February 1985 Vol. 64 No. 2 Part 2

# AT&T TECHNICAL JOURNAL

A JOURNAL OF THE AT&T COMPANIES

## Computing Science and Systems

Traffic

Discrete Line Representation

Modeling of Computing Systems

Decision Tables

Protocol Specifications

Clocked Schedules

AT&T TECHNICAL JOURNAL (ISSN 8756-2324) is published ten times each year by AT&T, 550 Madison Avenue, New York, NY 10022; C. L. Brown, Chairman of the Board; T. O. Davis, Secretary. The Computing Science and Systems section and the special issues are included as they become available. Subscriptions: United States—1 year $35; foreign—1 year $45.

Back issues of the special, single-subject supplements may be obtained by writing to the AT&T Customer Information Center, P.O. Box 19901, Indianapolis, Indiana 46219, or by calling (800) 432-6600. Back issues of the general, multisubject issues may be obtained from University Microfilms, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.

Payment for foreign subscriptions or single copies must be made in United States funds, or by check drawn on a United States bank and made payable to the AT&T Technical Journal and sent to AT&T Bell Laboratories, Circulation Dept., Room 1E335, 101 J. F. Kennedy Pky, Short Hills, NJ 07078.





Printed in U.S.A. Second-class postage paid at Short Hills, NJ 07078 and additional mailing offices. Postmaster: Send address changes to the AT&T Technical Journal, Room 1E335, 101 J. F. Kennedy Pky, Short Hills, NJ 07078.

AT&T
TECHNICAL
JOURNAL

VOL. 64 FEBRUARY 1985 NO. 2, PART 2



**Unintrusive Communication of Status in a Packet Network in Heavy Traffic** 463
G. J. Foschini

**A Note on Discrete Representation of Lines** 481
M. D. McIlroy

**Models for Configuring Large-Scale Distributed Computing Systems** 491
B. Gavish

**Inverted Decision Tables and Their Application: Automating the Translation of Specifications to Programs** 533
L. S. Levy and H. T. Stump

**Proposed Specification of BX.25 Link Layer Protocol** 559
R. P. Kurshan

**Approximate Analysis of a Generalized Clocked Schedule** 597
A. A. Fredericks, B. L. Farrell, and D. F. DeMaio

**Numerical Computation of Delays in Clocked Schedules** 617
M. H. Ackroyd

**Analysis of Clocked Schedules—High-Priority Tasks** 633
B. T. Doshi

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Unintrusive Communication of Status in a Packet Network in Heavy Traffic

By G. J. FOSCHINI*

(Manuscript received October 31, 1983)

We consider a packet switched network in the situation where communication resources are used close to capacity. Such heavy traffic may seem to present a dilemma. On one hand, at each node, the usefulness of status information about queues at other nodes is manifest. On the other hand, since the limitation of transmission resources causes backup, heavy load seems to be the worst situation in which to expend still more communication resources to convey status information. Under extremely general assumptions on interarrivals and services, a scaling appropriate for queueing processes in networks under heavy traffic has been established. Under these assumptions, we demonstrate that the status of the entire network can be communicated throughout the network, perfectly, in real time, without influencing the scaled queueing process. So, within a precise mathematical setting, we see that there is no dilemma: status can be conveyed at a negligible cost in a network operating at heavy load. Most of today's computer networks are designed for light-to-moderate loading. Yet heavy traffic analysis is growing in relevance, as is explained. A brief introduction to the subject of convergence of queueing systems is included.

## I. INTRODUCTION AND RESULT

### *1.1 Status information and the contention for communication resources*

The key concern in the mathematical theory of computer communication networks is the stochastic contention for limited network resources and the associated queueing and delay processes. (See Refs.

* AT&T Bell Laboratories.

1 through 5 for basic expositions.) Queueing theory has provided the approach to the subject of stochastic resource contention. The waiting lines in a computer network are represented as a vector $Q(t) = (Q_1(t), Q_2(t), \cdots, Q_K(t))$, where $Q_k(t)$ represents the number of items queued for the $k$th resource. The dynamics of the network are associated with the law of evolution of $Q(t)$.

In this paper, we focus on the contention for communication resources. We assume that the network is a packet switched network where transmission is the bottleneck resource, and we consider the operation of the network in the condition of heavy traffic. By heavy traffic we mean that the demand for communication resources is approaching the network's capability for communication, so that queues are very large.

Status information refers to the information needed to describe queueing (or delay) as it evolves in time at each node. At moderate traffic levels, the issue of communication of status appears formidable. (Some initial investigations are reported in Ref. 6.) How much information about the various components of $Q(t)$ should be communicated to other nodes? Is there a point of diminishing returns beyond which the communication channels become choked with status information, thereby significantly worsening the queueing problem? Questions such as these are very difficult to approach with queueing theory methods. However, under the assumption of heavy traffic, we show that the status issue crystallizes and becomes mathematically tractable.

The circumstance of heavy traffic emphasizes what seems to be the dilemma associated with the communication of nodal status information. On the one hand, during times of heavy traffic, such status information is especially useful. If at each node in the network there is information available about all the queues at every other node, this information could be used to control the flow of packets within the network and to judiciously control access to the network. On the other hand, the limitation of transmission resources causes queueing in the first place; therefore, during heavy loading, it may seem inadvisable to expend still more communication resources to convey status information.

We show that there is no dilemma. In the context of a precise mathematical model, the status information of the entire network can be transmitted throughout the network without any substantial taxing of communication resources. Specifically, one can send enough information to describe the queueing situation as it develops in real time. Yet the utilization of communication resources for status transmission as compared to the utilization of communication resources for transmitting other packets tends to zero as the traffic increases.

Figure 1 illustrates the main result in its simplest form. Two

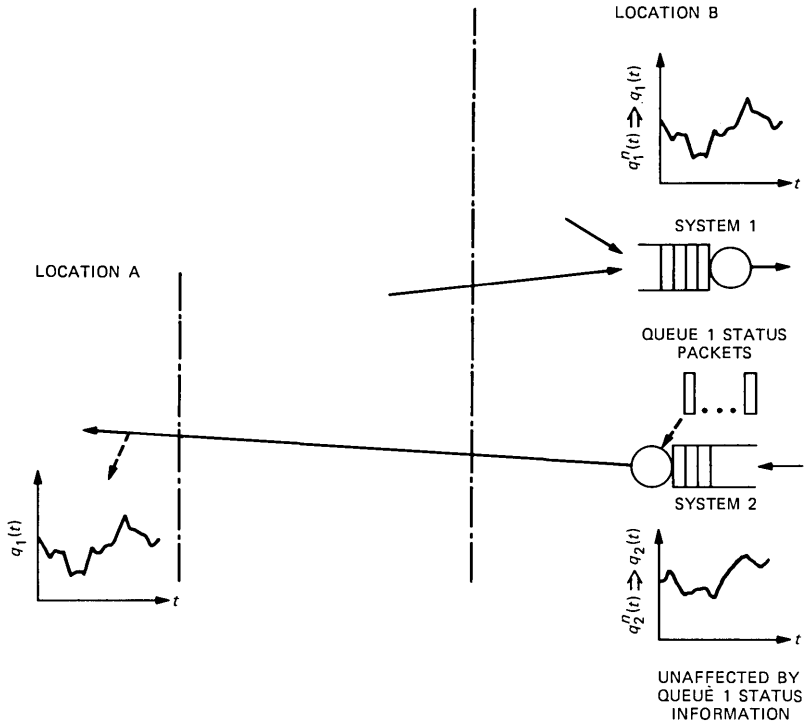



Fig. 1—Simple version of main result.

queueing systems collocated at B represent packets waiting for transmission resources. General independent sequences of positive random variables are assumed to govern interarrival and transmission times. Both systems are assumed to be in heavy traffic. In a sense that we will make exact, the following holds: Packets can be added to those on system 2 such that the recipient of the packets at location A can reconstruct $q_1(t)$ perfectly in real time. The process $q_2(t)$ remains the same after the status packets are added.

### *1.2 Heavy traffic scaling*

Showing that status can be conveyed unintrusively involves analyzing the scaling appropriate for the description of queues (or delays) associated with the condition of heavy load. As will be explained in the following sections, it has been established that in heavy traffic a continuous path process $q(t)$ replaces the jump process $Q(t)$ as the natural descriptor of the number of items queued. Roughly, the normalization involved in obtaining $q(t)$ is a sort of mathematical "lens" used to place the essential features of the $Q(t)$ process in focus as the traffic is increased. If one did not normalize, the queueing process

would tend to infinity in the heavy traffic limit. Furthermore, if one does not normalize in just the right way, the resulting process trivializes to be identically infinity or identically zero.

One reason why the communication of the queueing situation is easier in the heavy traffic limit is that it is not necessary to convey $Q(t)$ exactly. The limiting expression of status, $q(t)$, is insensitive to the fine structure of the approximating processes, so that it is not necessary to communicate each arrival and departure and the exact time of occurrence in order to communicate status. Yet the process $q(t)$ possesses sample paths of such elaborate structure that the result that negligible transmission resources are needed to communicate status in the limit may be surprising, or even counterintuitive.

