LENGTH                      DELAY
LINK    MEAN    VARIANCE   MAXIMUM       MEAN      VARIANCE
-----------------------------------------------------------------------------------
  FOR LINKINX = 1 TO NOLINK
    DO
      LET MEANQLEN = SLKMQLEN(LINKINX) / NOBATCH
      LET VARQLEN = SLKVQLEN(LINKINX) / NOBATCH
      LET MAXQLEN = SLKMXQLEN(LINKINX) / NOBATCH
      LET MEANDELAY = SLKMDELAY(LINKINX) / NOBATCH
      LET VARDELAY = SLKVDELAY(LINKINX) / NOBATCH
      PRINT 1 LINE WITH LINKINX, MEANQLEN, VARQLEN, MAXQLEN,
        MEANDELAY, AND VARDELAY THUS
*** ******.**** ******.**** ******.****  ******.**** ******.****
    LOOP '' FOR

  SKIP 1 LINE
  PRINT 1 LINE THUS
QUEUE LENGTH CONFIDENCE INTERVALS
  SKIP 1 LINE
  PRINT 3 LINES THUS
             MEAN                 VARIANCE
LINK  LEFT ENDPOINT  RIGHT ENDPOINT  LEFT ENDPOINT  RIGHT ENDPOINT
----------------------------------------------------------------------------
  FOR LINKINX = 1 TO NOLINK
    DO
      LET MSAMPMEAN = SLKMQLEN(LINKINX) / NOBATCH
      LET SAMPVAR = (S2LKMQLEN(LINKINX) - NOBATCH
                             * (MSAMPMEAN ** 2.0))
               / (NOBATCH - 1)
      LET MHALFLEN = TVAL * ((SAMPVAR / NOBATCH) ** 0.5)
      LET VSAMPMEAN = SLKVQLEN(LINKINX) / NOBATCH
      LET SAMPVAR = (S2LKVQLEN(LINKINX) - NOBATCH
                            * (VSAMPMEAN ** 2.0))
              / (NOBATCH - 1)
      LET VHALFLEN = TVAL * ((SAMPVAR / NOBATCH) ** 0.5)
      PRINT 1 LINE WITH LINKINX, MSAMPMEAN - MHALFLEN,
        MSAMPMEAN + MHALFLEN, VSAMPMEAN - VHALFLEN,
        VSAMPMEAN + VHALFLEN THUS
***  ******.****   ******.****    ******.****   ******.****
    LOOP '' FOR

  SKIP 1 LINE
  PRINT 1 LINE THUS
```

```
DELAY CONFIDENCE INTERVALS
  SKIP 1 LINE
  PRINT 3 LINES THUS
           MEAN                VARIANCE
LINK  LEFT ENDPOINT  RIGHT ENDPOINT  LEFT ENDPOINT  RIGHT ENDPOINT
--------------------------------------------------------------------
  FOR LINKINX = 1 TO NOLINK
    DO
      LET MSAMPMEAN = SLKMDELAY(LINKINX) / NOBATCH
      LET SAMPVAR = (S2LKMDELAY(LINKINX) - NOBATCH
                          * (MSAMPMEAN ** 2.0))
              / (NOBATCH - 1)
      LET MHALFLEN = TVAL * ((SAMPVAR / NOBATCH) ** 0.5)
      LET VSAMPMEAN = SLKVDELAY(LINKINX) / NOBATCH
      LET SAMPVAR = (S2LKVDELAY(LINKINX) - NOBATCH
                          * (VSAMPMEAN ** 2.0))
              / (NOBATCH - 1)
      LET VHALFLEN = TVAL * ((SAMPVAR / NOBATCH) ** 0.5)
      PRINT 1 LINE WITH LINKINX, MSAMPMEAN - MHALFLEN,
        MSAMPMEAN + MHALFLEN, VSAMPMEAN - VHALFLEN,
        VSAMPMEAN + VHALFLEN THUS
 ***  ******.****   ******.****    ******.****    ******.****
    LOOP '' FOR

  SKIP 1 LINE
  PRINT 1 LINE THUS
NETWORK DELAY:
  LET MSAMPMEAN = MNETMDELAY
  LET SAMPVAR = (S2NETMDELAY - NOBATCH * (MSAMPMEAN ** 2.0))
          / (NOBATCH - 1)
  LET MHALFLEN = TVAL * ((SAMPVAR / NOBATCH) ** 0.5)
  PRINT 2 LINES WITH MSAMPMEAN, MSAMPMEAN - MHALFLEN, AND
    MSAMPMEAN + MHALFLEN THUS
   MEAN: *******.****
   CONFIDENCE INTERVAL: (*******.****, *******.****)

  LET VSAMPMEAN = MNETVDELAY
  LET SAMPVAR = (S2NETVDELAY - NOBATCH * (VSAMPMEAN ** 2.0))
          / (NOBATCH - 1)
  LET VHALFLEN = TVAL * ((SAMPVAR / NOBATCH) ** 0.5)
  LET MEANNOMSG = SNETNOMSG / NOBATCH
  PRINT 2 LINES WITH VSAMPMEAN, VSAMPMEAN - VHALFLEN, AND
    VSAMPMEAN + VHALFLEN THUS
   VARIANCE: *******.****
   CONFIDENCE INTERVAL: (*******.****, *******.****)
  SKIP 1 LINE
  PRINT 1 LINE WITH MEANNOMSG THUS
MEAN NUMBER OF MESSAGES: *******.****
  SKIP 5 LINES

  RETURN
END


LEFT ROUTINE
NETNOMSG

 DEFINE SAVNOMSG AS.INT

  ENTER WITH SAVNOMSG
  IF (BATCHINX = 0)
    IF (NETNOMSG = NOTRANSOBS)
      LET TRANSSEC = TIME.V
      CALL BATCHRESET
      LET STARTTIME = TIME.V
      LET BATCHINX = 1
      MOVE FROM 0
    ELSE
      MOVE FROM SAVNOMSG
    ALWAYS
  ELSE
    IF (NETNOMSG = BATCHSIZE)
      CALL REPORTBATCH
      CALL BATCHRESET
      LET BATCHINX = BATCHINX + 1
      LET STARTTIME = TIME.V
      MOVE FROM 0
    ELSE
```

```
      MOVE FROM SAVNOMSG
    ALWAYS
  ALWAYS

  IF (BATCHINX > NOBATCH)
    LET SIMDONE = .TRUE
  ALWAYS

  RETURN
END
/*
//LOAD.SYSIN DD *
2
30000
30000
2.064
3
3
0.0 0.0 16.0
0.0 0.0 0.0
11.0 0.0 0.0
0.0 0.0 15.0
0.0 0.0 0.0
4.0 0.0 0.0
0.0 0.0 4.0
0.0 0.0 0.0
1.0 0.0 0.0
1 2 1 1302.39
2 1 2 2793.0375
3 3 2 1302.39
4 2 3 2793.0375
100.0
-16.0 9.0 7.0
9.0 -10.0 1.0
10.0 1.0 -11.0
0.3707 0.3605 0.2687
1
2 2
3 2
2 2
3 -1
2
1 1
3 3
1 1
3 4
3
1 2
2 2
1 -1
2 3
/*
//
```

APPENDIX H
DYNAMIC HIERARCHY MATRIX-GEOMETRIC ANALYSIS SOURCE CODE

Computation of the analytic results for comparison of the analytic and simulation models and comparison of the dynamic and static hierarchies is performed in the following stages:

(1) Computation of the (dynamic hierarchy) analytic results for comparison of the analytic and simulation models.

(2) Computation of the static hierarchy, analytic results for comparison of the dynamic and static hierarchies.

(3) Computation of the dynamic hierarchy, analytic results for comparison of the dynamic and static hierarchies.

To perform the computations, the following executable programs first are made from the C source code:

- *comprate*: Compute link arrival rate vectors from arrival rate matrices.

  To make: cc -o comprate comprate.c

- *mainmgp*: Compute matrix-geometric probabilities and means for a link.

  To make: cc -o mainmgp mainmgp.c mgprob.c gauselim.c mgmisc.c \
      probvect.c matinv.c matops.c input.c output.c

- *netdelay*: Compute dynamic hierarchy mean delay.

  To make: cc -o netdelay netdelay.c

- *statdel*: Compute square-root capacity assignments and static hierarchy mean delay.

  To make: cc -o statdel statdel.c -lm

Each of these programs reads from standard input and writes to standard output and is run, for example, as follows:

```
        comprate < netparam.inp > rates.out
```

where 'netparam.inp' and 'rates.out' are the input and output files, respectively.

Stages (1) and (3) are performed similarly according to the following steps (Note that the capacities required for stage (3) are taken from the output of stage (2)):

For each dynamic hierarchy topology/parameter set combination:

(a) Prepare a file according to file format 2 (Figure H.1) and run *comprate* with this file as input.

(b) For each link j, prepare a file according to file format 3 (Figure H.2) using the arrival rates for the link produced as output in (a) and service rate(s) μC_j. Run *mainmgp* with this file as input.

(c) Using the link arrival rates and network throughput produced as output in (a) and the mean link delay values produced as output in (b), prepare a file according to file format 4 (Figure H.3). Run *netdelay* with this file as input.

Step (c) yields the analytic estimate of mean network delay for this topology and parameter set.

Stage 2 is performed as follows:

For each static hierarchy topology/parameter set combination, prepare a file according to file format 5 (Figure H.4) and run *statdel* with this file as input. This run produces the analytic estimate of mean network delay for this static topology and parameter set.

<identifying label> { for documentation purposes }
<number of environment states> { M }
<environment state probabilities>
<number of nodes> { N }
<$\Gamma^{(1)}$> { traffic matrix for environment state 1 }
<$\Gamma^{(2)}$> { traffic matrix for environment state 2 }
.
.
.
<$\Gamma^{(M)}$> { traffic matrix for environment state M }
{ nodes are numbered from 0 to $N-1$ here }
0 1 { for nodes 0 and 1 }
<message route 0, 1> -1
0 2 { for nodes 0 and 2 }
<message route 0, 2> -1
.
.
.
<N> <$N-1$> { for nodes N and $N-1$ }
<message route N, $N-1$> -1
{ one for each j, k such that $j \neq k$ }

Figure H.1. File Format 2— Link Arrival Rate Computation

For each node pair j, k such that $j \neq k$, message route j, k is the list of links over which a jk-message is transmitted, arranged according to the order in which they are encountered by the message. Note that in file format 2, nodes are numbered from 0 to $N-1$ (instead of 1 to N as in the simulation input).

<identifying label> { for documentation purposes }
<number of environment states> { M }
<environment process generator>
<$\lambda_j^{(1)}$> <μC_j> { arrival and service rates under environment state 1 }
<$\lambda_j^{(2)}$> <μC_j> { arrival and service rates under environment state 2 }
.
.
.
<$\lambda_j^{(M)}$> <μC_j> { arrival and service rates under environment state M }

Figure H.2. File Format 3— Matrix-Geometric Mean Queue Length and Mean Link Delay Computation

<identifying label> { for documentation purposes }
<number of environment states> { M }
<number of links> { L }
$<\lambda_1^{(1)}>$ $<\lambda_2^{(1)}>$ . . . $<\lambda_L^{(1)}>$ $<\gamma^{(1)}>$
{ link arrival rates and network throughput under environment state 1 }
$<\lambda_1^{(2)}>$ $<\lambda_2^{(2)}>$ . . . $<\lambda_L^{(2)}>$ $<\gamma^{(2)}>$
{ link arrival rates and network throughput under environment state 2 }
.
.
.
$<\lambda_1^{(M)}>$ $<\lambda_2^{(M)}>$ . . . $<\lambda_L^{(M)}>$ $<\gamma^{(M)}>$
{ link arrival rates and network throughput under environment state M }
<link 1 delay>
<link 2 delay>
.
.
.
<link L delay>

Figure H.3. File Format 4— Dynamic Hierarchy Network Delay Computation

<identifying label> { for documentation purposes }
<number of links> { L }
$<\lambda_1>$ $<\lambda_2>$ . . . $<\lambda_L>$ $<\gamma>$
{ link arrival rates and network throughput }
$<\mu^{-1}>$ { mean message length }
<total cost (capacity)>

Figure H.4. File Format 5— Square-Root Capacity Assignment and Static Hierarchy Network Delay Computation