### *1.3 Content of sequel*

In the following we discuss the growing relevance of heavy traffic models. The paper is written so as to make the result accessible to readers unfamiliar with the specialized subject of convergence of queueing systems in heavy traffic and, at the same time, provide a concise introduction to this topic. This introductory material, presented in Sections II and III, gives the basis for a detailed probabilistic analysis in Section IV. This analysis develops the main result that the various effects associated with the heavy traffic limit can be resolved to allow a communication scheme providing unintrusive transmission of status. The literature providing the derivation of the foundation material is cited for readers who would like more detail.

For readers already familiar with the heavy traffic theory of packet networks, we mention at this point a key feature of our main result in Section IV. Namely, from the status information received at a node about a queue at any other network node, an approximation to the queueing process sample path can be constructed so that in the heavy traffic limit the distance between the queueing process sample path and its approximation converges in probability to zero. Moreover, the transmission of the approximation can be accomplished unintrusively in the sense that the limiting queueing process is unchanged in distribution by the additional packets transmitted to convey the approximations.

In Section V, with the status communication issue obviated, we broach the next level of inquiry dealing with use of status information to reduce queueing and delay.

## II. THE HEAVY TRAFFIC MODEL AND ITS RELEVANCE

In the heavy traffic model for the network queueing process, the components of $q(t) = (q_1(t), \cdots, q_K(t))$ represent the time evolution of normalized queue sizes in the limit as the rate $\Lambda$ at which work

enters the network approaches $U$, the ultimate rate at which the network can do work. If $U - \Lambda = \Delta/\sqrt{n}$, where $\Delta$ is a positive constant, and if $Q(t)$ is the aforementioned $K$ vector of queue sizes, then $q(t) = \lim_{n\to\infty} Q(nt)/\sqrt{n}$. The process $q(t)$ is an example of a diffusion process. A precise definition of the limit is discussed in Section III.

A mathematically equivalent way to obtain the same limit is to compose $Q(t)/\sqrt{n}$ and replace $U$ and $\Lambda$ by $nU$ and $n\Lambda$, respectively. The first procedure, involving $Q(nt)/\sqrt{n}$, relates to a situation where, in a given network, operation close to capacity is taking place. The $Q(t)/\sqrt{n}$ situation relates to a sequence of networks, each accommodating a new demand, $n\Lambda$. (There is nothing special about $n$. Any function of $n$ that increases without bound would yield the same limit.) Such a sequence is of interest if the demand for data communications grows substantially over a period of many years. The transmission capabilities will grow to keep pace with the demand. If processing power and memory are inexpensive relative to transmission cost, transmission will be the bottleneck. The behavior of the limit of $Q(t)/\sqrt{n}$ will be descriptive of long-term tendency of the network queueing processes.

Network design issues that have no crisp resolution when addressed in the context of today's networks under nominal operating conditions can have a clear answer in the context of the heavy traffic limit. The issue of the value of transmitting status information is an example, as we shall see. The result we will prove is a very general one in that the network can be quite arbitrary, as for example Reiman and Harrison's generalization of the Jackson-type network,[7] where each exogenous interarrival process, as well as each service process, is modeled as an independent, identically distributed (i.i.d.) sequence with a general nonnegative distribution.

### *2.1 Motivation for diffusion models in computer network theory*

Mathematically, diffusions are finite dimensional vectors whose components are continuous random time functions with the property that at each point in time the statistical law concerning future evolution depends only on the present value of the function. (Diffusion evolution can also depend on time, but we shall not use this degree of flexibility. References 8 and 9 are basic references on diffusions.) Diffusion representation of behavior is a natural representation to consider in instances where the randomness stems from a large number of cumulative influences. Diffusion models are used to replace alternative models, which are useful in addressing small-scale phenomena but are too cumbersome for analyzing large-scale behavior. In computer networks the demand for network resources establishes the scale of the queueing processes. The queues can be large enough to require

a new scale that no longer tracks each quantum jump (i.e., arrival or departure).

### 2.2 Weiner process with drift

In this subsection we mention for future reference some diffusions of special interest in the theory of computer networks, namely the reflecting Weiner processes with drift. Although one can point out significant usefulness of other diffusions,[10] the following are the diffusions that are relevant in much of computer network theory.

Let $\Delta$ and $\sigma$ be constants where $\sigma > 0$. By the Wiener process $W(\Delta, \sigma^2)$ with drift $\Delta$ (and dispersion $\sigma^2$) we mean the continuous path Gaussian process starting at $w(0)$ (a constant) with $E[w(t_1) - w(t_2)] = \Delta(t_1 - t_2)$ and $E[(w(t_1) - Ew(t_1))(w(t_2) - Ew(t_2))] = \sigma^2 \min(t_1, t_2)$. This Wiener process, as it stands, is often unsuited to addressing heavy traffic queueing problems. After all, a queue cannot be negative. The Wiener process can be modified to keep the sample paths positive. This brings us to the subject of reflection. For $\Delta = 0$, the reflecting Wiener process $\underline{W}(0, \sigma^2)$ (reflected about the zero state) is defined to be $\underline{w}(t) = |w(t)|$. For $\Delta \neq 0$, $|w(t)|$ is not appropriate for describing reflection since $|w(t)|$ has drifts of opposite sign on the sets $\{t \mid w(t) > 0\}$ and $\{t \mid w(t) < 0\}$. For the reflecting Wiener process, we want a continuous path process whose incremental behavior matches that of $w(t)$ but with the constraint of nonnegativity. The variant $\underline{w}(t) = w(t) - \min[0, \min_{(0,t)} w(t)]$ is the natural definition for the reflecting Wiener process. As long as $\underline{w}(t) > 0$, then $\underline{w}(t)$ has the same differential properties as $w(t)$. The subtractive term serves to keep the process nonnegative. The notation $\underline{W}(\Delta, \sigma^2)$ is used to denote the reflecting Wiener process.

Vector versions of $W$ and $\underline{W}$ have been defined where $\Delta$ is a vector and $\sigma^2$ is a positive definite matrix. In the vector case, to complete the description, angles of reflection must be specified[7] at the boundaries that prevent the process components from being negative.

### 2.3 Importance of diffusion models

In the context of today's computer networks, the condition of heavy traffic represents an extreme condition. We discuss reasons why consideration of this extreme is worthwhile.

First, the maturation of demand for computer network services is not in sight. The accelerating demand coupled with the rapid technological advances associated with network components is likely to foster a rapid evolution of networks accommodating more and more demand. As demand for network resources grows, economies of scale will encourage the operation of networks at higher utilizations. This is most easily seen in the context of a single M/M/1 queue.[11] Recall the mean delay formula $\bar{D} = (\mu - \lambda)^{-1}$. Fix the mean delay $\bar{D}$, and it is

apparent that as the demand, $\lambda$, increases, $\mu$ increases by a smaller amount, so that $\rho = (\lambda/\mu) \rightarrow 1$. (See Ref. 12 for a related discussion.)

From the delay formula, we see that if $\lambda$ increases with slope bounded away from zero, the increase in $\mu$ relative to $\lambda$ that is needed to make $\bar{D}$ negligible goes to zero. Note, however, from the formula for mean queue size $\bar{Q} = (1 - \rho)^{-1}$, that the memory resources needed to queue packets do not enjoy this economy of scale. In packet communication systems where $\bar{D}$ is small, one should consider whether packet voice services should be accommodated. If the answer is yes, $\lambda$ grows at a still faster rate. In such circumstances, one would anticipate a sequence of queueing processes behaving more like the Wiener process idealization.

Another reason diffusion models are relevant is that for a specific network operating at moderate utilization it is important to understand behavior in crisis situations. Emergencies can arise, for example, if a node crashes and the surviving network attempts to accommodate the resulting overload. Another cause can be a community of customers that suddenly present the network with an unanticipated level of demand. Diffusion methods for tracking the degradation of service and providing the capability of inquiring into which method of operation yields the most graceful degradation are especially useful.

Certain constructions give rise to processes that tend to be characterized by a $\underline{W}$ process. While today the quiescent operation of packet networks is not the condition of heavy load, we have cited above influences that motivate a long-term importance for $\underline{W}$ processes in the theory of computer networks. Sometimes, however, because it is extraordinarily difficult to analyze certain computer networks, a more precise model is replaced by an associated diffusion simply to gain tractability. Such heuristics are usually accompanied by measurements or simulations,[10,13] or else the diffusion can be a very useful "caricature"[14] of the real situation. Looking at the diffusion counterpart of a situation arising in practice in a context of light to moderate load, one can get an answer that, when interpreted in the original loading, gives the correct result.[15] Sometimes diffusion provides special insight[16] so that a new result in the realm of light to moderate traffic is first discovered by obtaining it in the heavy traffic range.

References 17 and 18 exhibited situations where state sensitive network behavior can be accommodated by diffusion models. It is well known that G/G models or state dynamic models are not amenable to exact analysis using traditional queueing theory approaches.

## III. COMMENTS ON CONVERGENCE TO DIFFUSIONS

In this section we very briefly describe the convergence of $Q(nt)/\sqrt{n}$ to $q(t)$. We aim our presentation to highlight those aspects of the

convergence that provide the backdrop for our main result. The material in this section is tutorial. The abstract convergence theory is adapted from Refs. 19 through 23, while the queueing network specific results cited can be found in Refs. 7, and 24 through 26.

We introduce some notation needed in the sequel. Superscripts, unless they are simple fractions, indicate the $n$th term in a sequence rather than exponentiation. By max $(q(t))$ we mean that the maximum is over the time interval $[0, T]$. We use $\approx$ to denote asymptotic equivalence for large values of a parameter ($f(n) \approx g(n)$ if $g(n)/f(n) \to 1$ as $n \to \infty$). For example, $n^{1/2} + n^{1/3} \log_2 n \approx n^{1/2}$.

### *3.1 Convergence of random processes*

The set $D^L$ of all vector-valued functions on a time interval $[0, T]$, where the components have left-hand limits and are right-continuous, is a very general set of functions for applications. Included are the diffusion sample paths, which are continuous, and the network queueing processes, which have piecewise constant paths. In the set $D^L$ a definition of distance between functions has been given. The metric is too involved to be discussed here, but for our purposes we will only encounter pairs $x_1$ and $x_2$ whose distance can be measured by $\rho(x_1, x_2) = \max \| x_1 - x_2 \|$, where $\| \ \|$ is the usual definition of length of an $L$-dimensioned vector.

By $\mathscr{D}^L$ we mean the set of all vector-valued random processes whose sample paths are in $D^L$. A notion of distance between processes has been defined so that meaning is given to the convergence ($\Rightarrow$) of a sequence of random processes to a limit process. The metric space $\mathscr{D}^L$ is a highly specialized topic, and we shall not define or discuss it in detail. However, we mention facts about $\mathscr{D}^L$ that are relevant to our result.

$\mathscr{D}^L$ accommodates both network queueing processes and diffusions and provides the setting for demonstrating convergence of queueing processes to diffusions. Since $D^L$ is a metric space, it makes sense to consider continuous real functions defined on $D^L$. If $f$ is a bounded continuous real function on $D^L$, then $f$ evaluated on the paths of an element of $\mathscr{D}^L$ is a random variable. Indeed, $f$ maps random functions into random numbers. If $\{x^n\}_1^\infty$ is a sequence of stochastic processes (points in $\mathscr{D}^L$), then $\{f(x^n)\}_1^\infty$ is a sequence of random variables. $x^n$ is said to converge to $x$ if for all bounded-continuous $f$ the distribution functions of $f(x_n)$ converge in the usual sense to the distribution function for $x$. This definition of convergence of general vector processes is considered to be *the* definition because of its significant practical value. This practical value partially stems from the fact that all the finite dimensional distributions of $q^n(t) = Q(nt)/\sqrt{n}$ converge to those for $q(t)$. Even more important is that the continuous functions

on the processes include those of interest in practice, and we have already mentioned that, by definition, the distributions of continuous functions converge. For example, a chief concern in the operation of a network is that queues not exceed certain levels (corresponding to overflow) or that maximum delay not be excessive. Thus, there is interest in $\max_t Q^n(t)$, and the maximum over a time interval can be shown to be a continuous function. Therefore, in situations where $q^n(t)$ is intractable and $q(t)$ is tractable, it is meaningful to use $q(t)$ to approximate the behavior of $q^n(t)$ for large $n$. Regarding tractability, we note that a great deal is known about Wiener processes, so that properties of general limits of queueing processes can often be easily obtained from previously derived results.

### *3.2 The G/G/1 queue*

We shall need this example in Section IV. For arrivals the mean rate is $\lambda$ per second, while $a$ denotes the interarrival variance. For the successive service times $\mu^{-1}$ is the mean and $s$ is the variance.

Let $A_{\#}(\tau_1, \tau_2)$ denote the number of arrivals in $(\tau_1, \tau_2)$, and let $D_{\#}(\tau_1, \tau_2)$ denote the number of departures. Define

$$a^n(t) = \frac{A_{\#}(0,nt) - \lambda nt}{\sqrt{n}}.$$

Using the central limit theorem and an asymptotic (large $n$) analysis, it is not difficult to show that $a^n(t)$ is distributed as $N(0, \lambda^3 at)$. The connection to the central limit theorem stems from the fact that $\{A_{\#}(0,nt) < k\} = \{A_1 + A_2 + \cdots A_k > nt\}$ (where $A_i$ are the interarrival times).

We can conclude the same sort of result for departures, but that is more difficult. Consider first an imaginary system where the departure process runs forever with no idle times. Then, for each $t$,

$$d^n(t) = \frac{D_{\#}(0,nt) - \mu nt}{\sqrt{n}} \text{ is distributed as } N(0, \mu^3 st).$$

For this imaginary system $a^n(t) - d^n(t)$ is distributed as $N(0,(\mu^3 a + \mu^3 s)t)$. These elementary results regarding $a^n(t)$ and $d^n(t)$ are suggestive of much more significant results regarding convergence in $\mathscr{D}^1$.

For the $\mathscr{D}^1$ convergence, we can let $\mu$ and $\lambda$ be functions of $n$ (and write $\mu^n$ and $\lambda^n$) so long as $\mu^n$ and $\lambda^n$ converge to constants and to each other so that

$$\lim_{n\to\infty} \sqrt{n}\,(\lambda^n - \mu^n) = \Delta$$

is a constant. This flexibility in $\mu^n$ will be useful in Section 4.2. For

each $n$, we have an arrival sequence $\{A_{jn}\}_1^\infty$ and a departure sequence $\{B_{nj}\}_1^\infty$ of i.i.d. random variables. For the imaginary system

$$\hat{q}^n(t) = \frac{\hat{Q}(nt)}{\sqrt{n}} = \frac{A_\#(0,nt) - D_\#(0,nt)}{\sqrt{n}}$$

$$= \frac{(A_\#(0,nt) - \lambda^n nt) + (D_\#(0,nt) - \mu^n nt)}{\sqrt{n}} + \frac{(\mu^n - \lambda^n)}{\sqrt{n}} nt,$$

we have that $\hat{q}^n(t)$ converges to $N(\Delta t, \lambda^3 a + \mu^3 s)$. The convergence is for each $t$. But much more is true, namely, it has been shown that $\hat{q}^n(t)$ converges to $W(\Delta t, \lambda^3 a + \mu^3 s)$ in $\mathscr{D}^1$. Now for $q^n(t)$ one needs to account for departures not occurring when the queue is empty. This is a substantial complication that has been dealt with in Refs. 7, and 24 through 26. The upshot is that $q^n(t) \Rightarrow \underline{W}(\Delta, \lambda^3 a + \mu^3 s)$.

For $\underline{W}$ (or $W$) one can write and solve a partial differential equation (called the Fokker-Planck equation) for the probability transition density associated with the system being in state $q'$ at time $t$ given that it was in state $q_0'$ at an earlier time $t_0$.[27] So long as $\Delta < 0$ the process $\underline{W}$ has a statistical equilibrium for which the state density is characterized by taking the limit of the probability transition density as $t \to \infty$.

In higher dimensions (networks) there is a vector version of the construction[7] in which the boundaries present even more difficulties than in the one-dimensional case.

### 3.3 Two results for future reference

We next cite two additional results* from the theory that we will use to derive our result on communication of status.

*Lemma 1: If $\{x^n\}_1^\infty$ and $\{y^n\}_1^\infty$ are each positive sequences of random processes in $\mathscr{D}^L$ with $x^n \Rightarrow x$, then if, for each $\epsilon > 0$,*

$$\lim_{n\to\infty} Pr\{\max_t \| x^n(t) - y^n(t) \| \geq \epsilon\} = 0,$$

*we conclude that $y^n \Rightarrow x$.*

*Lemma 2: If $\{x^n\}_{n=1}^\infty$ is a sequence of random processes in $\mathscr{D}^L$ with continuous paths, convergent to a process with continuous paths, then for each positive $\epsilon$ and $\eta$ there exist a $\delta$, $0 < \delta < T$, and an integer $n_0$ such that*

$$P\{\max_{|s-t|<\delta} \| x^n(s) - x^n(t) \| \geq \epsilon\} \leq \eta, \; n \geq n_0.$$

---

* Those familiar with the subject of convergence of probability measures will notice that Lemma 1 is a weakened form of the Converging Together Theorem (4.1 in Ref. 20), where the Skorokhod metric $\rho$ has been replaced by a simpler metric that dominates $\rho$. Lemma 2 is a tightness consequence (8.2 in Ref. 20).

## IV. DERIVATION OF THE MAIN RESULT

### *4.1 Approach*

Refer to Fig. 1. Were it not for the complication of the status packets on system 2, we would simply have a pair of independent G/G/1 systems. The queueing process $(q_1^n, q_2^n)$ would converge in $\mathscr{D}^2$ to a two-dimensional $\underline{W}$ process with each independent component a $\underline{W}$ process in $\mathscr{D}^1$, as already described in Section 3.2. We want to show that in the limit the status packets adequate to describe $q_1(t)$ in real time can be communicated on system 2 without perturbing the $\mathscr{D}^1$ limit, $q_2(t)$, from that for the case with no status packets.

So far we considered $T$ to be fixed, and consequently, we have written $\mathscr{D}^L$ rather than use the more complete notation $\mathscr{D}^L[0, T]$. The variable $T$ is needed to express mathematically the meaning of communication of status in real time. We mean that for each time $T$, from the status packets communicated up to time $T$, we can construct a sequence, $r^n(t)$, so that for each $\epsilon > 0$

$$\lim_{n\to\infty} P\{\rho(r^n(t), q_1^n(t)) > \epsilon\} = 0$$

(i.e., the distance between the process $q_1^n(t)$ and its approximation converges in probability to zero). From Lemma 1 we see that for each $T$ the convergence of $r^n(t)$ to $q_1(t)$ in $\mathscr{D}^1[0, T]$ is implied.