```
/*
** gauselim.h: (header file)
**      Gaussian elimination solution to N linear equations in N
**      unknowns. Uses row pivoting with scaled values. Solves
**      intentionally singular systems by solving a reduced size system
**      and then setting undetermined values to ARBVAL. The coeffients
**      corresponding to the reduced size system must be in the first
**      lastrow rows of amat. No solution is produced if an "unintentionally
**      singular" coefficient matrix (of possibly reduced size) is detected.
*/

#include "matgeom.h"

/*
** gauselim:
**      Main driver routine.
*/

extern FILE *inpfile, *outfile;
```

```
void
gauselim();


/*
** dabs:
**      Absolute value of long float
*/

double
dabs();


/*
** init:
**      Initialize scaling factor vector
*/

void
init();


/*
** triang:
**      Triangularize coefficient matrix (augmented with RHS vector)
**      using row pivoting with implicitly scaled row elements. If
**      lastrow < matdim, triangularize only to row lastrow. This allows
**      the partial solution of systems with intentionally singular
**      coefficient matrices.
*/

void
triang();


/*
** swaprow:
**      Swap two rows in the coefficient matrix and RHS vector
*/

void
swaprow();


/*
** swapd:
**      Swap two long float values
*/

void
swapd();


/*
** findpivt:
**      Find (index of) next pivot row based on implicitly scaled
**      elements
*/

void
findpivt();


/*
** sweep:
**      Sweep elements in rows from row after current pivot to last
*/

void
sweep();


/*
** backsub:
**      Solve for x values using backsubstitution with the triangularized
**      matrix and the RHS vector. Set undetermined values (at end of xval)
**      to ARBVAL.
*/
```

```
void
backsub();


/*
** input.h: (header file)
**      Input routines for dynamic hierarchy/ matrix geometric analysis
*/

#include "matgeom.h"

extern FILE *inpfile;

/*
** inpcmat:
**      Read augmented coefficient matrix (matrix and RHS) vector.
*/

void
inpcmat();


/*
** inpmat:
**      Read arbitrary square matrix.
*/

void
inpmat();


/*
** inpqrate:
**      Read queue arrival and service rates (noenv read by inpmat).
*/

void
inpqrate();


# ifndef MATGEOM_ACTIVE
#   define MATGEOM_ACTIVE 1


/* matgeom.h:
**      Header file of defines and typedefs for dynamic hierarchy analysis
**      as a network of M/M/1 queues in a random environment using matrix
**      geometric methods.
*/

#include <stdio.h>

#define FALSE 0
#define TRUE 1

#define MAX_ENV 10
#define MAX_QLEN 1000
#define ARBVAL 1.0
#define EPSILON (double) 0.0000001
#define STOPEPSV (double) 0.000000001
#define ACCEPSV (double) 0.000001
#define NEWLINE '\n'

typedef double COEFMAT[MAX_ENV][MAX_ENV];
                                /* Coefficient matrix */
typedef double PROBMAT[MAX_ENV][MAX_QLEN+1];
                                /* Matrix for matrix-geometric
                                   queue length probabilities */

#include "gauselim.h"
#include "probvect.h"
#include "matops.h"
#include "matinv.h"
#include "mgprob.h"
#include "mgmisc.h"
#include "input.h"
#include "output.h"
```

```
#endif


/*
** matinv.h: (header file)
**      Inverse of a matrix (if nonsingular). Computes inverse through
**      a diagonalization of the original matrix by Gaussian elimination.
**      Uses row pivoting with scaled values.  No solution is produced
**      if the matrix is singular.
*/

#include "matgeom.h"

/*
** matinv:
**      Main driver routine.
*/

extern FILE *inpfile, *outfile;

void
matinv();


/*
** diag:
**      Diagonalize a matrix (augmented with inverse matrix)
**      using row pivoting with implicitly scaled row elements.
**      Detects singular matrix.
*/

void
diag();


/*
** dsweep:
**      Sweep elements in all rows except the current row
*/

void
dsweep();


/*
** dswaprow:
**      Swap two rows in the matrix and RHS matrix
*/

void
dswaprow();


/*
** norminv:
**      Normalize inverse matrix by dividing each row by the
**      diagonal element in the diagonalized matrix
*/

void
norminv();


/*
** matops.h: (header file)
**      Miscellaneous matrix and vector operations.
*/

#include "matgeom.h"

/*
** matsum:
**      Sum of two matrices
*/

void
matsum();
```

```
/*
** vectsum:
**      Sum of two vectors
*/

void
vectsum();


/*
** matdiff:
**      Difference of two matrices
*/

void
matdiff();


/*
** vectdiff:
**      Diff of two vectors
*/

void
vectdiff();


/*
** matprod:
**      Product of two matrices
*/

void
matprod();


/*
** matpwr:
**      Power of a square matrix (Power must be nonnegative)
*/

void
matpwr();


/*
** innrprod:
**      Inner product of two vectors
*/

double
innrprod();


/*
** vmprod:
**      Product of a vector and a matrix
*/

void
vmprod();


/*
** mvprod:
**      Product of a matrix and a vector
*/

void
mvprod();


/*
** matcopy:
**      Copy a matrix
*/

void
```

```
matcopy();


/*
** vectcopy:
**      Copy a vector
*/

void
vectcopy();


/*
** vmcopyc:
**      Copy a vector into a column of a matrix (Assumes that the matrix
**      is of type PROBMAT)
*/

void
vmcopyc();


/*
** vmcopyr:
**      Copy a vector into a row of a matrix (Assumes that the matrix
**      is of type PROBMAT)
*/

void
vmcopyr();


/*
** mvcopyc:
**      Copy a column of a matrix into a vector (Assumes that the matrix
**      is of type PROBMAT)
*/

void
mvcopyc();


/*
** zeromat:
**      Set all entries in a matrix to zero
*/

void
zeromat();


/*
** zerovect:
**      Set all entries in a vector to zero
*/

void
zerovect();


/*
** onevect:
**      Set all entries in a vector to one
*/

void
onevect();


/*
** identmat:
**      Construct identity matrix
*/

void
identmat();
```

```
/*
** diagmat:
**      Construct a diagonal matrix from a vector
*/

void
diagmat();


/*
** mmaxdiff:
**      Maximum element by element absolute difference between two
**      square matrices
*/

double
mmaxdiff();


/*
** vectelsum:
**      Sum the elements of a vector
*/

double
vectelsum();


/*
** vmaxdiff:
**      Maximum element by element absolute difference between two
**      vectors
*/

double
vmaxdiff();


/*
** mattransp:
**      Transpose a square matrix
*/

void
mattransp();


/*
** mgmisc.h: (header file)
**      Miscellaneous routines for dynamic hierarchy/ matrix geometric
**      analysis.
*/

#include "matgeom.h"

/*
** mgwarn:
**      Write error (warning) message and source routine name to standard
**      error.
*/

void
mgwarn();


/*
** labelio:
**      Read ID label from first line of input file and print it as first
**      line of output file.
*/

void
labelio();


/*
** mgprob.h: (header file)
**      Routines for computing rate matrix, matrix-geometric queue
**      length probabilities, mean queue length, and virtual waiting
**      time for dynamic hierarchy/ matrix-geometric analysis.
*/

#include "matgeom.h"


/*
** ratemat:
**      Compute rate rate matrix by iteration starting with R = 0.
*/

void
ratemat();


/*
** qprob:
**      Compute matrix of matrix-geometric queue length probabilities.
**      Stopping condition is MAX{ (sum of queue length prob. for level
**      i) - (environment i stationary prob.)} < a small number }.
*/

void
qprob();


/*
** qmean:
**      Compute conditional (on environment) and unconditional queue
**      length means.
*/

void
qmean();


/*
** qdelay:
**      Compute (unconditional) mean virtual waiting time.
*/

double
qdelay();


/*
** output.h: (header file)
**      Output routines for dynamic hierarchy/ matrix geometric analysis
*/

#include "matgeom.h"

extern FILE *outfile;

/*
** outcmat:
**      Write augmented coefficient matrix (matrix and RHS) vector.
*/

void
outcmat();


/*
** outmat:
**      Write arbitrary square matrix.
*/

void
outmat();


/*
** outxval:
**      Write solution vector of N linear eqnations in N unknowns
*/
```

```
void
outxval();


/*
** outenv:
**      Write generator and stationary probability vector of the
**      environment process.
*/

void
outenv();


/*
** outqrate:
**      Write queue arrival rates, service rates, and rate matrix.
*/

void
outqrate();


/*
** outqprob:
**      Write joint queue length/environment probabilities (and
**      their sum for each queue length).
*/

void
outqprob();


/*
** outqmean:
**      Write conditional and unconditional mean queue lengths.
*/

void
outqmean();


/*
** outqdelay:
**      Write (unconditional) mean virtual waiting time for a queue.
*/

void
outqdelay();


/*
** probvect.h: (header file)
**      Routines for computing the invariant probability vector for
**      a Markov process with a finite state space. Part of the dynamic
**      hierarchy/ matrix geometric analysis routines.
*/

#include "matgeom.h"

/*
** probvect:
**      Compute invariant probability vector for a Markov process with a
**      finite state space.
*/

void
probvect();


/*
** normprob:
**      Normalize state probabilities so that they sum to 1
*/

void
normprob();
```

```
/*
** comprate.c
**      Main program and routines for computing link arrival rate vectors
**      from external arrival rate matrices and message routes.
*/

#include <stdio.h>

#define FALSE 0
#define TRUE 1
#define NEWLINE '\n'

#define MAX_ENV 10
#define MAX_NODE 50
#define MAX_LINK 98
        /* = 2 * (MAX_NODE - 1) */

/* Nodes are numbered 0 to nonode - 1, env states are numbered 0
  to noenv - 1, and links are numbered 1 to nolink. */
typedef double TRAFMAT[MAX_ENV][MAX_NODE][MAX_NODE];
                                /* Arrival rate matrices */
typedef double RATEMAT[MAX_ENV][MAX_LINK + 1];
                                /* Link arrival rate vectors */
typedef char ROUTEMAT[MAX_NODE][MAX_NODE][MAX_LINK + 1];
                                /* Message route sets */

FILE *inpfile, *outfile;


/*
** inpenv:
**      Read environment process information.
*/

void
inpenv(noenv, eprob)
        int     *noenv;
        double eprob[];

{
        int     envinx, tnoenv;

        fscanf(inpfile, "%d", noenv);
        tnoenv = *noenv;
        for (envinx = 0; envinx < tnoenv; envinx++)
                fscanf(inpfile, "%lf", &(eprob[envinx]));
};


/*
** inparrvrate:
**      Read arrival rate matrices.
*/

void
inparrvrate(noenv, nonode, arrvrate)
        int     noenv;
        int     *nonode;
        TRAFMAT         arrvrate;

{
        int     envinx;
        int     nodeinx1, nodeinx2;
        int     tnonode;

        fscanf(inpfile, "%d", nonode);
        tnonode = *nonode;
        for (envinx = 0; envinx < noenv; envinx++)
                for (nodeinx1 = 0; nodeinx1 < tnonode; nodeinx1++)
                        for (nodeinx2 = 0; nodeinx2 < tnonode; nodeinx2++)
                          fscanf(inpfile, "%lf",
                                &(arrvrate[envinx][nodeinx1][nodeinx2]));

};


/*
** inpmsgroute:
```

```
**      Read set of links in the route of jk-messages for each node pair
**      jk, j != k.
*/

void
inpmsgroute(nonode, msgroute)
        int     nonode;
        ROUTEMAT msgroute;

{
        int     nodeinx1, nodeinx2, linkinx;
        int     nolink;
        int     nodeid1, nodeid2, linkid;

        /* Initialize route matrix */
        nolink = 2 * (nonode - 1);
        for (nodeinx1 = 0; nodeinx1 < nonode; nodeinx1++)
                for (nodeinx2 = 0; nodeinx2 < nonode; nodeinx2++)
                        for (linkinx = 1; linkinx <= nolink; linkinx++)
                          msgroute[nodeinx1][nodeinx2][linkinx] = FALSE;

        /* Read message route sets */
        for (nodeinx1 = 0; nodeinx1 < nonode; nodeinx1++)
                for (nodeinx2 = 0; nodeinx2 < nonode; nodeinx2++)
                        {
                          fscanf(inpfile, "%d%d", &nodeid1, &nodeid2);
                          fscanf(inpfile, "%d", &linkid);
                          while (linkid != -1)
                            {
                              msgroute[nodeid1][nodeid2][linkid] = TRUE;
                              fscanf(inpfile, "%d", &linkid);
                            };
                        };