In Section 4.2 we derive a channel for status information that is asymptotically, for large $n$, of negligible rate in comparison to the total information rate of the communication resource. In the remaining two subsections, it is shown that the changes in $q_1^n(t)$ are such that there is enough rate to convey a sampled, clipped version of $q_1^n(t)$ that satisfies the above requirement for $r^n(t)$. The purpose of the clipping is to limit the number of bits per sample so as to meet the capacity limitation on the status channel.

### *4.2 Deriving a channel of negligible rate for status*

The normalization parameter $n$ can have different interpretations, as discussed in Section II. For definiteness in the presentation of the proof, we choose the perspective that the demand for service, $\Lambda^n$, is increasing and the capacity of the resource, $M^n$, is likewise increasing:

$$\Lambda^n = n\left(\mu - \frac{\Delta}{\sqrt{n}}\right)$$

and

$$M^n = n\mu,$$

where $\mu$ and $\Delta$ are constants.

From this viewpoint we see countervailing aspects of the nature of the limit. On the one hand, with $M^n$ increasing indefinitely, we see the opportunity of deriving a status channel of some significance that is vanishingly small relative to $M^n$. On the other hand, despite the fact that the limit $q_1(t)$ is insensitive to considerable changes in $Q_1^n(t)$, the limit has an extremely intricate structure that must be conveyed on the status channel. In fact, the $w(t)$ constituent of $q_1(t)$ is the epitome of a chaotic continuous random function—it is the indefinite integral of white Gaussian noise.[28]

*Proposition:* There exists a channel of negligible rate for status.

*Proof:* The units of $\Lambda^n$ and $M^n$ are packets per second. Say that there are on the average $B$ bits per packet so that the communication resource has a capacity of $M^nB$ bits per second. Since $M^n/n$ and $\Lambda^n/n$ correspond to $\mu^n$ and $\lambda^n$, respectively, as indicated in Section 3.2, we can alter $M^n$ and not perturb the $q_1(t)$ diffusion if we preserve the asymptotic behavior:

$$M^n/n \to \mu \quad \text{and} \quad \frac{M^n - \Lambda^n}{\sqrt{n}} \to \Delta$$

for large $n$.

This flexibility in $M^n$ enables us to derive a status channel. Specifically, we choose a channel for status that has rate

$$n^{1/3}\left(\frac{1}{2} + \theta\right)\log_2 n$$

bits per second, where $\theta$ is any positive number. We shall see the adequacy of this rate. If the transmission resources for channel number 2 provide the status channel, then

$$M^n = n\mu - \{n^{1/3}[(1/2) + \theta]\log_2 n\}/B$$

packets per second, and still $M^n \approx n\mu$ and

$$M^n - \Lambda^n = n\mu - n^{1/3}\frac{\left(\frac{1}{2} + \theta\right)\log_2 n}{B} - n\left(\mu - \frac{\Delta}{\sqrt{n}}\right)$$

$$= \sqrt{n}\,\Delta - \frac{n^{1/3}\left(\frac{1}{2} + \theta\right)\log_2 n}{B}.$$

From Section 3.2 we have that $q_2^n(t)$ converges to the same limit whether or not the server provides the derived channel.

We use the derived channel as follows to transmit information for

the required approximation to $Q_1^n(t)$: first we clip the process $Q_1^n(t)$ at the upper threshold $n^{\theta+(1/2)}$, and then we sample the result every $n^{-1/3}$ seconds. So, the set $\{0, 1, \cdots [n^{\theta+(1/2)}]\}$ is the range of the samples (where $[n^{\theta+(1/2)}]$ means the largest integer less than $n^{\theta+(1/2)}$).

We now show that in the limit of large $n$ the clipped and sampled replica of $Q_1^n(t)$ has the desired convergence property. Partition $[0, T]$ letting $I(i, n)$ denote the interval $[(i-1)/n^{1/3}, i/n^{1/3}]$, $i = 1, 2, \cdots$. In what follows, let $v^n(t)$ denote the piecewise constant process defined by $q_1^n(i/\sqrt{n})$ on $I(i, n)$. Let $r^n(t)$ be defined in precisely the same way as $v^n(t)$ except that $r^n(t)$ is truncated at $n^\theta$ so that $r^n(t) = \min(n^\theta, v^n(t))$. (Of course, if $q_1^n(t)$ has an upward or downward drift, it would seem advisable to define $v^n(t)$ and $r^n(t)$ to have a linear slope between samples matching the trend in $q_1^n(t)$. However, the piecewise constant $r^n(t)$ and $v^n(t)$ are adequate for us and connect easily to existing results we want to use.) The successive values of $r^n(t)$ are transmitted over the derived channel. The function $v^n(t)$ is introduced to help establish the convergence of $r^n(t)$ to $q_1(t)$.

Using the above proposition we can now establish convergence. We must demonstrate the following.

*Theorem: For each* $\epsilon > 0$

$$\lim_{n\to\infty} \Pr\{\max_t |q_1^n(t) - r^n(t)| > \epsilon\} = 0. \tag{1}$$

*Proof:* It is convenient to note two ways in which $r^n(t)$ can fail to track $q_1^n(t)$ to within $\epsilon$:

A. $q_1^n(t)$ could take on large values sufficiently beyond the peak value $n^\theta$ of the tracking process.

B. $q_1^n(t)$ could, on some $I(i, n)$, depart too much from the sample value approximation.

We shall see that both ways occur with sufficiently small probability. It is evident that

$$\{\max_t |q_1^n(t) - r^n(t)| > \epsilon\}$$

is contained in

$$\{\max_t q_1^n(t) > n^\theta\} \cup \{\max_t |q_1^n(t) - v^n(t)| > \epsilon\} = A^n \cup B^n,$$

where $A^n$ and $B^n$ are defined in the obvious way. We next show $\lim_{n\to\infty} P\{A^n\} = \lim_{n\to\infty} P\{B^n\} = 0$. From these limits (1) follows.

To see that $P\{A^n\} \to 0$, use the fact that $q_1^n(t) \Rightarrow w(t)$ so that $\max_t q_1^n(t) \to \max_t \underline{w}(t)$. In terms of distributions we have that for each number $y$

$$\lim_{n\to\infty} P\{\max_t q_1^n(t) > y\} = P\{\max_t \underline{w}(t) > y\}.$$

Now

$$\lim_{n\to\infty} P\{\max_t q_1^n > n^\theta\} < \lim_{n\to\infty} P\{\max_t q_1^n(t) > C\}$$

$$= P\{\max_t \underline{w}(t) > C\}, \quad (2)$$

where $C$ is any number. So the right-hand side of (2) can be made arbitrarily small. Thus we have obtained the desired result for $A^n$.

Next we show that $\lim_{n\to\infty} P(B^n) = 0$; that is, for each $\epsilon > 0$

$$\lim_{n\to\infty} P\{\max_t | q_1^n(t) - v^n(t) | \geq \epsilon\} = 0. \quad (3)$$

First let $\hat{q}_1^n(t)$ denote the continuous variant of $q_1^n(t)$ formed by connecting the consecutive points of discontinuity in the graph of $q_1^n(t)$ by line segments. By construction $\max_t | \hat{q}_1^n(t) - q_1^n(t) | \leq (1/\sqrt{n})$. So by Lemma 1, $\hat{q}_1^n(t) \Rightarrow q_1(t)$. Employing Lemma 2, we have that for each $\eta > 0$ there exists a $\delta > 0$, so that for

$$P\{\max_{|s-t|<\delta} | \hat{q}_1^n(t) - \hat{q}_1^n(s) | \geq \epsilon\} < \eta$$

for sufficiently large $n$. Therefore,

$$P\{\max_{|s-t|\leq n^{-1/3}} | \hat{q}_1^n(t) - \hat{q}_1^n(s) | \geq \epsilon\} < \eta$$

for sufficiently large $n$, or what is the same, the limit of this sequence of probabilities is zero. Rewriting the maximum in terms of $q_1(t)$, we have

$$P\{e^n + \max_{|s-t|\leq n^{-1/3}} | q_1^n(t) - q_1^n(s) | \geq \epsilon\} \to 0,$$

where the magnitude of the error term, $e^n$, cannot exceed $2/\sqrt{n}$. Since

$$\lim_{n\to\infty} P\{\max_{|s-t|\leq n^{-1/3}} | q_1^n(t) - q_1^n(s) | \geq \epsilon - e_n\} = 0,$$

by set containment it follows that for each fixed $\epsilon' > \epsilon$

$$P\{\max_t | q_1^n(t) - v^n(t) | \geq \epsilon'\} \to 0.$$

The containment stems from taking the maximum over a smaller set and from the fact that the threshold $\epsilon'$ eventually exceeds $\epsilon - e^n$. Now (3) follows because $\epsilon$ is arbitrary.

## V. DISCUSSION

### *5.1 Immediate extension*

We have proven a simple form of the result that in heavy traffic, status can be conveyed unintrusively. The result extends immediately to much more general network settings. It is not difficult to go beyond

the simple form of the result and conclude that status can be conveyed unintrusively throughout an entire network such as Reiman and Harrison's generalization of a Jackson network. One simply derives as many parallel status channels as is necessary on each communication link.

### 5.2 Similarities

While heavy traffic analysis of computer communication networks is more intricate than established analytical techniques for electrical networks, there are some striking parallels. For example, as with electrical networks, one writes a differential equation[7] (the aforementioned intricacy stems from the fact that in computer networks it is a partial-differential equation: the Fokker-Planck equation). There is interest in both transient and steady-state analysis. As with voltage and current there is an extremely simple steady-state relationship between the two dependent variables of most interest, queue size, and delay.[7,29] The main result of this paper can be considered to add to this list of similarities by deriving what amounts to a sampling theorem.

### 5.3 Future work

The availability of status information that was established here is only one aspect of the broader subject of network control. This availability prompts the question of how should status information be used to optimally control $q(t)$ under certain performance criteria?

Examples already appear in the literature[17,18] that demonstrate significant improvements using status information to guide the evolution of a queueing network in heavy traffic. One of the established examples involves that of the case of a Poisson arrival process where each arrival has the option of joining one of $K$ queues. The $K$ queueing systems have i.i.d. exponential service times. Two systems are contrasted. In the first, status information is used and the arrival joins the queues offering the least expected delay at the time of arrival. In the second system the arrival is blind to status information and randomly selects a queue. In heavy traffic the performance of the first system is superior to that of the second by a factor of $K$ in mean queue size and delay and, more importantly, a factor of $K$ in the tail exponent of the equilibrium distribution for queue size and delay.

The published examples seem part of a more general theory that would allow more relaxed assumptions on arrivals and services and would address the question of how status information should be transferred and used. The result of this paper needs to be shown in the context of $q(t)$ being controlled with the status information. We remark that the proof we have given utilizes very little of the structure

of the approximating queueing processes and appears to allow great flexibility of form for $Q^n(t)$.

For a Reiman-Harrison-type of network, but with state dynamic routing, the problem of finding the optimum control to minimize the maximum (over source-sink pairs) average delay is a very challenging one.

Another layer of difficulty is introduced if one includes fixed delays in status information to account for processing or propagation.

## VI. ACKNOWLEDGMENT

The author gratefully acknowledges helpful discussions with Marty Reiman, Jack Salz, Ward Whitt, and Clark Woodworth. Marty Reiman suggested that using Lemma 2 might simplify an earlier (more cumbersome and restrictive) derivation of the main result, and indeed he was correct.

## REFERENCES

1. L. Kleinrock, *Queueing Systems, Computer Applications,* Vol. 2, New York: Wiley, 1976.
2. H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,* Reading, MA: Addison-Wesley, 1978.
3. M. Schwartz, *Computer-Communication Network Design and Analysis,* Englewood Cliffs, NJ: Prentice-Hall, 1977.
4. K. M. Chandy and C. H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computing Systems," Comput. Surv., *10,* No. 3 (September 1978), pp. 281–317.
5. H. Kobayashi and A. G. Konheim, "Queueing Models for Computer Communication System Analysis," IEEE Trans. Commun., *COM-25,* No. 1 (January 1975), pp. 2–29.
6. J. M. Holtzman, "Routing Updates in Packet Switched Networks," in *Computer Performance,* K. M. Chandy and M. Reiser, Eds., New York: North-Holland, pp. 537–46, 1977.
7. M. Reiman, "Open Queueing Networks in Heavy Traffic," Math. Oper. Res., *9* (August 1984), pp. 441–58.
8. S. Karlin and H. M. Taylor, *A First Course in Stochastic Processes,* New York: Academic Press, 1975.
9. S. Karlin and H. M. Taylor, *A Second Course in Stochastic Processes,* New York: Academic Press, 1981.
10. J. P. Lehoczky and D. P. Gaver, "Gaussian Approximation to Service Problems: A Communications System Example," J. Appl. Probab., *13,* No. 4 (December 1976), pp. 768–80.
11. L. Kleinrock, Computer Networks Seminar, Washington, D. C., August 1976.
12. S. Halfin and W. Whitt, "Heavy-Traffic Limits for Queues With Many Exponential Servers," Oper. Res., *29,* No. 3 (May–June 1981), pp. 567–88.
13. E. Gelenbe and I. Mitrani, *Analysis and Synthesis of Computer Systems,* New York: Academic Press, 1980.
14. F. B. Knight, *"Essentials of Brownian Motion and Diffusion,"* Math. Surv., No. 18, American Mathematical Society, RI, 1981.
15. D. P. Heyman, "An Approximation for the Busy Period of the M/G/1 Queue Using a Diffusion Model," J. Appl. Probab., *11,* No. 1 (March 1974), pp. 159–69.
16. G. J. Foschini and B. Gopinath, "Sharing Memory Optimally," IEEE Trans. Commun. Technol., *COM-31,* No. 3 (March 1983), pp. 352–60.
17. G. J. Foschini and J. Salz, "A Basic Dynamic Routing Problem and Diffusion," IEEE Trans. Commun. Technol., *COM-26* (March 1978), pp. 320–7.
18. G. J. Foschini, "On Heavy Traffic Diffusion Analysis and Dynamic Routing in Packet Switched Networks," in *Computer Performance,* K. M. Chandy and M. Reiser, Eds., New York: North-Holland, 1977, pp. 499–514.

19. Yu. V. Prokhorov, "Convergence of Random Processes and Limit Theorems in Probability Theory," Theory Probab. Appl., *1,* No. 2 (1956), pp. 157–214.
20. P. Billingsly, *Convergence of Probability Measures,* New York: Wiley, 1968.
21. P. Billingsly, "Weak Convergence of Measures Applications in Probability," SIAM, Philadelphia, 1971.
22. K. R. Parthasarathy, *Probability Measures in Metric Spaces,* New York: Academic Press, 1967.
23. S. R. S. Varadhan, *Stochastic Processes,* The Courant Institute, New York University, 1968.
24. M. Reiman and M. Harrison, "Reflected Brownian Motion in an Orthant," Ann. Probab., *9,* No. 2 (1981), pp. 302–8.
25. A. A. Borovkov, "Some Limit Theorems in the Theory of Mass Service, Theory Probab. Appl., *10,* No. 4 (1965), pp. 550–65.
26. D. L. Iglehart and W. Whitt, "Multiple Channel Queues in Heavy Traffic, II," Advanc. Appl. Probab., *2,* No. 1, pp. 355–69.
27. D. R. Cox and H. D. Miller, *The Theory of Stochastic Processes,* London: Chapman and Hall, 1972.
28. E. Wong, *Stochastic Processes in Information and Dynamical Systems,* New York: McGraw-Hill, 1971.
29. G. J. Foschini, "Equilibria for Diffusion Models for Pairs of Communicating Computers-Symmetric Case," IEEE Trans. Inform. Theory, *IT-28,* No. 2 (March 1982), pp. 273–84.

## AUTHOR

**Gerard J. Foschini,** B.S.E.E., 1961, Newark College of Engineering, Newark, NJ; M.E.E., 1963, New York University, New York; Ph.D. (Mathematics), 1967, Stevens Institute of Technology, Hoboken, NJ; AT&T Bell Laboratories, 1961—. Mr. Foschini initially worked on real-time program design. For many years he worked in the area of communication theory. In the spring of 1979 he taught at Princeton University. Mr. Foschini has supervised planning the architecture of data communications networks. Currently, he is involved with digital radio and optical communication research. Member, Sigma Xi, Mathematical Association of America, IEEE, New York Academy of Sciences.

AT&T Technical Journal
Vol. 64, No. 2, February 1984
Printed in U.S.A.

# A Note on Discrete Representation of Lines

By M. D. McILROY*

(Manuscript received March 30, 1984)

In raster graphics a line must be drawn as a "discrete segment", a set of integer grid points that lie close to the line. Equivalence classes of identically drawn lines are described in terms of Farey series; this treatment notably simplifies previous work of Dorst and Smeulders. A log $n$ algorithm serves to identify a line's equivalence class. Using it to choose among precomputed $n$-pixel discrete segments, we may draw lines in $n$-pixel blocks rather than the customary single-pixel steps.

## I. INTRODUCTION

The problem of approximating a straight line by points of an integer grid may be reduced to that of finding the extreme grid points in a closed half-plane.[1,2] By reflections and translations the problem may be standardized to the form: For each integer $x$ maximize $y$ over the integers subject to

$$y \leqslant mx + b,$$

where $m$ and $b$ are restricted to the region

$$S = \{(m, b) \mid 0 \leqslant m \leqslant 1, 0 \leqslant b < 1\}.$$

An equivalent formulation is: For each integer $x$ find the unique integer $y$ that satisfies

$$mx + b - 1 < y \leqslant mx + b.$$

An explicit solution is $y = \lfloor mx + b \rfloor$, where $\lfloor b \rfloor$ denotes the largest

* AT&T Bell Laboratories.

integer that does not exceed $b$. The conventions of the preceding equations will be followed throughout:

$n, p, q, x, y$ are integers,

$s, t, u, v, w$ are rationals, and

$b, m$ are reals.

In particular, range restrictions, such as $0 \leqslant x \leqslant n$ and $0 \leqslant m \leqslant 1$, must be understood in the domain of the variables involved.