};


/*
** labelio:
**      Read ID label from first line of input file and print it as first
**      line of output file.
*/

void
labelio()

{
        char    iochar;

        fscanf(inpfile, "%c", &iochar);
        while(iochar != NEWLINE)
                {
                        fprintf(outfile, "%c", iochar);
                        fscanf(inpfile, "%c", &iochar);
                };
        fprintf(outfile, "\n");
};


/*
** outlinktraf:
**      Print link arrival rate vectors and compute and print mean
**      arrival rate to each link.
*/

void
outlinktraf(noenv, nonode, linkrate, eprob, gamma)
        int     nonode, noenv;
        RATEMAT         linkrate;
        double eprob[];
        double gamma[];

{
        int     envinx, linkinx;
        int     nolink;
        double ratesum;

        nolink = 2 * (nonode - 1);
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                {
                        fprintf(outfile, "%d", linkinx);
                        ratesum = 0.0;
                        for (envinx = 0; envinx < noenv; envinx++)
                          {
                             fprintf(outfile, " %8.4f",
                                  linkrate[envinx][linkinx]);
                             ratesum += eprob[envinx]
                                     * linkrate[envinx][linkinx];
                          };
                        fprintf(outfile, " %8.4f\n", ratesum);
                };
        for (envinx = 0; envinx < noenv; envinx++)
                fprintf(outfile, "%10.4f ", gamma[envinx]);
        fprintf(outfile,"\n");

};


/*
** linktraf:
**      Compute link arrival rate matrices given the external arrival rate
**      matrices and message route sets.
*/

void
linktraf(noenv, nonode, msgroute, arrvrate, linkrate)
        int     noenv, nonode;
        ROUTEMAT msgroute;
        TRAFMAT         arrvrate;
        RATEMAT         linkrate;

{
        int     linkinx, envinx, nodeinx1, nodeinx2;
        int     nolink;
        double ratesum;

        nolink = 2 * (nonode - 1);
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                for (envinx = 0; envinx < noenv; envinx++)
                  {
                     ratesum = 0.0;
                     for (nodeinx1 = 0; nodeinx1 < nonode; nodeinx1++)
                       for (nodeinx2 = 0; nodeinx2 < nonode; nodeinx2++)
                         if (msgroute[nodeinx1][nodeinx2][linkinx])
                           ratesum +=
                             arrvrate[envinx][nodeinx1][nodeinx2];

                     linkrate[envinx][linkinx] = ratesum;
                  };

};


/*
** thruput:
**      Calculate total arrival rate (throughput) to network under
**      each environment state.
*/

void
thruput(noenv, nonode, arrvrate, gamma)
        int     noenv;
        int     nonode;
        TRAFMAT         arrvrate;
        double gamma[];

{
        int     envinx;
        int     nodeinx1, nodeinx2;
        double ratesum;

        for (envinx = 0; envinx < noenv; envinx++)
                {
                        ratesum = 0.0;
                        for (nodeinx1 = 0; nodeinx1 < nonode; nodeinx1++)
                          for (nodeinx2 = 0; nodeinx2 < nonode; nodeinx2++)
                        ratesum += arrvrate[envinx][nodeinx1][nodeinx2];

                   gamma[envinx] = ratesum;
               };

};


main()

{
        int     noenv, nonode;
        ROUTEMAT msgroute;
        TRAFMAT        arrvrate;
        RATEMAT        linkrate;
        double eprob[MAX_ENV];
        double gamma[MAX_ENV];

        inpfile = stdin;
        outfile = stdout;

        labelio();
        inpenv(&noenv, eprob);
        inparrvrate(noenv, &nonode, arrvrate);
        inpmsgroute(nonode, msgroute);

        linktraf(noenv, nonode, msgroute, arrvrate, linkrate);
        thruput(noenv, nonode, arrvrate, gamma);

        outlinktraf(noenv, nonode, linkrate, eprob, gamma);

};


/*
** gauselim.c:
**      Gaussian elimination solution to N linear equations in N
**      unknowns. Uses row pivoting with scaled values. Solves
**      intentionally singular systems by solving a reduced size system
**      and then setting undetermined values to ARBVAL. The coefficients
**      corresponding to the reduced size system must be in the first
**      lastrow rows of amat. No solution is produced if an "unintentionally
**      singular" coefficient matrix (of possibly reduced size) is detected.
*/

#include "matgeom.h"

/*
** gauselim:
**      Main driver routine.
*/

extern FILE *inpfile, *outfile;

void
gauselim(amat, rhs, matdim, lastrow, xval, singular)
        COEFMAT        amat;                 /* Coefficient matrix */
        double rhs[];                /* Right hand side vector */
        int     matdim;                      /* Matrix size */
        int     lastrow;             /* Number of rows for reduced size
                                        coefficient matrix */
        double xval[];                       /* Solution vector */
        char    *singular;           /* TRUE if coefficient matrix is
                                        singular */

{
        double *scale;                       /* Scaling factors - row magnitudes
                                        (row elements are not actually
                                         scaled) */
        COEFMAT        tmat;                 /* Copy of amat */
        double *trhs;                /* Copy of rhs */
        int     index1, index2;              /* Matrix and vector indices */

        /* Scale and trhs space allocated dynamically */
        scale = (double *) malloc(matdim * sizeof(double));
        trhs = (double *) malloc(matdim * sizeof(double));

        /* Make copies of amat and rhs */
        matcopy(amat, matdim, matdim, tmat);
        vectcopy(rhs, matdim, trhs);

        /* Initialize, triangularize, and back-substitute */
        init(tmat, matdim, scale);
        triang(tmat, trhs, matdim, lastrow, scale, singular);
        if (! *singular)
                backsub(tmat, trhs, matdim, lastrow, xval);
        else
                mgwarn("gauselim", "singular system");

};


/*
** dabs:
**      Absolute value of long float
*/

double
dabs(dvalue)
        double dvalue;

{
        double result;

        if (dvalue < 0.0)
                result = -dvalue;
        else
                result = dvalue;
        return( result );
};


/*
** init:
**      Initialize scaling factor vector
*/

void
init(amat, matdim, scale)
        COEFMAT         amat;                   /* Coefficient matrix */
        int     matdim;                         /* Matrix size */
        double scale[];                 /* Scaling factors vector */

{
        int     index1, index2;                 /* Matrix and vector indices */
        double scalesum;                /* Temp. sum of row elements */

        for (index1 = 0; index1 < matdim; index1++)
                {
                        /* Calculate scaling factors = row sums */
                        scalesum = 0.0;
                        for (index2 = 0; index2 < matdim; index2++)
                                scalesum += dabs(amat[index1][index2]);
                        scale[index1] = scalesum;
                        if (scalesum < EPSILON)
                                mgwarn("init", "scale = 0");
                };

};


/*
** triang:
**      Triangularize coefficient matrix (augmented with RHS vector)
**      using row pivoting with implicitly scaled row elements. If
**      lastrow < matdim, triangularize only to row lastrow. This allows
**      the partial solution of systems with intentionally singular
**      coefficient matrices.
*/

void
triang(amat, rhs, matdim, lastrow, scale, singular)
        COEFMAT         amat;                   /* Coefficient matrix */
        double rhs[];                   /* RHS vector */
        int     matdim;                         /* Matrix size */
        int     lastrow;                /* Number of rows for reduced
```

```
                                   sized coefficient matrix */
        double scale[];           /* Scaling factors */
        char   *singular;         /* TRUE if the (possibly reduced
                                     size) coefficient matrix is
                                     singular */

{
        int    index1;                   /* Row index */
        int    colinx;                   /* Column index */
        int    maxrow;                   /* Pivot row index - has maximum
                                   scale ratio */
        char   nonzero;           /* TRUE if pivot row found - maximum
                                     scale ratio nonzero */

        colinx = 0;
        index1 = 0;
        /* Continue triangularization until next to last row done or until
          last column reached.
        /* Lastrow replaces matdim everywhere except in the row swap
          operation, which uses matdim as the number of columns */
        while ((index1 < lastrow - 1) && (colinx < lastrow - 1))
                {
                        /* Find index of next pivot row */
                        findpivt(amat, scale, lastrow, index1, &colinx,
                                &maxrow, &nonzero);
                        /* None found if ! nonzero. ! nonzero ==> done */
                        if (nonzero)
                                {
                                        /* Swap current and pivot rows */
                                        swaprow(amat, rhs, matdim, index1,
                                                maxrow);
                                        /* Sweep to bottom using pivot row */
                                        sweep(amat, rhs, matdim, lastrow,
                                            index1, colinx);

                                        if (colinx < lastrow - 1)
                                                colinx++;
                                        index1++;
                                };
                };

        /* Undetermined values if index1 != lastrow - 1 or if last
          diagonal element is 0 */
        *singular = (index1 != lastrow - 1)
                   || (dabs(amat[lastrow - 1][lastrow - 1]) < EPSILON);
};


/*
** swaprow:
**      Swap two rows in the coefficient matrix and RHS vector
*/

void
swaprow(amat, rhs, matdim, index1, maxrow)
        COEFMAT         amat;                   /* Coefficient matrix */
        double rhs[];             /* RHS vector */
        int    matdim;                   /* Matrix size */
        int    index1, maxrow;           /* Rows to swap */

{
        int    index2;                   /* Matrix index */

        for (index2 = 0; index2 < matdim; index2++)
               swapd(&(amat[index1][index2]), &(amat[maxrow][index2]));
        swapd(&(rhs[index1]), &(rhs[maxrow]));

};


/*
** swapd:
**      Swap two long float values
*/

void
swapd(dvalue1, dvalue2)
        double *dvalue1, *dvalue2; /* Values to swap */
```

```
{
        double tempval;              /* Temp. location for swap */

        tempval = *dvalue1;
        *dvalue1 = *dvalue2;
        *dvalue2 = tempval;
};


/*
** findpivt:
**      Find (index of) next pivot row based on implicitly scaled
**      elements
*/

void
findpivt(amat, scale, matdim, index1, colinx, maxrow, nonzero)
        COEFMAT        amat;                /* Coefficient matrix */
        double scale[];              /* Scaling factors */
        int     matdim;                     /* Matrix size */
        int     index1;                     /* Row index from outer loop in
                                       triang */
        int     *colinx;             /* Column index */
        int     *maxrow;             /* Index of pivot row if found */
        char    *nonzero;            /* TRUE if pivot row (and nonzero
                                       maximum scale ratio) found */

{
        double maxscale;             /* Maximum scale ratio */
        double scaleval;             /* Scale ratio */
        int     index2;                     /* Matrix and vector index */

        *nonzero = FALSE;
        /* Continue until pivot row found or until last column reached */
        while ((! *nonzero) && (*colinx < matdim))
                {
                        maxscale = 0.0;
                        /* Check from row index1 to last row for maximum
                          scale value */
                        for (index2 = index1; index2 < matdim; index2++)
                          {
                             scaleval = dabs(amat[index2][*colinx]
                                        / scale[index2]);
                             if (scaleval > maxscale)
                               {
                                 maxscale = scaleval;
                                 *maxrow = index2;
                               };
                          };
                        /* If maxscale < EPSILON, maximum is 0 for the
                          column. Otherwise pivot row found */
                        if (maxscale < EPSILON)
                          (*colinx)++;
                        else
                          *nonzero = TRUE;
                };

};


/*
** sweep:
**      Sweep elements in rows from row after current pivot to last
*/

void
sweep(amat, rhs, matdim, lastrow, index1, colinx)
        COEFMAT        amat;                /* Coefficient matrix */
        double rhs[];                /* RHS vector */
        int     matdim;                     /* Matrix size */
        int     lastrow;             /* Number of rows for reduced size
                                       coefficient matrix */
        int     index1;                     /* Row index from outer loop in
                                       triang */
        int     colinx;                     /* Column index */

{
        int     index2, index3;             /* Matrix and vector indices */
        double multfact;            /* Row multiplication factor */

        /* Perform sweep starting at row after current */
        for (index2 = index1 + 1; index2 < lastrow; index2++)
                {
                        multfact = amat[index2][colinx] / amat[index1][colinx];
                        /* Adjust row from colinx to end and RHS element */
                        for (index3 = colinx; index3 < matdim; index3++)
                                amat[index2][index3] -=
                                  multfact * amat[index1][index3];
                        rhs[index2] -= multfact * rhs[index1];
                };

};


/*
** backsub:
**      Solve for x values using backsubstitution with the triangularized
**      matrix and the RHS vector. Set undetermined values (at end of xval)
**      to ARBVAL.
*/

void
backsub(amat, rhs, matdim, lastrow, xval)
        COEFMAT       amat;                 /* Coefficient matrix */
        double rhs[];               /* RHS vector */
        int    matdim;                      /* Matrix size */
        int    lastrow;             /* Number of rows for reduced
                                      size coefficient matrix */
        double xval[];                      /* x value (solution) vector */

{
        int    index1, index2;              /* Matrix and vector indices */
        double diagval;             /* Temp. diagonal element */

        /* For reduced size coefficient matrix, set undetermined x values
          to ARBVAL */
        for (index1 = matdim - 1; index1 > lastrow - 1; index1--)
                xval[index1] = ARBVAL;

        /* Backsubstitute to calculate the remaining values. */
        for (index1 = lastrow - 1; index1 >= 0; index1--)
                {
                        xval[index1] = rhs[index1];
                        for (index2 = index1 + 1; index2 < matdim;
                           index2++)
                                xval[index1] -= amat[index1][index2]
                                                * xval[index2];
                        /* Divide by diagonal element to get final x value */
                        diagval = amat[index1][index1];
                        if (dabs(diagval) < EPSILON)
                                mgwarn("backsub", "diagonal = 0");
                        xval[index1] /= amat[index1][index1];
                };

};


/*
** input.c:
**      Input routines for dynamic hierarchy/ matrix geometric analysis
*/

#include "matgeom.h"

extern FILE *inpfile;

/*
** inpcmat:
**      Read augmented coefficient matrix (matrix and RHS) vector.
*/

void
inpcmat(amat, rhs, matdim, lastrow)
        COEFMAT       amat;                 /* Coefficient matrix */
        double rhs[];               /* RHS vector */
        int    *matdim;             /* Matrix size */
        int     *lastrow;           /* Number of linearly independent
                                       rows in the coefficient matrix */

{
        int     index1, index2;             /* Matrix indices */
        int     tmatdim;            /* Temp. matrix size */

        fscanf(inpfile, "%d%d", matdim, lastrow);
        tmatdim = *matdim;
        for (index1 = 0; index1 < tmatdim; index1++)
                {
                        for (index2 = 0; index2 < tmatdim; index2++)
                                fscanf(inpfile, "%lf", &(amat[index1][index2]));
                        fscanf(inpfile, "%lf", &(rhs[index1]));
                };

};


/*
** inpmat:
**      Read arbitrary square matrix.
*/

void
inpmat(amat, matdim)
        COEFMAT         amat;               /* The matrix */
        int     *matdim;            /* Matrix size */

{
        int     index1, index2;             /* Matrix indices */
        int     tmatdim;            /* Temp. matrix size */

        fscanf(inpfile, "%d", matdim);
        tmatdim = *matdim;
        for (index1 = 0; index1 < tmatdim; index1++)
                for (index2 = 0; index2 < tmatdim; index2++)
                        fscanf(inpfile, "%lf", &(amat[index1][index2]));

};


/*
** inpqrate:
**      Read queue arrival and service rates (noenv read by inpmat).
*/

void
inpqrate(arrvrate, servrate, noenv)
        double arrvrate[];          /* Arrival rates */
        double servrate[];          /* Service rates */
        int     noenv;              /* Number of environments */

{
        int     envinx;                     /* Environment index */

        for (envinx = 0; envinx < noenv; envinx++)
                fscanf(inpfile, "%lf %lf", &(arrvrate[envinx]),
                        &(servrate[envinx]));
};


/*
** mainmgp.c:
**      Main routine for running dynamic hierarchy/ matrix geometric
**      queue length probability and queue length mean routines.
*/

#include "matgeom.h"

FILE    *inpfile, *outfile;

main()

{
        double arrvrate[MAX_ENV]; /* Arrival rates */
        double servrate[MAX_ENV]; /* Service rates */
        int     noenv;              /* Number of environments */
        COEFMAT       envgenrt;           /* Environment process generator */
        double eprob[MAX_ENV];            /* Stationary probability vector
                                       of environment process */
        COEFMAT       rmatrix;            /* Rate matrix */
        PROBMAT       qlenprob;           /* Joint environment/queue length
                                       probabilities */
        double condqmean[MAX_ENV];/* Conditional queue length means */
        double uncqmean;          /* Unconditional mean queue
                                       length */
        double meandelay;         /* Mean delay (virtual waiting time)
                                       at queue */
        int    stopqlen;          /* Index of last queue length
                                       probability computed */
        char   singular;          /* TRUE if submatrix of environment
                                        process generator is singular */
        COEFMAT       gentrans;           /* Transpose of environment process
                                       generator */

        inpfile = stdin;
        outfile = stdout;

        labelio();
        /* Read number of environment states and environment process
          generator */
        inpmat(envgenrt, &noenv);
        /* Read rate vectors */
        inpqrate(arrvrate, servrate, noenv);

        /* Find stationary probabilities for environment process.
          Then compute rate matrix, environment/queue length probabilities,
          and queue length means */
        /* Transpose of generator must be used for probvect/gauselim to
          work right */
        mattransp(envgenrt, noenv, gentrans);
        probvect(gentrans, noenv, eprob, &singular);
        ratemat(envgenrt, arrvrate, servrate, noenv, eprob, rmatrix);
        qprob(rmatrix, arrvrate, servrate, eprob, noenv, qlenprob,
            &stopqlen);
        qmean(eprob, noenv, qlenprob, stopqlen, condqmean, &uncqmean);

        /* Compute mean delay (virtual waiting time) */
        meandelay = qdelay(envgenrt, eprob, noenv, servrate, qlenprob,
                        stopqlen);

        /* Write everything */
        outenv(envgenrt, eprob, noenv);
        outqrate(arrvrate, servrate, rmatrix, noenv);
        outqprob(qlenprob, noenv, stopqlen);
        outqmean(condqmean, uncqmean, noenv);
        outqdelay(meandelay);

        exit(0);
};


/*
** mainminv.c:
**      Main routine for testing matrix inversion routine.
*/

#include "matgeom.h"

FILE    *inpfile, *outfile;

main()

{
        COEFMAT       amat;               /* Coefficient matrix */
        COEFMAT       inverse;            /* Inverse matrix */
        int    matdim;                    /* Matrix size */
        char   singular;          /* TRUE if coefficient matrix is
                                       singular */

        inpfile = stdin;
        outfile = stdout;

        inpmat(amat, &matdim);
        fprintf(outfile, "Original matrix:\n");
        outmat(amat, matdim);
        matinv(amat, matdim, inverse, &singular);
        if (! singular)
                {
                        fprintf(outfile, "Inverse matrix:\n");
                        outmat(inverse, matdim);
                };
};


/*
** mainpvec.c:
**      Main routine for testing probability vector routine.
*/

#include "matgeom.h"

FILE    *inpfile, *outfile;

main()

{
        COEFMAT         amat;                   /* Coefficient matrix */
        double rhs[MAX_ENV];                    /* RHS vector */
        int     matdim;                         /* Matrix size */
        double xval[MAX_ENV];                   /* Solution vector */
        char    singular;               /* TRUE if coefficient matrix is
                                          singular */
        int     lastrow;                /* Number of linearly independent
                                          rows in the coefficient matrix */
        COEFMAT         atrans;                         /* Transpose of amat */

        inpfile = stdin;
        outfile = stdout;

        inpcmat(amat, rhs, &matdim, &lastrow);
        outcmat(amat, rhs, matdim);
        mattransp(amat, matdim, atrans);
        probvect(atrans, matdim, xval, &singular);
        if (! singular)
                outxval(xval, matdim, matdim - lastrow);
};


/*
** mainseqn.c:
**      Main routine for testing linear system solution routine.
*/

#include "matgeom.h"

FILE    *inpfile, *outfile;

main()

{
        COEFMAT         amat;                   /* Coefficient matrix */
        double rhs[MAX_ENV];                    /* RHS vector */
        int     matdim;                         /* Matrix size */
        double xval[MAX_ENV];                   /* Solution vector */
        char    singular;               /* TRUE if coefficient matrix is
                                          singular */
        int     lastrow;                /* Number of linearly independent
                                          rows in the coefficient matrix */

        inpfile = stdin;
        outfile = stdout;

        inpcmat(amat, rhs, &matdim, &lastrow);
        outcmat(amat, rhs, matdim);
        gauselim(amat, rhs, matdim, lastrow, xval, &singular);
        if (! singular)
                outxval(xval, matdim, matdim - lastrow);
};


/*
** matinv.c:
```

```
**      Inverse of a matrix (if nonsingular). Computes inverse through
**      a diagonalization of the original matrix by Gaussian elimination.
**      Uses row pivoting with scaled values.  No solution is produced
**      if the matrix is singular.
*/

#include "matgeom.h"

/*
** matinv:
**      Main driver routine.
*/

extern FILE *inpfile, *outfile;

void
matinv(amat, matdim, inverse, singular)
        COEFMAT         amat;                   /* Matrix to invert */
        COEFMAT         inverse;                /* Inverse matrix */
        int     matdim;                         /* Matrix size */
        char    *singular;              /* TRUE if matrix is singular */

{
        double *scale;                          /* Scaling factors - row magnitudes
                                        (row elements are not actually
                                         scaled) */
        COEFMAT         tmat;                   /* Copy of amat */
        int     index1, index2;                 /* Matrix and vector indices */

        /* Scale space allocated dynamically */
        scale = (double *) malloc(matdim * sizeof(double));

        /* Make copy of amat */
        matcopy(amat, matdim, matdim, tmat);
        /* Initialize the inverse matrix to the identity matrix */
        identmat(inverse, matdim);

        /* Initialize, triangularize, and normalize */
        init(tmat, matdim, scale);
        diag(tmat, inverse, matdim, scale, singular);
        if (! *singular)
                norminv(tmat, inverse, matdim);
        else
                mgwarn("matinv", "singular matrix");

};


/*
** diag:
**      Diagonalize a matrix (augmented with inverse matrix)
**      using row pivoting with implicitly scaled row elements.
**      Detects singular matrix.
*/

void
diag(amat, rhsmat, matdim, scale, singular)
        COEFMAT         amat;                   /* Coefficient matrix */
        COEFMAT         rhsmat;                         /* RHS (inverse) matrix */
        int     matdim;                         /* Matrix size */
        double scale[];                 /* Scaling factors */
        char    *singular;              /* TRUE if the matrix is singular */

{
        int     index1;                         /* Row index */
        int     colinx;                         /* Column index */
        int     maxrow;                         /* Pivot row index - has maximum
                                        scale ratio */
        char    nonzero;                /* TRUE if pivot row found - maximum
                                        scale ratio nonzero */

        colinx = 0;
        index1 = 0;
        /* Continue diagonalization until next to last row done or until
          last column reached. */
        while ((index1 < matdim - 1) && (colinx < matdim - 1))
                {
                        /* Find index of next pivot row */
                        findpivt(amat, scale, matdim, index1, &colinx,
                              &maxrow, &nonzero);
                        /* None found if ! nonzero. ! nonzero ==> done */
                        if (nonzero)
                              {
                                    /* Swap current and pivot rows */
                                    dswaprow(amat, rhsmat, matdim, index1,
                                          maxrow);
                                    /* Sweep entire row using pivot row */
                                    dsweep(amat, rhsmat, matdim, index1,
                                        colinx);

                                    if (colinx < matdim - 1)
                                          colinx++;
                                    index1++;
                              };
                  };

      /* Undetermined values if index1 != matdim - 1 or if last
        diagonal element is 0 */
      *singular = (index1 != matdim - 1)
                || (dabs(amat[matdim - 1][matdim - 1]) < EPSILON);
      /* Sweep last column */
      if (! *singular)
            dsweep(amat, rhsmat, matdim, matdim - 1, matdim - 1);
};