A *discrete segment of length n* is a set $L(m, b, n)$ given by

$$L(m, b, n) = \{(x, \lfloor mx + b \rfloor) \mid 0 \leqslant x \leqslant n\}.$$

Usually $n$ is fixed, so we may speak simply of a *discrete segment.*

Section II investigates the equivalence classes of lines that are approximated by identical discrete segments. These equivalence classes induce an interesting polygonal pattern in the space of parameters $(m, b)$. Farey sequences abound in the pattern, which is accordingly dubbed a "Farey fan". To each region, or "facet", of a Farey fan there corresponds a distinct discrete segment. Thus the facet in which the parameters of a given line lie tells what discrete segment approximates that line. A canonical characterization for facets and an $O(\log n)$ algorithm for locating the facet for a given line are developed in Section III. The algorithm may be used in raster graphics to choose among precomputed discrete segments, so that lines may be drawn by block transfers in $n$-pixel chunks. The main results were first given by Dorst and Smeulders.[3] The contributions of this paper are (1) recognition of the role of Farey series, (2) simplified analysis based on the Farey fan, and (3) algorithms.

## II. FAREY FANS

To each discrete segment $L$ of length $n$ there corresponds an equivalence class $C(L)$ of points in $S$ given by

$$C(L) = \{(m, b) \mid (m, b) \in S \text{ and } L(m, b, n) = L\},$$

or

$$C(L) = \{(m, b) \mid mx + b - 1 < y \leqslant mx + b, (x, y) \in L\}.$$

Defined by linear inequalities, the equivalence classes are polygonal subregions of $S$, called *facets*. For each point $(x, y)$ in $L$ there is a pair of parallel bounding *rays*, $R(x, y)$ and $R(x, y + 1)$, where

$$R(x, y) = \{(m, b) \mid b = -xm + y\}, \qquad 0 < y \leqslant x \leqslant n.$$

$R(x, y)$ has slope $-x$, $b$-intercept $y$, and $m$-intercept $y/x$. The $m$-

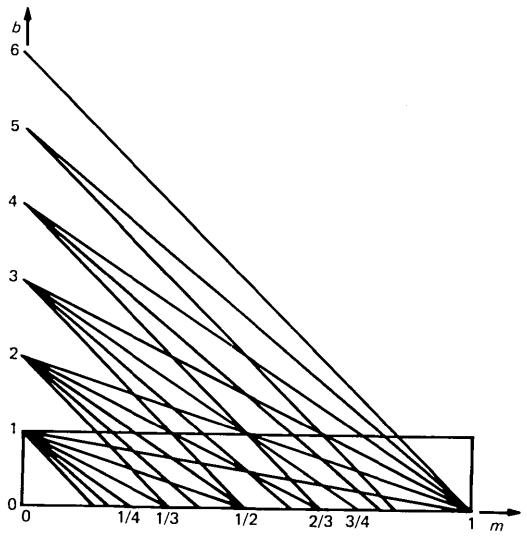



Fig. 1—Farey fan of order 6. Heavy lines delimit region $S$.

intercepts $\{y/x \mid 0 \leqslant y \leqslant x, 1 \leqslant x \leqslant n\}$ constitute a Farey series.* (See Chapter 3 of Ref. 4.) The rays make a pattern like Fig. 1. The part of this figure that is contained in the closure of $S$ is called a *Farey fan of order n*, often shortened simply to "Farey fan".

A duality holds between the $m$-$b$ space of Fig. 1 and the original $x$-$y$ space where we are approximating. To every point $(x, y)$† in original space there corresponds a line $b = -xm + y$ in dual space, whose points $(m, b)$ are the slope and intercept parameters of members of a pencil of lines through $(x, y)$ in original space. Similarly, to every point $(m, b)$ of dual space there corresponds a line $y = mx + b$ in original space, whose points $(x, y)$ are slope-intercept parameters of members of a pencil of lines through $(m, b)$ in dual space.

A side of a facet is either a side of $S$ or a segment of a ray. To include the top and bottom sides of $S$ in the Farey fan, we admit $R(0, 0)$ and $R(0, 1)$ as well:

$$R(x, y) = \{(m, b) \mid b = -xm + y\}, \qquad \begin{cases} 0 \leqslant y \leqslant x \leqslant n \\ \text{or } x = 0,\ y = 1. \end{cases} \tag{1}$$

---

* The Farey series of order $n$ is the ordered set of rational numbers $p/q$, $1 \leqslant q \leqslant n$, in the interval $[0, 1]$. For example, the Farey series of order 7 is:

$$\frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1}.$$

† In this paragraph $x$ and $y$ are real variables.

The vertex where two rays, $R(x, y)$ and $R(x', y')$, $x > x'$, intersect is the point

$$(m, b) = \left(\frac{y - y'}{x - x'}, \frac{xy' - x'y}{x - x'}\right). \tag{2}$$

To simplify the analysis further, we extend the domain of $R(x, y)$ again, to include $0 \leq x \leq n$ and all integer $y$. None of the extra rays passes through $S$, so this extension has no effect on the facets. In the strip $0 \leq m \leq 1$, the $m$-coordinates of the intersections of $R(x, y)$ with other rays are then given by rational numbers

$$m = \frac{p}{q}, \qquad 0 \leq p \leq q, \qquad 1 \leq q \leq \max(x, n - x). \tag{3}$$

Taken in arithmetic order, these values constitute a segment of a Farey series of order $\max(x, n - x)$. From (1) any ray $R(x, y)$ must cross $m = p/q$ at $b = y - px/q$, that is, at a multiple of $1/q$. We have proved

*Theorem 1: The abscissas of intersections of a given ray of a Farey fan with other rays constitute a Farey series.*

*Corollary 1: The abscissas of adjacent vertices of a facet are adjacent members of a Farey series. Moreover, if the abscissa of a vertex is $p/q$, the ordinate is a multiple of $1/q$.*

Figure 1 suggests that every facet is triangular or quadrilateral. This observation is true in general:

*Theorem 2: A facet of a Farey fan has at most four sides.*

*Proof, by induction:* The statement is obviously true for a Farey fan of order 1. Assume it true for Farey fans of order $n - 1$, and assume also that all four-sided facets are *diamonds* that have two *middle vertices* with identical $m$-coordinates. Recalling that all rays have nonpositive slope, we see that each facet of the Farey fan of order $n - 1$ has one of the shapes exemplified in Fig. 2. Each shape will be considered separately.

Figure 2a shows a triangular facet bounded by $m = 0$. The only new line that crosses (enters the interior of) the facet in question is as shown in Fig. 3a. It divides the facet into two triangles.

Figure 2b shows a general triangular facet with long side upward. Any new ray that crosses the facet has a slope more negative than that of any of the sides and hence must cross side $uw$ (Fig. 3b). Now $u$ and $w$ are adjacent members of a Farey series of some order $q$. From eq. (3) $u$, $s$, $w$ must be members of a Farey series of order $q + 1$. Since at most one new member can appear between any two adjacent members of a Farey series in going from order $q$ to order $q + 1$, at most one new ray can cross side $uw$. Thus $u$, $s$, $w$ are consecutive

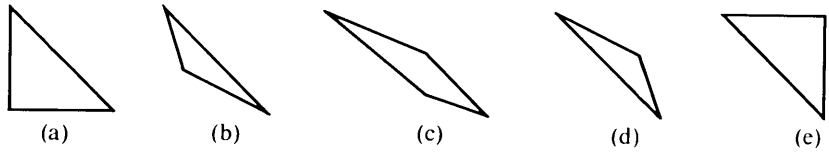



Fig. 2—Possible facets of a Farey fan: (a) triangular facet bounded by $m = 0$; (b) general triangular facet with long side upward; (c) diamond facet; (d) general triangular facet with long side downward; (e) triangular facet bounded by $m = 1$.

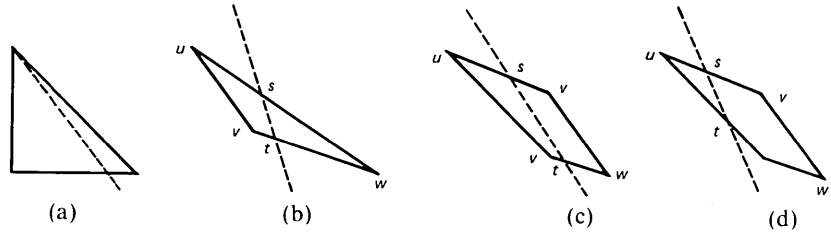



Fig. 3—Adding lines to a Farey fan of order $n - 1$ to make a Farey fan of order $n$. Vertices are labeled with their $m$-coordinates.

members of some Farey series. By Corollary 1, there is, for each pair drawn from $\{u, v, w\}$, a Farey series in which that pair is adjacent. Hence $u$, $v$, $w$ are also consecutive members of some Farey series. Consequently $s = v$. The facet has been partitioned into one triangle and one diamond.

Figure 2c shows a diamond facet. A new ray that crosses two opposite sides as in Fig. 3c would join points $s$ and $t$, where $s < v < t$ and where $v$ belongs to a lower-order Farey series than do either $s$ or $t$. Thus $s$ and $t$ cannot be adjacent members of any Farey series, contradicting Corollary 1. A ray that crosses two adjacent sides (Fig. 3d) would introduce new terms into both Farey series. Since the flanking terms, $u$ and $v$, are the same in both series, the new terms must be the same: $s = t$. But this is impossible because the new ray would have infinite, not negative, slope. Because a new ray that crosses a diamond can cross neither two opposite nor two adjacent sides, it must pass through one of the two middle vertices and partition the diamond into a triangle and a diamond.

Figures 2d and e are analogous to 2b and a.

In every case the addition of rays to convert a Farey fan of order $n - 1$ into a Farey fan of order $n$ yields triangular or diamond facets. The induction is complete.