/*
** dsweep:
**      Sweep elements in all rows except the current row
*/

void
dsweep(amat, rhsmat, matdim, index1, colinx)
      COEFMAT       amat;                   /* The matrix */
      COEFMAT       rhsmat;                         /* RHS matrix */
      int     matdim;                       /* Matrix size */
      int     index1;                       /* Row index from outer loop in
                                       diag */
      int     colinx;                       /* Column index */

{
      int     index2, index3;               /* Matrix and vector indices */
      double multfact;               /* Row multiplication factor */

      /* Perform sweep starting at first row */
      for (index2 = 0; index2 < matdim; index2++)
            if (index2 != index1)
                  {
                    multfact = amat[index2][colinx] /
                            amat[index1][colinx];
                    /* Adjust row from colinx to end and RHS row */
                    for (index3 = colinx; index3 < matdim; index3++)
                      amat[index2][index3] -=
                        multfact * amat[index1][index3];
                    for (index3 = 0; index3 < matdim; index3++)
                      rhsmat[index2][index3] -=
                        multfact * rhsmat[index1][index3];
                  };

};


/*
** dswaprow:
**      Swap two rows in the matrix and RHS matrix
*/

void
dswaprow(amat, rhsmat, matdim, index1, maxrow)
      COEFMAT       amat;                   /* Coefficient matrix */
      COEFMAT       rhsmat;                         /* RHS matrix */
      int     matdim;                       /* Matrix size */
      int     index1, maxrow;               /* Rows to swap */

{
      int     index2;                       /* Matrix index */
```

```
        for (index2 = 0; index2 < matdim; index2++)
                {
                        swapd(&(amat[index1][index2]), &(amat[maxrow][index2]));
                        swapd(&(rhsmat[index1][index2]),
                            &(rhsmat[maxrow][index2]));
                }

};


/*
** norminv:
**      Normalize inverse matrix by dividing each row by the
**      diagonal element in the diagonalized matrix
*/

void
norminv(amat, rhsmat, matdim)
        COEFMAT       amat;                 /* The diagonalized matrix */
        COEFMAT       rhsmat;                       /* Inverse matrix to normalize */
        int     matdim;                     /* Matrix size */

{
        int     index1, index2;             /* Matrix indices */
        double diagval;             /* Diagonal value from amat */

        for (index1 = 0; index1 < matdim; index1++)
                {
                        diagval = amat[index1][index1];
                        if (dabs(diagval) > EPSILON)
                                for (index2 = 0; index2 < matdim; index2++)
                                        rhsmat[index1][index2] /= diagval;
                        else
                                mgwarn("norminv", "diagonal = 0.0");
                };

};


/*
** matops.c:
**      Miscellaneous matrix and vector operations.
*/

#include "matgeom.h"

/*
** matsum:
**      Sum of two matrices
*/

void
matsum(mat1, mat2, norow, nocol, matres)
        COEFMAT       mat1, mat2;           /* Matrices to sum */
        int     norow, nocol;       /* Number of rows and columns */
        COEFMAT       matres;                       /* Sum matrix */

{
        int     index1, index2;             /* Matrix indices */

        for (index1 = 0; index1 < norow; index1++)
                for (index2 = 0; index2 < nocol; index2++)
                        matres[index1][index2] = mat1[index1][index2]
                                                  + mat2[index1][index2];

};


/*
** vectsum:
**      Sum of two vectors
*/

void
vectsum(vect1, vect2, vectsize, vectres)
        double vect1[], vect2[];   /* Vectors to sum */
        int     vectsize;           /* Vector size */
        double vectres[];          /* Sum vector */
```

```
{
        int     index;                  /* Vector index */

        for (index = 0; index < vectsize; index++)
                vectres[index] = vect1[index] + vect2[index];
};


/*
** matdiff:
**      Difference of two matrices
*/

void
matdiff(mat1, mat2, norow, nocol, matres)
        COEFMAT         mat1, mat2;             /* Mat2 subtracted from mat1 */
        int     norow, nocol;           /* Number of rows and columns */
        COEFMAT         matres;                         /* Difference matrix */

{
        int     index1, index2;                 /* Matrix indices */

        for (index1 = 0; index1 < norow; index1++)
                for (index2 = 0; index2 < nocol; index2++)
                        matres[index1][index2] = mat1[index1][index2]
                                                - mat2[index1][index2];

};


/*
** vectdiff:
**      Difference of two vectors
*/

void
vectdiff(vect1, vect2, vectsize, vectres)
        double vect1[], vect2[];        /* Vect2 subtracted from vect1 */
        int     vectsize;               /* Vector size */
        double vectres[];               /* Difference vector */

{
        int     index;                  /* Vector index */

        for (index = 0; index < vectsize; index++)
                vectres[index] = vect1[index] - vect2[index];
};


/*
** matprod:
**      Product of two matrices
*/

void
matprod(mat1, mat2, dim1, dim2, dim3, matres)
        COEFMAT         mat1, mat2;             /* Matrices to multiply */
        int     dim1, dim2, dim3;       /* Matrix dimensions */
        COEFMAT         matres;                         /* Product matrix */

{
        int     index1, index2, index3;     /* Matrix indices */
        double sum;                     /* Temp. sum */

        for (index1 = 0; index1 < dim1; index1++)
                for (index2 = 0; index2 < dim3; index2++)
                        {
                                sum = 0.0;
                                for (index3 = 0; index3 < dim2; index3++)
                                        sum += mat1[index1][index3]
                                            * mat2[index3][index2];
                                matres[index1][index2] = sum;
                        };

};


/*
```

```
** matpwr:
**      Power of a square matrix (Power must be nonnegative)
*/

void
matpwr(amat, matsize, power, matres)
        COEFMAT       amat;                 /* Matrices to raise to power */
        int    matsize;            /* Matrix size */
        int    power;              /* Power to raise the matrix to */
        COEFMAT       matres;                       /* Power matrix */

{
        int    index1;                      /* Power index */
        COEFMAT tempmat;           /* Temp. matrix */

        identmat(matres, matsize);
        for (index1 = 1; index1 <= power; index1++)
                {
                        matprod(amat, matres, matsize, matsize, matsize,
                                tempmat);
                        matcopy(tempmat, matsize, matsize, matres);
                };
};


/*
** innrprod:
**      Inner product of two vectors
*/

double
innrprod(vect1, vect2, vectsize)
        double vect1[], vect2[];   /* Vectors for inner product */
        int    vectsize;           /* Size of vectors */

{
        int    index;              /* Vector index */
        double sum;                /* Temp. sum */

        sum = 0.0;
        for (index = 0; index < vectsize; index++)
                sum += vect1[index] * vect2[index];

        return( sum );
};


/*
** vmprod:
**      Product of a vector and a matrix
*/

void
vmprod(avect, amat, dim1, dim2, vectres)
        double avect[];            /* Vector to multiply */
        COEFMAT       amat;                 /* Matrix to multiply */
        int    dim1, dim2;         /* Matrix and vector dimensions */
        double vectres[];          /* Product vector */

{
        int    index1, index2;              /* Matrix and vector indices */
        double sum;                /* Temp. sum */

        for (index1 = 0; index1 < dim1; index1++)
                {
                        sum = 0.0;
                        for (index2 = 0; index2 < dim2; index2++)
                                sum += avect[index2] * amat[index2][index1];
                        vectres[index1] = sum;
                };

};


/*
** mvprod:
**      Product of a matrix and a vector
*/
```

```
void
mvprod(amat, avect, dim1, dim2, vectres)
	COEFMAT	amat;		/* Matrix to multiply */
	double avect[];		/* Vector to multiply */
	int	dim1, dim2;	/* Matrix and vector dimensions */
	double vectres[];	/* Product vector */

{
	int	index1, index2;		/* Matrix and vector indices */
	double sum;		/* Temp. sum */

	for (index1 = 0; index1 < dim1; index1++)
		{
			sum = 0.0;
			for (index2 = 0; index2 < dim2; index2++)
				sum += amat[index1][index2] * avect[index2];
			vectres[index1] = sum;
		};

};


/*
** matcopy:
**	Copy a matrix
*/

void
matcopy(amat, dim1, dim2, matres)
	COEFMAT	amat;		/* Original matrix */
	int	dim1, dim2;	/* Matrix dimensions */
	COEFMAT	matres;			/* Copy matrix */

{
	int	index1, index2;		/* Matrix indices */

	for (index1 =  0; index1 < dim1; index1++)
		for (index2 = 0; index2 < dim2; index2++)
			matres[index1][index2] = amat[index1][index2];

};


/*
** vectcopy:
**	Copy a vector
*/

void
vectcopy(avect, vectsize, vectres)
	double avect[];		/* Original vector */
	int	vectsize;	/* Vector size */
	double vectres[];	/* Copy vector */

{
	int	index;		/* Vector index */

	for (index =  0; index < vectsize; index++)
		vectres[index] = avect[index];

};


/*
** vmcopyc:
**	Copy a vector into a column of a matrix (Assumes that the matrix
**	is of type PROBMAT)
*/

void
vmcopyc(avect, amat, vectsize, matcol)
	double avect[];		/* Vector to copy */
	PROBMAT	amat;		/* Matrix to copy into */
	int	vectsize;	/* Vector size */
	int	matcol;		/* Column of matrix to copy into */

{
	int	index;		/* Vector and matrix index */
```

```
        for (index = 0; index < vectsize; index++)
                amat[index][matcol] = avect[index];
};


/*
** vmcopyr:
**      Copy a vector into a row of a matrix (Assumes that the matrix
**      is of type PROBMAT)
*/

void
vmcopyr(avect, amat, vectsize, matrow)
        double avect[];                 /* Vector to copy */
        PROBMAT         amat;                   /* Matrix to copy into */
        int     vectsize;               /* Vector size */
        int     matrow;                         /* Row of matrix to copy into */

{
        int     index;                  /* Vector and matrix index */

        for (index = 0; index < vectsize; index++)
                amat[matrow][index] = avect[index];
};


/*
** mvcopyc:
**      Copy a column of a matrix into a vector (Assumes that the matrix
**      is of type PROBMAT)
*/

void
mvcopyc(amat, avect, colsize, matcol)
        PROBMAT         amat;                   /* Matrix to copy into */
        double avect[];                 /* Vector to copy */
        int     colsize;                /* Matrix column size */
        int     matcol;                         /* Column of matrix to copy from */

{
        int     index;                  /* Vector and matrix index */

        for (index = 0; index < colsize; index++)
                avect[index] = amat[index][matcol];
};


/*
** zeromat:
**      Set all entries in a matrix to zero
*/

void
zeromat(amat, dim1, dim2)
        COEFMAT         amat;                   /* Matrix to zero */
        int     dim1, dim2;             /* Matrix dimensions */

{
        int     index1, index2;                 /* Matrix indices */

        for (index1 = 0; index1 < dim1; index1++)
                for (index2 = 0; index2 < dim2; index2++)
                        amat[index1][index2] = 0.0;
};


/*
** zerovect:
**      Set all entries in a vector to zero
*/

void
zerovect(avect, vectsize)
        double avect[];                 /* Vector to zero */
        int     vectsize;               /* Vector size */

{
        int     index;                  /* Vector index */
```

```
	for (index = 0; index < vectsize; index++)
		avect[index] = 0.0;
};


/*
** onevect:
**	Set all entries in a vector to one
*/

void
onevect(avect, vectsize)
	double avect[];		/* Vector to set to ones */
	int	vectsize;		/* Vector size */

{
	int	index;			/* Vector index */

	for (index = 0; index < vectsize; index++)
		avect[index] = 1.0;
};


/*
** identmat:
**	Construct identity matrix
*/

void
identmat(amat, matdim)
	COEFMAT	amat;			/* Matrix set to identity matrix
					(Must be square) */
	int	matdim;				/* Matrix size */

{
	int	index1, index2;			/* Matrix indices */

	for (index1 = 0; index1 < matdim; index1++)
		for (index2 = 0; index2 < matdim; index2++)
			amat[index1][index2] = (index1 == index2)? 1.0: 0.0;
};


/*
** diagmat:
**	Construct a diagonal matrix from a vector
*/

void
diagmat(avect, vectlen, dmat)
	double avect[];		/* Vector to use for diagonal */
	int	vectlen;		/* Vector length */
	COEFMAT	dmat;			/* Matrix set to a diagonal matrix
					(Must be square) */

{
	int	index1, index2;			/* Vector and matrix indices */

	for (index1 = 0; index1 < vectlen; index1++)
		for (index2 = 0; index2 < vectlen; index2++)
			dmat[index1][index2] = (index1 == index2)?
						avect[index1]: 0.0;
};


/*
** mmaxdiff:
**	Maximum element by element absolute difference between two
**	square matrices
*/

double
mmaxdiff(mat1, mat2, matdim)
	COEFMAT	mat1, mat2;		/* The matrices (must be square) */
	int	matdim;				/* Matrix size */

{
	int	index1, index2;			/* Matrix indices */
```

```
        double diff;                /* Absolute value of difference
                                       between two elements */
        double maxd;                /* Maximum difference */

        maxd = 0.0;
        for (index1 = 0; index1 < matdim; index1++)
                for (index2 = 0; index2 < matdim; index2++)
                        {
                                diff = dabs(mat1[index1][index2] -
                                        mat2[index1][index2]);
                                if (diff > maxd)
                                        maxd = diff;
                        };

        return(maxd);
};


/*
** vectelsum:
**      Sum the elements of a vector.
*/

double
vectelsum(avect, vectlen)
        double avect[];
        int    vectlen;

{
        int    vectinx;
        double sum;

        sum = 0.0;
        for (vectinx = 0; vectinx < vectlen; vectinx++)
                sum += avect[vectinx];

        return( sum );
};


/*
** vmaxdiff:
**      Maximum element by element absolute difference between two
**      vectors
*/

double
vmaxdiff(vect1, vect2, vectlen)
        double vect1[], vect2[];    /* The vectors */
        int    vectlen;             /* Vector length */

{
        int    index;               /* Vector index */
        double diff;                /* Absolute value of difference
                                       between two elements */
        double maxd;                /* Maximum difference */

        maxd = 0.0;
        for (index = 0; index < vectlen; index++)
                {
                        diff = dabs(vect1[index] - vect2[index]);
                        if (diff > maxd)
                                maxd = diff;
                };

        return(maxd);
};


/*
** mattransp:
**      Transpose a square matrix.
*/

void
mattransp(amat, matdim, matres)
        COEFMAT        amat;                /* Matrix to transpose */
        int    matdim;                      /* Matrix size */
        COEFMAT         matres;                 /* Transpose matrix */

{
        int     matinx1, matinx2;   /* Matrix indices */

        for (matinx1 = 0; matinx1 < matdim; matinx1++)
                for (matinx2 = 0; matinx2 < matdim; matinx2++)
                        matres[matinx2][matinx1] = amat[matinx1][matinx2];
};


/*
** mgmisc.c:
**      Miscellaneous routines for dynamic hierarchy/ matrix geometric
**      analysis.
*/

#include "matgeom.h"

/*
** mgwarn:
**      Write error (warning) message and source routine name to standard
**      error.
*/

void
mgwarn(source, message)
        char    *source;                /* Name of calling routine */
        char    *message;               /* Error or warning message */

{
        fprintf(stderr, "warning- %s: %s\n", source, message);
};


/*
** labelio:
**      Read ID label from first line of input file and print it as first
**      line of output file.
*/

void
labelio()

{
        char    iochar;

        fscanf(inpfile, "%c", &iochar);
        while(iochar != NEWLINE)
                {
                        fprintf(outfile, "%c", iochar);
                        fscanf(inpfile, "%c", &iochar);
                };
        fprintf(outfile, "\n");
};


/*
** mgprob.c:
**      Routines for computing rate matrix, matrix-geometric queue
**      length probabilities, mean queue length, and virtual waiting
**      time for dynamic hierarchy/ matrix-geometric analysis.
*/

#include "matgeom.h"


/*
** ratemat:
**      Compute rate matrix by iteration starting with R = 0.
*/

void
ratemat(envgenrt, arrvrate, servrate, noenv, eprob, rmatrix)
        COEFMAT         envgenrt;               /* Environment process generator */
        double arrvrate[];          /* Arrival rate vector */
        double servrate[];          /* Service rate vector */
        int     noenv;              /* Number of environment states
```

```
                              (and size of rate matrix) */
double eprob[];             /* Stationary probability vector
                              of the environment process */
COEFMAT     rmatrix;               /* The rate matrix */

{
COEFMAT     gentransp;             /* Transpose of environment process
                              generator */
double avgarrv;             /* Mean arrival rate */
double avgserv;             /* Mean service rate */
double mediff;                     /* Maximum element by element
                              difference between two iterates
                              of the rate matrix */
double vediff;                     /* Maximum element by element
                              difference between two vectors
                              - used for accuracy check */

double *tempvec1, *tempvec2;       /* Temp. vectors used in accuracy
                              check */
char   singular;            /* TRUE is rate matrix is singu-
                              lar */
char   done;                /* TRUE when stopping condition
                              (on iterates of rmatrix)
                              satisfied */
COEFMAT     diagarrv;              /* Diagonal matrix based on arrival
                              rate vector */
COEFMAT     diagserv;              /* Diagonal matrix based on service
                              rate vector */
COEFMAT     diagas;                       /* Diagonal matrix based on sum of
                              arrival and service rate vectors */
COEFMAT almat;                     /* Matrix difference between envgenrt
                              and diagas */
COEFMAT alinv;                     /* Inverse of almat */
COEFMAT rmatold;            /* Previous iterate of rate matrix */
COEFMAT     rold2;                 /* Previous iterate of rate matrix
                              squared */
COEFMAT     tempmat1, tempmat2; /* Temp. matrices */
                            /* Also used in accuracy check */
double altqmean;
/* diags var */
int    iter = 0;

/* Stationary probability vector for environment process (eprob)
  is calculated in main or elsewhere */

/* Test for system stability */
avgarrv = innrprod(eprob, arrvrate, noenv);
avgserv = innrprod(eprob, servrate, noenv);
if (avgarrv > avgserv)
       mgwarn("ratemat", "queue is not stable");

/* Compute some matrices for use in the iteration for the rate
  matrix */
diagmat(arrvrate, noenv, diagarrv);
diagmat(servrate, noenv, diagserv);
matsum(diagarrv, diagserv, noenv, noenv, diagas);
matdiff(diagas, envgenrt, noenv, noenv, almat);
matinv(almat, noenv, alinv, &singular);
if (singular)
       mgwarn("ratemat", "matrix A1 singular");

/* Set first iterate to the zero matrix and continue iteration
  until condition on maximum difference satisfied */
zeromat(rmatold, noenv, noenv);
done = FALSE;
while (! done)
       {
              matprod(rmatold, rmatold, noenv, noenv, noenv,
                     rold2);
              matprod(rold2, diagserv, noenv, noenv, noenv,
                     tempmat1);
              matsum(diagarrv, tempmat1, noenv, noenv, tempmat2);
              matprod(tempmat2, alinv, noenv, noenv, noenv,
                     rmatrix);

              mediff = mmaxdiff(rmatrix, rmatold, noenv, noenv);
              done = mediff < STOPEPSV;
              /* diags */
```

```
                iter++;
                if (! done)
                        matcopy(rmatrix, noenv, noenv, rmatold);
        };

        /* Check accuracy of solution */
        tempvec1 = (double *) malloc(noenv * sizeof(double));
        tempvec2 = (double *) malloc(noenv * sizeof(double));
        matdiff(envgenrt, diagarrv, noenv, noenv, tempmat1);
        matprod(rmatrix, diagserv, noenv, noenv, noenv, tempmat2);
        /* tempmat1 is reused and overwritten here- ok with matsum */
        matsum(tempmat1, tempmat2, noenv, noenv, tempmat1);
        onevect(tempvec1, noenv);
        mvprod(tempmat1, tempvec1, noenv, noenv, tempvec2);
        zerovect(tempvec1, noenv);
        vediff = vmaxdiff(tempvec2, tempvec1, noenv);
        if (vediff > ACCEPSV)
                mgwarn("ratemat", "accuracy check failed");
        /* diags */
        fprintf(stderr, "ratemat acc = %f\n", vediff);
        fprintf(stderr, "number of iter for R= %d\n", iter);
        /* diags. - alternate computation of mean Q len */
        identmat(tempmat1, noenv);
        matdiff(tempmat1, rmatrix, noenv, noenv, tempmat2);
        matinv(tempmat2, noenv, tempmat2, &singular);
        vmprod(eprob, tempmat2, noenv, noenv, tempvec1);
        vmprod(tempvec1, rmatrix, noenv, noenv, tempvec2);
        onevect(tempvec1, noenv);
        altqmean = innrprod(tempvec1, tempvec2, noenv);
        fprintf(stderr, "alt. mean q len = %f\n", altqmean);

};


/*
** qprob:
**      Compute matrix of matrix-geometric queue length probabilities.
**      Stopping condition is MAX{ (sum of queue length prob. for level
**      i) - (environment i stationary prob.)} < a small number }
*/

void
qprob(rmatrix, arrvrate, servrate, eprob, noenv, qlenprob, stopqlen)
        COEFMAT         rmatrix;                /* Rate matrix */
        double arrvrate[];              /* Arrival rate vector */
        double servrate[];              /* Service rate vector */
        double eprob[];                 /* Stationary probability vector
                                          for the environment process */
        int     noenv;                  /* Number of environment states */
        PROBMAT         qlenprob;               /* Matrix of queue length prob-
                                          abilities. (Row index = env.
                                          state. Column index = queue
                                          length) */
        int     *stopqlen;              /* Index of last queue length for
                                          which probability computed */

{
        int     envinx;                         /* Environment index */
        int     qleninx;                /* Queue length index */
        double *tempvec1, *tempvec2;            /* Temp vectors */
        double *probsum;                /* Sum of queue length probabili-
                                          ties for each environment state */
        double vediff;                          /* Difference between eprob and
                                          probsum. Used for checking stop-
                                          ping condition */
        COEFMAT         irdiff;                         /* Difference between identity
                                          matrix and rate matrix */
        COEFMAT         idmat;                  /* Identity matrix */
        char    done;                   /* TRUE when stopping condition
                                          satisfied */

        /* Allocate local vector space */
        tempvec1 = (double *) malloc(noenv * sizeof(double));
        tempvec2 = (double *) malloc(noenv * sizeof(double));
        probsum = (double *) malloc(noenv * sizeof(double));

        /* Compute level 0 probabilities */
        identmat(idmat, noenv);
        matdiff(idmat, rmatrix, noenv, noenv, irdiff);
        vmprod(eprob, irdiff, noenv, noenv, tempvec1);
        /* Copy probabilities into prob. matrix */
        vmcopyc(tempvec1, qlenprob, noenv, 0);

        /* Compute probabilities until stopping condition satisfied
          or end of prob. matrix reached */
        /* Initialize prob. sums to level 0 probabilities */
        vectcopy(tempvec1, noenv, probsum);
        done = FALSE;
        qleninx = 1;
        while ((! done) && (qleninx <= MAX_QLEN))
                {
                        vectcopy(tempvec1, noenv, tempvec2);
                        vmprod(tempvec2, rmatrix, noenv, noenv, tempvec1);
                        /* Copy probabilities for this level into prob.
                          matrix */
                        vmcopyc(tempvec1, qlenprob, noenv, qleninx);

                        /* Check stopping condition */
                        vectsum(probsum, tempvec1, noenv, probsum);
                        vediff = vmaxdiff(eprob, probsum, noenv);;
                        done = vediff < STOPEPSV;
                        qleninx++;
                };

        if (! done)
                mgwarn("qprobmat", "accuracy check failed");
        *stopqlen = --qleninx;
};


/*
** qmean:
**      Compute conditional (on environment) and unconditional queue
**      length means.
*/

void
qmean(eprob, noenv, qlenprob, stopqlen, condqmean, uncqmean)
        double eprob[];                 /* Env. stationary probabilities */
        int     noenv;                  /* Number of environments */
        PROBMAT         qlenprob;               /* Environment/queue length
                                          probabilities */
        int     stopqlen;               /* Last index for which queue length
                                          computed */
        double condqmean[];             /* Conditional mean queue length */
        double *uncqmean;               /* Unconditional mean queue length */

{
        int     envinx;                         /* Environment index */
        int     qleninx;                /* Queue length index */
        double wtpsum;                          /* Partial sum in computing expected
                                          value */