## III. IDENTIFYING FACETS

Since distinct discrete segments of length $n$ correspond one-to-one with facets of the Farey fan of order $n$, the Farey fan makes a natural


Fig. 4—A ladder from Fig. 1.

index to discrete segments. As Sproull has suggested,[5] one may use such an index to generate arbitrary approximate lines in gulps of length $n$, possibly beating the usual point-by-point methods. The words of a display memory make natural gulps.

To look entries up in the index, we need a way to identify in which facet a given $(m, b)$ pair lies. Let $f_i$ be the $i$th term of the Farey series of order $n$. Since all ray intersections in the Farey fan occur at points whose $m$-coordinates belong to the series, in each interval $f_i \leqslant m \leqslant f_{i+1}$ the fan is a simple *ladder* of *rungs* that never cross each other in the interior (Fig. 4). This observation suggests Method M below, a workable method for locating the facet in which a given $(m, b)$ lies.

Method M:

1. Find the interval to which $(m, b)$ belongs.
2. In that ladder locate the highest rung that lies on or below $(m, b)$.
3. From that rung read off a precomputed discrete segment of length $n$.

An easy way to do Step 1 is binary search in the Farey series, or, since Farey series are fairly smooth, interpolation search. Step 2 can be done by binary search in the ladder. The total running time is $O(\log n)$: Step 1 searches a table of $O(n^2)$ Farey series members (see Ref. 4, Theorem 231); Step 2 searches a table of $n$ rungs (Theorem 3).

The suggested implementation of Method M requires preprocessing and storage for $O(n^2)$ intervals times $n$ rungs of tabulated information. This information is dispensable. Step 1, identification of an interval, can be done by an algorithm of Papadimitriou in $O(\log n)$ time.[6] The equation of a rung through $(p/q, r/q)$ for use in Step 2 can be generated by substituting $(m, b) = (p/q, r/q)$ in (1):

$$qy - px = r,$$

and solving for $x$ and $y$ by an extended Euclidean algorithm in $O(\log n)$ time.[7] One solution is required at each of the $O(\log n)$ stages of binary search. Thus facets can be identified in $O(\log^2 n)$ time, single-shot, with no preprocessing. Preprocessing, however, reduces the prob-

lem to simple binary search; it is the technique of choice for modest, fixed values of $n$.

Any scheme for uniquely identifying facets may be used for a canonical description for discrete segments. Before we demonstrate one, it will be helpful to have another theorem.

*Theorem 3: A ladder in a Farey fan of order n has exactly one rung of each slope $-x$, $1 \leqslant x \leqslant n$. Moreover, adjacent rungs that meet at a ladder edge, $m = p/q$, where $gcd(p, q) = 1$, have slopes that differ by q.*

*Proof:* By eq. (1) a rung of slope $-x$ must meet the vertical line $m = m_0$ at an ordinate $b$ such that there exists an integer $y$ satisfying

$$0 \leqslant b = -xm_0 + y < 1. \tag{4}$$

Equation (4) reflects the fact that $S$ is open along the top edge, $b = 1$. The closure of a rung, however, may meet that edge; we replace $<$ with $\leqslant$ in (4) to capture such rung ends. The solution

$$y = \lceil xm_0 \rceil \tag{5}$$

satisfies the constraints of (1) since $0 \leqslant m_0 \leqslant 1$. It is unique unless $xm_0 = 0$, in which case $y = 0$ and $y = 1$ are both solutions. But only one of two parallel rungs incident on $(m_0, 0)$ and $(m_0, 1)$ actually lies in $S$. The first claim of the theorem is established.

To prove the second claim, let two rungs $R(x, y)$ and $R(x' y')$, $x > x'$, meet at $m = p/q$. From eq. (2),

$$m = \frac{p}{q} = \frac{y - y'}{x - x'},$$

showing that $q \mid x - x'$. Thus we must have

$$y = y' + kp, \qquad x = x' + kq \tag{6}$$

for some positive integer $k$. Substitution in (2) yields

$$b = \frac{qy' - px'}{q}.$$

Since this value of $b$ is independent of $k$, all rungs $R(x' + kq, y' + kp)$ meet $R(x' y')$ at the same point. If any value of $k$ in (6) yields values of $x$ and $y$ that satisfy (1), then $k = 1$ certainly does. Thus, if $-x'$ is the larger slope of two rungs incident on a vertex at $m = p/q$, there is another rung through that vertex with slope $-(x' + q)$, and no rung with an intermediate slope. The second claim has been proved.

We now give the Dorst-Smeulders characterization of a discrete segment. Here the notion of "middle vertex" is extended from diamonds to all facets, meaning any but the upper left and lower right vertices.

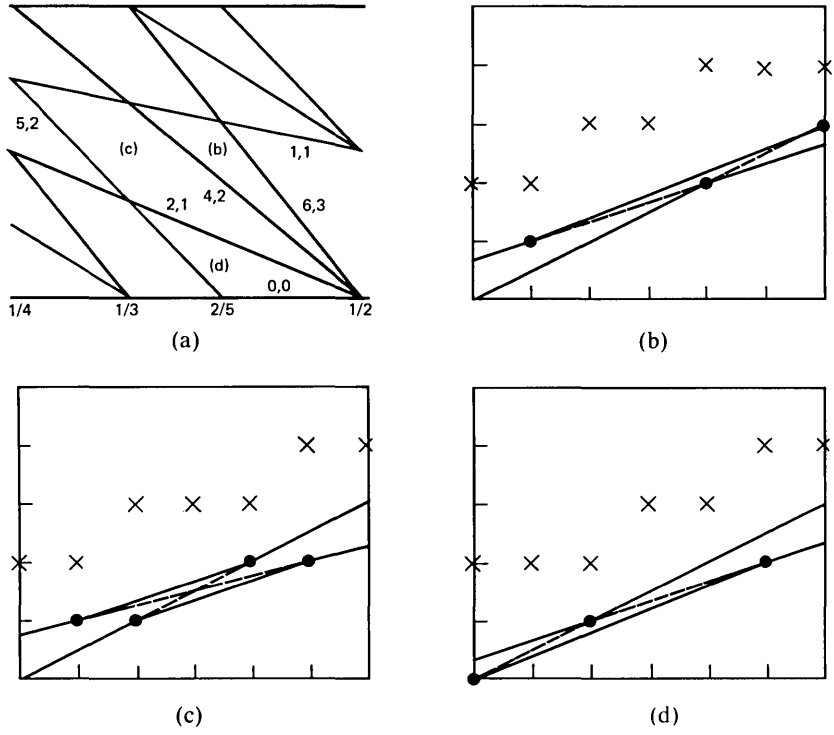



Fig. 5—(a) A fragment of Fig. 1. Certain rays are labeled with negated slope $x$ and intercept $y$. (b) through (d) Frontier positions of lines in original space that are dual to facets labeled (b) through (d) in (a). The coordinates of breakpoints (●) are the same as the labels of the rays in (a) that are dual to the breakpoints. The discrete segment (X) for each facet is shown shifted up two units.

*D-S Representation: A facet of a Farey fan and its corresponding discrete segment may be characterized by four integers, n, q, p, and x, where*

*n is the length of the discrete segment,*

*$p/q$, where $\gcd(p, q) = 1$, is the abscissa of a middle vertex, and*

*$-x$ is the slope of the lower of the two rungs incident on the lower right vertex.*

Theorem 3 justifies the representation: the slope, $-x$, of the lower rung serves to distinguish among the various facets that have middle vertices on $m = p/q$.

The four parameters have simple interpretations. Each facet is a convex combination of its vertices, which represent limiting positions of lines in the original space. Figure 5 shows the lines in those limiting

positions; they form broken-line upper and lower *frontiers*. Any line that falls between the frontiers and does not touch the upper one will be approximated by the same discrete segment. If a frontier has two breakpoints, the slope of the line joining them is $p/q$.

Exactly one line of slope $p/q$ can touch the lower frontier. Parameter $x$ specifies the $x$-coordinate of the leftmost point where it touches; by (5) the $y$-coordinate is $\lceil px/q \rceil$. This line, with slope $p/q$ and intercept $\lceil px/q \rceil - px/q$, may be taken to be the canonical representative of the equivalence class of all lines approximated by the same discrete segment. Because the canonical representative has rational slope, the discrete approximation to its infinite extension is invariant under translation by the vector $(q, p)$. Hence the first differences ("chain-code," in computer graphics parlance) of the discrete approximation are periodic with period $q$. Dorst and Smeulders took this periodicity to be the defining property of $q$.

## IV. DISCUSSION

Rob Pike first brought the motivating question to my attention: How may one quickly identify the discrete segment of length $n$ that approximates a given line? The apparent complexity of a quick sketch of the Farey fan discouraged further consideration—a cautionary experience. In this subject one well-drawn picture is worth a pot of algebra. In fact, computer graphics helped prove Theorem 2: the induction hypothesis came from playing with Theo Pavlidis's PED graphics editor to make a Farey fan.[8] The conclusion of Theorem 2 fairly shrieks from Fig. 1, and Theorem 1 isn't far behind. In the absence of well-drawn diagrams, however, it took the work of Dorst and Smeulders to reveal the basic simplicity of the diagram. Their treatment involved intricate discrete analysis in the original space. The route through dual space, signposted by Farey series, turns out to be much easier. Moreover, the newer analysis solves the original problem: Method M may be used to select, from among a precomputed list, the correct discrete segment to approximate any given line. The method provides an exact alternative to a heuristic proposed by Sproull.[5]

The search scheme in Method M may be recognized as a variation on Shamos's slab test for the inclusion of a point in a polygon.[9,10] The asymptotic performance of Method M is the same as that of Lipton and Tarjan for locating a point in a general polygonal tiling.[11] The single-shot algorithm, at $O(\log^2 n)$, is much better than general single-shot methods, which require $O(n^3)$ time in this special case.[10]

I am indebted to Jon Bentley for pointers from his mental encyclopedia of computational geometry.