        /* Calculate vector of queue length means conditioned on env.
          state. Conditional queue length probability is joint env./queue
          length probability divided by env. stationary probability */
        for (envinx = 0; envinx < noenv; envinx++)
                {
                        wtpsum = 0.0;
                        for (qleninx = 1; qleninx < stopqlen; qleninx++)
                                wtpsum += qleninx * qlenprob[envinx][qleninx]
                                                / eprob[envinx];
                        condqmean[envinx] = wtpsum;
                };

        /* Calculate unconditional mean queue length by removing condition
          on env. state in the conditional means and summing */
        wtpsum = 0.0;
        for (envinx = 0; envinx < noenv; envinx++)
                wtpsum += eprob[envinx] * condqmean[envinx];
        *uncqmean = wtpsum;
};


/*
** qdelay:
```

```
**      Compute (unconditional) mean virtual waiting time.
*/

double
qdelay(envgenrt, eprob, noenv, servrate, qlenprob, stopqlen)
        COEFMAT      envgenrt;           /* Env. process generator */
        double eprob[];            /* Env. stationary probabilities */
        int     noenv;             /* Number of env. states */
        double servrate[];         /* Service rate vector */
        PROBMAT      qlenprob;           /* Queue length probabilities */
        int     stopqlen;          /* Last index for which queue length
                                     computed */

{
        int     qleninx;           /* Queue length index */
        int     innerinx;          /* Inner loop index */
        double wbar;               /* Partial sum in computing mean
                                     virtual waiting time */
        double meanserv;           /* Mean service rate */
        COEFMAT      innersum;           /* Matrix sum formed in inner loop */
        double       *qprobvect;         /* Queue length probabilities for
                                     a fixed queue length */
        COEFMAT      diagserv;           /* Diagonal matrix based on service
                                     rate vector */
        COEFMAT      tempmat;            /* Temp. matrix */
        double *tempvec1, *tempvec2;     /* Temp. vectors */
        COEFMAT mat1, mat2, mat3, mat4, mat5;
                                   /* Misc. use matrices */
        char   singular;           /* Returned as TRUE if singular
                                     matrix passed to matinv */

        /* Allocate space for vectors */
        qprobvect = (double *) malloc((stopqlen + 1) * sizeof(double));
        tempvec1 = (double *) malloc(noenv * sizeof(double));
        tempvec2 = (double *) malloc(noenv * sizeof(double));

        /* Form initial values of matrices */
        diagmat(servrate, noenv, diagserv);
        matdiff(diagserv, envgenrt, noenv, noenv, tempmat);
        matinv(tempmat, noenv, mat1, &singular);
        matprod(mat1, diagserv, noenv, noenv, noenv, mat2);
        /* Mat1 is initial value of inner sum matrix. */
        matprod(mat1, mat2, noenv, noenv, noenv, mat3);
        matcopy(mat3, noenv, noenv, mat4);

        /* Compute mean virtual waiting time */
        wbar = 0.0;
        for (qleninx = 1; qleninx <= stopqlen; qleninx++)
                {
                        /* Add current iterate of inner sum. */
                        mvcopyc(qlenprob, tempvec1, noenv, qleninx);
                        vmprod(tempvec1, mat3, noenv, noenv, tempvec2);
                        onevect(tempvec1, noenv);
                        wbar += innrprod(tempvec2, tempvec1, noenv);

                        /* Compute next iterate of inner sum. */
                        matprod(mat4, mat2, noenv, noenv, noenv, mat5);
                        matprod(mat2, mat3, noenv, noenv, noenv, mat1);
                        matsum(mat5, mat1, noenv, noenv, mat3);
                };
        meanserv = innrprod(eprob, servrate, noenv);
        /* Add mean service time to get total waiting time */
        wbar += 1.0 / meanserv;

        return( wbar );
};


/*
** netdelay.c
**      Main program and routines for dynamic hierarchy network delay.
*/

#include <stdio.h>

#define FALSE 0
#define TRUE 1
#define NEWLINE '\n'
```

```
#define MAX_ENV 10
#define MAX_NODE 50
#define MAX_LINK 98
        /* = 2 * (MAX_NODE - 1) */

/* nodes are numbered 0 to nonode - 1, env states are numbered 0
 to noenv - 1, and links are numbered 1 to nolink. */
typedef double RATEMAT[MAX_ENV][MAX_LINK + 1];
                                /* Link arrival rate vectors */

FILE *inpfile, *outfile;


/*
** inpenv:
**      Read envrionment process information.
*/

void
inpenv(noenv, eprob)
        int     *noenv;
        double eprob[];

{
        int     envinx;
        int     tnoenv;

        fscanf(inpfile, "%d", noenv);
        tnoenv = *noenv;
        for (envinx = 0; envinx < tnoenv; envinx++)
                fscanf(inpfile, "%lf", &(eprob[envinx]));
};


/*
** inplinkrate:
**      Read arrival rate matrices.
*/

void
inplinkrate(noenv, nolink, linkrate, thruput)
        int     noenv;
        int     *nolink;
        RATEMAT         linkrate;
        double thruput[];

{
        int     envinx;
        int     linkinx;
        int     tnolink;

        fscanf(inpfile, "%d", nolink);
        tnolink = *nolink;
        for (envinx = 0; envinx < noenv; envinx++)
                {
                        for (linkinx = 1; linkinx <= tnolink; linkinx++)
                          fscanf(inpfile, "%lf",
                                &(linkrate[envinx][linkinx]));
                        fscanf(inpfile, "%lf", &(thruput[envinx]));
                };

};


/*
** labelio:
**      Read ID label from first line of input file and print it as first
**      line of output file.
*/

void
labelio()

{
        char    iochar;

        fscanf(inpfile, "%c", &iochar);
        while(iochar != NEWLINE)
```

```
                {
                        fprintf(outfile, "%c", iochar);
                        fscanf(inpfile, "%c", &iochar);
                };
        fprintf(outfile, "\n");
};


/*
** inplinkdelay:
**      Read mean delay for individual links.
*/

void
inplinkdelay(nolink, linkdelay)
        int     nolink;
        double linkdelay[];

{
        int     linkinx;

        for (linkinx = 1; linkinx <= nolink; linkinx++)
                fscanf(inpfile, "%lf", &(linkdelay[linkinx]));
};


/*
** outmeanrate:
**      Print mean link arrival rates.
*/

void
outmeanrate(nolink, mlinkrate, meantput)
        int     nolink;
        double mlinkrate[];
        double meantput;

{
        int     linkinx;

        fprintf(outfile, "\n");
        fprintf(outfile, "LINK  MEAN ARRIVAL RATE\n");
        fprintf(outfile, "------------------------\n");
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                fprintf(outfile, " %3d   %10.4f\n", linkinx,
                        mlinkrate[linkinx]);
        fprintf(outfile, "\n");
        fprintf(outfile, "MEAN THRUPUT: %11.4f\n", meantput);

};


/*
** outnetdelay:
**      Print mean network delay.
*/

void
outnetdelay(mnetdelay)
        double mnetdelay;

{
        fprintf(outfile, "\n");
        fprintf(outfile, "MEAN NETWORK DELAY: %12.4f\n", mnetdelay);
};



/*
** meanrate:
**      Compute mean arrival rates (over all environments) for links
**      and mean throughput (over all environments).
*/

void
meanrate(noenv, nolink, eprob, linkrate, thruput, mlinkrate, meantput)
        int     noenv;
        int     nolink;
        double eprob[];
        RATEMAT       linkrate;
        double thruput[];
        double mlinkrate[];
        double *meantput;

{
        int    envinx, linkinx;
        double ratesum;

        /* Compute mean link arrival rates. */
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                {
                        ratesum = 0.0;
                        for (envinx = 0; envinx < noenv; envinx++)
                                ratesum += eprob[envinx]
                                           * linkrate[envinx][linkinx];
                        mlinkrate[linkinx] = ratesum;
                };

        /* Compute mean throughput. */
        ratesum = 0.0;
        for (envinx = 0; envinx < noenv; envinx++)
                ratesum += eprob[envinx] * thruput[envinx];
        *meantput = ratesum;

};


/*
** netdelay:
**      Compute mean network delay based on virtual delay (computed
**      elsewhere) and arrival rates for individaul links.
*/

double
netdelay(nolink, mlinkrate, linkdelay, meantput)
        int    nolink;
        double *mlinkrate;
        double *linkdelay;
        double meantput;

{
        int    linkinx;
        double delaysum;

        delaysum = 0.0;
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                delaysum += mlinkrate[linkinx] * linkdelay[linkinx];
        delaysum /= meantput;

        return( delaysum );
};


main()

{
        int    noenv, nolink;
        RATEMAT       linkrate;
        double *thruput[MAX_ENV];
        double *eprob[MAX_ENV];
        double *linkdelay[MAX_LINK];
        double *mlinkrate[MAX_LINK];
        double meantput;
        double mnetdelay;
        int    envinx;

        inpfile = stdin;
        outfile = stdout;

        labelio();
        inpenv(&noenv, eprob);
        inplinkrate(noenv, &nolink, linkrate, thruput);
        inplinkdelay(nolink, linkdelay);
        meanrate(noenv, nolink, eprob, linkrate, thruput, mlinkrate,
                &meantput);
```

```
	mnetdelay = netdelay(nolink, mlinkrate, linkdelay, meantput);

	outmeanrate(nolink, mlinkrate, meantput);
	outnetdelay(mnetdelay);
};


/*
** output.c:
**	Output routines for dynamic hierarchy/ matrix geometric analysis.
*/

#include "matgeom.h"

extern FILE *outfile;

/*
** outcmat:
**	Write augmented coefficient matrix (matrix and RHS) vector.
*/

void
outcmat(amat, rhs, matdim)
	COEFMAT		amat;			/* Coefficient matrix */
	double rhs[];			/* RHS vector */
	int	matdim;				/* Matrix size */

{
	int	index1, index2;			/* Matrix indices */

	/* Output fields are set for matrix size <= 99 and coefficient and
	  RHS values <= 999.999 (%6.3f field). Doubles are printed in
	  single precision format. Lines will be too long and will wrap
	  if matrix size > 8 */
	fprintf(outfile, "\nCoefficient matrix:\n");
	fprintf(outfile, " ");
	for (index1 = 0; index1 < matdim; index1++)
		fprintf(outfile, "   %2d ", index1);
	fprintf(outfile, "  RHS \n");
	for (index1 = 0; index1 < matdim; index1++)
		{
			fprintf(outfile, "%2d", index1);
			for (index2 = 0; index2 < matdim; index2++)
				fprintf(outfile, " %6.3f",
					(float) amat[index1][index2]);
			fprintf(outfile, " %6.3f\n", (float) rhs[index1]);
		};

};


/*
** outmat:
**	Write arbitrary square matrix.
*/

void
outmat(amat, matdim)
	COEFMAT		amat;			/* The matrix */
	int	matdim;				/* Matrix size */

{
	int	index1, index2;			/* Matrix indices */

	/* Output fields are set for matrix size <= 99 and coefficient and
	  RHS values <= 999.999 (%6.3f field). Doubles are printed in
	  single precision format. Lines will be too long and will wrap
	  if matrix size > 8 */
	fprintf(outfile, " ");
	for (index1 = 0; index1 < matdim; index1++)
		fprintf(outfile, "   %2d ", index1);
	fprintf(outfile, "\n");
	for (index1 = 0; index1 < matdim; index1++)
		{
			fprintf(outfile, "%2d", index1);
			for (index2 = 0; index2 < matdim; index2++)
				fprintf(outfile, " %6.3f",
					(float) amat[index1][index2]);
```

```
                fprintf(outfile, "\n");
        };

};


/*
** outxval:
**      Write solution vector of N linear eqnations in N unknowns
*/

void
outxval(xval, vectlen, noarb)
        double xval[];                  /* Solution vector */
        int     vectlen;                /* Vector (and system) size */
        int     noarb;                  /* Number of arbitrary values if
                                          coefficient matrix is singular */

{
        int     index;                  /* Vector index */

        /* Output fields are set for vector size <= 99 and solution values
          <= 999.9999 (%7.4f field). Doubles are printed in single
          precision format. */
        fprintf(outfile, "\nSolution of %2d equations in %2d unknowns:\n",
                vectlen, vectlen);
        fprintf(outfile, " i   x(i)\n");
        for (index = 0; index < vectlen; index++)
                fprintf(outfile, "%2d  %7.4f\n", index, xval[index]);
        fprintf(outfile, "(%2d undetermined values set to %7.4f)\n",
                noarb, ARBVAL);

};


/*
** outenv:
**      Write generator and stationary probability vector of the
**      environment process.
*/

void
outenv(envgenrt, eprob, noenv)
        COEFMAT         envgenrt;               /* Environment process generator */
        double eprob[];                 /* Stationary probability vector
                                          of envvironment */
        int     noenv;                  /* Number of environment states */

{
        int     envinx;                         /* Environment index */

        fprintf(outfile, "\nEnvironment Process Generator\n");
        outmat(envgenrt, noenv);
        fprintf(outfile, "\nEnvironment Process Stationary ");
        fprintf(outfile, "Probabilities\n\n");
        fprintf(outfile, "ENVIRONMENT  PROBABILITY\n");
        fprintf(outfile, "------------------------\n");
        for (envinx = 0; envinx < noenv; envinx++)
                fprintf(outfile, "   %3d       %5.4f\n", envinx,
                        eprob[envinx]);
};


/*
** outqrate:
**      Write queue arrival rates, service rates, and rate matrix.
*/

void
outqrate(arrvrate, servrate, rmatrix, noenv)
        double arrvrate[];              /* Arrival rates */
        double servrate[];              /* Service rates */
        COEFMAT         rmatrix;                /* Rate matrix */
        int     noenv;                  /* Number of environments */

{
        int     envinx;                         /* Environment index */
        fprintf(outfile, "\nQueue Rates\n\n");
        fprintf(outfile, "ENVIRONMENT  ARRIVAL RATE  SERVICE RATE\n");
        fprintf(outfile, "----------------------------------------\n");
        for (envinx = 0; envinx < noenv; envinx++)
                fprintf(outfile, "  %3d     %7.4f    %7.4f\n",
                        envinx, arrvrate[envinx], servrate[envinx]);
        fprintf(outfile, "\nRate Matrix\n");
        outmat(rmatrix, noenv);
};


/*
** outqprob:
**      Write joint queue length/environment probabilities (and
**      their sum for each queue length).
*/

void
outqprob(qlenprob, noenv, stopqlen)
        PROBMAT         qlenprob;               /* Matrix of probabilities */
        int     noenv;                  /* Number of environments */
        int     stopqlen;               /* Index of last queue length
                                           probability computed */

{
        int     envinx;                         /* Env. (row) index */
        int     qleninx;                /* Queue length (column) index */
        double qprobsum;                /* Sum of probabilities for a fixed
                                           queue length */

        fprintf(outfile, "\nQueue Length/Environment Probabilities\n\n");
        fprintf(outfile, " QUEUE\n");
        fprintf(outfile, "LENGTH   ENVIRONMENT\n");
        fprintf(outfile, "    ");
        for (envinx = 0; envinx < noenv; envinx++)
                fprintf(outfile, "   %3d", envinx);
        fprintf(outfile, "   SUM\n");
        fprintf(outfile, "------");
        for (envinx = 0; envinx <= noenv; envinx++)
                fprintf(outfile, "--------");
        fprintf(outfile, "-\n");

        for (qleninx = 0; qleninx <= stopqlen; qleninx++)
                {
                        fprintf(outfile, " %4d ", qleninx);
                        qprobsum = 0.0;
                        for (envinx = 0; envinx < noenv; envinx++)
                                {
                                  fprintf(outfile, " %5.4f",
                                        qlenprob[envinx][qleninx]);
                                  qprobsum += qlenprob[envinx][qleninx];
                                };
                        fprintf(outfile, " %5.4f\n", qprobsum);
                };

};


/*
** outqmean:
**      Write conditional and unconditional mean queue lengths.
*/

void
outqmean(condqmean, uncqmean, noenv)
        double condqmean[];             /* Conditional mean queue lengths */
        double uncqmean;                /* Unconditional mean queue length */
        int     noenv;                  /* Number of environments */

{
        int     envinx;                         /* Environment index */

        fprintf(outfile, "\nQueue Length Means\n\n");
        fprintf(outfile, "        CONDITIONAL MEAN\n");
        fprintf(outfile, "ENVIRONMENT   QUEUE LENGTH\n");
        fprintf(outfile, "----------------------------\n");
        for (envinx = 0; envinx < noenv; envinx++)
                fprintf(outfile, "  %3d     %10.5f\n", envinx,
                condqmean[envinx]);
        fprintf(outfile, "\nUNCONDITIONAL MEAN QUEUE LENGTH: %10.5f\n",
                uncqmean);
};


/*
** outqdelay:
**      Write (unconditional) mean virtual waiting time for a queue.
*/

void
outqdelay(meandelay)
        double meandelay;           /* Mean delay (virtual waiting
                                      time) */

{
        fprintf(outfile, "\n");
        fprintf(outfile, "MEAN DELAY (VIRTUAL WAITING TIME): %11.5f\n\n",
                meandelay);
};


/*
** probvect.c:
**      Routines for computing the invariant probability vector for
**      a Markov process with a finite state space. Part of the dynamic
**      hierarchy/ matrix geometric analysis routines.
*/

#include "matgeom.h"

/*
** probvect:
**      Compute invariant probability vector for a Markov process with a
**      finite state space.
*/

void
probvect(generatr, nostate, statprob, singular)
        COEFMAT     generatr;                   /* Transpose of generator
                                                  for the process */
        int     nostate;                    /* Number of states */
        double statprob[];                  /* Invariant probability vector
                                              for the process */
        char    *singular;                  /* TRUE if the reduced size
                                              system of equations is
                                              singular */

{
        double *rhs;                        /* RHS vector (of zeros) used
                                              in solving for the state
                                              probabilities */
        int     index;                      /* Vector index */

        /* Allocate and initialize RHS vector */
        rhs = (double *) malloc(nostate * sizeof(double));
        zerovect(rhs, nostate);

        /* Solve the first nostate - 1 equations of the system and normalize
           the solution */
        gauselim(generatr, rhs, nostate, nostate - 1, statprob, singular);
        if (! *singular)
                normprob(statprob, nostate);
        else
                mgwarn("probvect", "singular submatrix of generator");

};


/*
** normprob:
**      Normalize state probabilities so that they sum to 1.
*/

void
normprob(statprob, nostate)
        double statprob[],                  /* Invariant probability
```

```
                                              vector */
        int     nostate;                    /* Number of states */

{
        int     index;                      /* Vector index */
        double probsum;                     /* Sum of probabilities */

        probsum = 0.0;
        for (index = 0; index < nostate; index++)
              probsum += statprob[index];
        if (probsum < EPSILON)
              mgwarn("probvect", "probability sum = 0");

        for (index = 0; index < nostate; index++)
              statprob[index] /= probsum;

};


/*
** statdel.c
**      Main program and routines for calculating link capacity
**      for a static hierarchy and then    calculating mean network delay.
*/

#include <stdio.h>
#include <math.h>

#define MAX_NODE 50
#define MAX_LINK 98
        /* = 2 * (MAX_NODE - 1) */

#define NEWLINE '\n'

typedef double EVALVECT[MAX_LINK + 1];
                                      /* Vector for misc. uses */

FILE *inpfile, *outfile;


/*
** inpnet:
**      Read all network parameters.
*/

void
inpnet(linktraf, thruput, meanmlen, totcap, nolink)
        EVALVECT linktraf;
        double *thruput, *meanmlen;
        double *totcap;
        /* double     *tmax; */
        int     *nolink;

{
        int     linkinx;
        int     tnolink;

        fscanf(inpfile, "%d", &tnolink);
        *nolink = tnolink;
        for (linkinx = 1; linkinx <= tnolink; linkinx++)
              fscanf(inpfile, "%lf", &(linktraf[linkinx]));
        fscanf(inpfile, "%lf", thruput);

        fscanf(inpfile, "%lf", meanmlen);
        fscanf(inpfile, "%lf", totcap);
        /* fscanf(inpfile, "%lf", tmax); */
};


/*
** labelio:
**      Read ID label from first line of input file and print it as first
**      line of output file.
*/

void
labelio()
```

```
{
        char   iochar;

        fscanf(inpfile, "%c", &iochar);
        while(iochar != NEWLINE)
                {
                        fprintf(outfile, "%c", iochar);
                        fscanf(inpfile, "%c", &iochar);
                };
        fprintf(outfile, "\n");
};


/*
** outcaps:
**      Print capacity assignments.
*/

void
outcaps(capacity, nolink)
        EVALVECT capacity;
        int    nolink;

{
        int    linkinx;

        fprintf(outfile, "\n");
        fprintf(outfile, "link   capacity\n");
        fprintf(outfile, "----------------\n");
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                fprintf(outfile, " %3d %11.4f\n", linkinx,
                        capacity[linkinx]);

};


/*
** outdelay:
**      Print link and network delay.
*/

void
outdelay(linkdelay, netdelay, nolink)
        EVALVECT linkdelay;
        double netdelay;
        int    nolink;

{
        int    linkinx;

        fprintf(outfile, "\n");
        fprintf(outfile, "link   delay\n");
        fprintf(outfile, "----------------\n");
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                fprintf(outfile, " %3d %10.6f\n", linkinx,
                        linkdelay[linkinx]);
        fprintf(outfile, "\n");
        fprintf(outfile, "network delay: %11.6f\n", netdelay);

};


/*
** dsqrt:
**      Calculate dual square-root capacity assignments for the links in
**      a static hierarchy (Not used).
*/

void
dsqrt(linktraf, thruput, meanmlen, tmax, nolink, capacity)
        EVALVECT linktraf;
        EVALVECT capacity;
        double thruput, meanmlen, tmax;
        int    nolink;

{
        int    linkinx;
        double trafsum, assign;
        trafsum = 0.0;
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                trafsum += sqrt(linktraf[linkinx]);

        for (linkinx = 1; linkinx <= nolink; linkinx++)
                capacity[linkinx] = linktraf[linkinx] * meanmlen
                                  + meanmlen * sqrt(linktraf[linkinx])
                                  / (thruput * tmax) * trafsum;

};


/*
** psqrt:
**      Calculate primal square-root capacity assignments for the links in
**      a static hierarchy.
*/

void
psqrt(linktraf, thruput, meanmlen, totcap, nolink, capacity)
        EVALVECT linktraf;
        EVALVECT capacity;
        double thruput, meanmlen;
        double totcap;
        int     nolink;

{
        int     linkinx;
        double trafsum, assign, mincap, exccap;

        trafsum = 0.0;
        mincap = 0.0;
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                {
                        trafsum += sqrt(linktraf[linkinx]);
                        mincap += linktraf[linkinx] * meanmlen;
                };

        exccap = totcap - mincap;
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                capacity[linkinx] = linktraf[linkinx] * meanmlen
                                  + exccap * sqrt(linktraf[linkinx])
                                  / trafsum;

};


/*
** delay:
**      Calculate mean delay for each link and mean network delay.
*/

void
delay(linktraf, meanmlen, capacity, thruput, nolink, linkdelay, netdelay)
        EVALVECT linktraf, capacity;
        double meanmlen, thruput;
        int     nolink;
        EVALVECT linkdelay;
        double *netdelay;

{
        int     linkinx;
        double delaysum, servrate;

        delaysum = 0.0;
        for (linkinx = 1; linkinx <= nolink; linkinx++)
                {
                        servrate = (1.0 / meanmlen) * capacity[linkinx];
                        linkdelay[linkinx] = (1.0 / servrate) / (1.0
                                        - (linktraf[linkinx] / servrate));
                        delaysum += linktraf[linkinx] * linkdelay[linkinx];
                };
        delaysum /= thruput;

        *netdelay = delaysum;
};
```

```
main()

{
        EVALVECT linktraf;
        EVALVECT capacity;
        double thruput;
        double meanmlen;
        double tmax;
        double totcap;
        EVALVECT linkdelay;
        double netdelay;
        int    nolink;

        inpfile = stdin;
        outfile = stdout;

        labelio();
        inpnet(linktraf, &thruput, &meanmlen, &totcap, &nolink);
        psqrt(linktraf, thruput, meanmlen, totcap, nolink, capacity);
        delay(linktraf, meanmlen, capacity, thruput, nolink, linkdelay,
            &netdelay);
        outcaps(capacity, nolink);
        outdelay(linkdelay, netdelay, nolink);
};
```