## REFERENCES


1. J. E. Bresenham, "Algorithm for Computer Control of a Digital Plotter," IBM Syst. J., *4*, No. 1 (1965), pp. 25–30.
2. W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, New York: McGraw-Hill, 1979.
3. L. Dorst and A. W. M. Smeulders, "Discrete Representation of Straight Lines," IEEE Trans. Pattern Analysis and Machine Intelligence, *PAMI-6* (July 1984), pp. 450–62.
4. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, London: Oxford University Press, 1971.
5. R. F. Sproull, "Using Program Transformations to Derive Line-Drawing Algorithms," ACM Trans. Graphics, *1* (October 1982), pp. 259–73.
6. C. H. Papadimitriou, "Efficient Search for Rationals," Inform. Processing Letters, *8* (January 1979), pp. 1–4.
7. D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, Reading, MA: Addison-Wesley, 1969, Section 4.5.2.
8. T. Pavlidis, "PED User's Manual," Computing Science Tech. Report #110, AT&T Bell Laboratories, Murray Hill, NJ (March 1984).
9. J. L. Bentley and W. Carruthers, "Algorithms for Testing the Inclusion of Points in Polygons," Proc. Eighteenth Annual Allerton Conf. Commun., Control, and Computing, EE-CSL, University of Illinois, Urbana-Champaign, October 1980, pp. 11–19.
10. M. I. Shamos, *Computational Geometry*, Thesis, Carnegie-Mellon University, Pittsburgh (May 1978).
11. R. J. Lipton and R. E. Tarjan, "Applications of a Planar Separator Theorem," Proc. Eighteenth Annual IEEE Symp. Foundations of Computer Science, Providence, October 1977, pp. 162–70.


## AUTHOR


**M. Douglas McIlroy,** BEP, Cornell University; Ph.D. (Applied Mathematics), The Massachusetts Institute of Technology; AT&T Bell Laboratories, 1958—. Since 1965 Mr. McIlroy has been Head of the Computing Techniques Research Department at AT&T Bell Laboratories; he was a visiting lecturer at Oxford University in 1967–68. Particularly interested in programming languages and software tools, Mr. McIlroy did early work in macroprocessors that led to the idea of extensible languages, participated in the original design of PL/I and *Multics*,* made numerous contributions to the *UNIX*™ operating system, including the concept of "pipes", and devised the first program for converting English text to synthetic speech in real time. He has served on various advisory panels to federal and state bodies, and has served the ACM as national lecturer, an editor for several journals, and the Turing award chairman.


---

* Registered trademark of Honeywell Information Systems, Inc.

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Models for Configuring Large-Scale Distributed Computing Systems

By B. GAVISH*

(Manuscript received October 13, 1983)

This paper presents a model for designing the architecture of a distributed computing system of the type used to support the management and control activities of a large corporation. The input to the design process consists of two major data components describing the inputs and outputs to the information system and the relationships between them. Based on this information and the structure of communication and processing costs, an optimization model is formulated. It aggregates transactions in distributed databases, selects the locations in which those databases will be placed, assigns data sources to those databases, and selects for each report the report generation location. The problem is formulated as a combinatorial optimization problem and procedures are developed for computing lower bounds on the value of the optimal solution and heuristics for generating good feasible solutions for the problem. The procedures were tested on several examples and have generated good initial designs. Computational examples are presented to design problems including organizations with a hierarchical structure.

## I. INTRODUCTION

Providing data collection, storage, and retrieval capabilities to industrial service and governmental organizations is a primary responsibility of management information systems and of operations systems in those organizations. Traditionally, such services have been provided by centralized computing systems, using large computers with cen-

* University of Rochester, N.Y. Part of this research was done while Mr. Gavish was an employee at AT&T Bell Laboratories.

tralized databases and communication networks. Due to the increased dissatisfaction of users with the quality of services provided by centralized systems, distributed computing systems are evolving as superior substitutes to many of those large centralized systems, for providing data storage and retrieval services. Distributed computer systems offer many potential advantages to those organizations. Those advantages include:

- Reduction in data processing costs by trading communications for processing costs
- Easier and smoother introduction of new hardware and software into the system
- Shorter response time
- Direct user control over computing resources to allow for setting up their own scheduling priorities
- Establishment of site autonomy and direct user responsibility and control over their application programs, and the quality and timeliness of the data that they generate and collect.

Distributed computing systems present system designers with many challenges, some of which already have been encountered in centralized computing systems and were easier to solve in a centralized setting, while others are new and unique to a distributed environment. These include technical issues such as: system integrity; concurrency control;[1-4] query parsing and decomposition;[5] addressing, naming, and directory management; backup recovery;[6] and security.[7] Other issues involve managerial and design problems that are faced by the managers and designers of those systems. These include: decisions on the allocation of files and databases to computer locations;[8-12] placement of processors and allocation of databases to those processors;[13-15] query optimization;[16-18] directory assignment; and design and analysis of the communication subnetworks.[19]

The potential promised by distributed systems and the importance of the above questions have attracted the attention of many researchers. Several models and procedures have been developed for answering some of those design issues. All of the existing models for file, database, or processor allocation assume that the physical and logical designs of centralized files or databases are given, and that the primary responsibility of the system designer is to assign those files to computer locations.

The design issues are relatively well understood when the files and databases have been predetermined. Unfortunately, in reality, the situation faced by a distributed system architecture designer is inherently different. At the beginning of the design and analysis process, the files and databases are either not defined or, if they are given, they are expressed in terms that are recognizable and acceptable for the

design of a centralized computing system. If they are left unchanged, however, they could severely restrict the performance of a distributed system. This paper is a first attempt to define, formulate, and develop design tools for cases in which the structure of the databases is not given in advance. The underlying theme of this paper is the view that it is the responsibility of the analyst and system designer to determine the structure of the system databases and to allocate them to computer locations.

The input to the design process consists, in part, of the set of locations (data sources) from which data are recorded, collected, and transferred to the system databases. The data stored in the databases are used in report generation processes to generate periodic and on-demand reports, or provide responses to queries. These reports are used by the companies that are served by the distributed computing system. The final destinations, volume, and generation frequency of those reports constitute part of the input to the system design process.

The data that are stored in the databases arrive there as transactions that have originated at the system data sources. The transactions report on events that have occurred in those locations.

Many reports are generated from data stored in the system databases and are distributed to end users located in different destinations. A report generation process might need as input data that are retrieved from several databases located in distinct locations. The retrieved data have to be sorted, aggregated, and edited in order to generate the requested report; this report will then be distributed to those who use it as part of their regular activities. Different reports might be generated in different locations. The decisions about which location should generate each one of the reports are based on which locations hold the databases that provide input to the report generation process and on the final destinations to which the edited reports have to be transmitted.

The problem that is being addressed in this paper is as follows. The designer is given detailed information on data sources, the volume of update transactions that originate in each one of those locations, the list of reports and queries that have to be generated, their frequency, the volume of output generated, and their distribution to final users. For each report, the designer is also given the set of update transactions needed to generate it, the location (data sources) from which those transactions originate, and their volume per production run. A clear distinction is made between the locations from which update transactions originate and the locations in which those transactions are stored as part of the systems database.

The system designer must decide:

- How many processors to have in the system.

- What constitutes a database, i.e., what aggregations of update transactions form each one of the system databases, and how to partition those databases to realize some of the advantages offered by a distributed system.
- Placement of computers and databases to locations, i.e., which locations will be selected for placement of processing capabilities.
- Assignment of data sources to database locations, i.e., to which database location will the users in each data source send their transactions.
- Report generation locations, the location in which each report will be generated, from which databases the data will be retrieved, and through what distribution channels the edited reports will be sent to the end users.

Many factors have to be taken into consideration when making those decisions. These include operational restrictions, such as response time, reliability, availability, and security constraints; and many cost factors, such as:

- Setup and operational costs for processors, storage devices, and access rights to communication channels.
- The costs involved in transferring transactions from their origination points (data sources) to their databases.
- The cost of updating and maintaining the databases.
- The cost of retrieving data needed for report generation from the databases.
- The cost of transferring the retrieved data from the data retrieval locations to the locations in which the reports are generated.
- The cost of generating a report in a location. Those costs might depend on the location in which the report is being processed.
- The cost of distributing the edited reports to the end users, which could take the form of physical distribution of the printed reports through a carrier or mail service, or through the communication network.

It is highly unrealistic to assume that in a single model we will be able to encompass all of those design issues. Instead we concentrate on the development of a simplified model that addresses the major cost factors influencing the design of a distributed computer system. The results obtained from such a model should be taken with caution and viewed as general guidelines for an initial design that can be used as input to the detailed design of such a system.

An integral part of every systems analysis and design process includes the identification and definition of the sets of inputs (transactions) and outputs (reports) from and to the different entities of the managed system. The major objective of the design process is to decide on a data collection and report generation strategy that will minimize

the system costs. To the best of our knowledge in spite of its practical and fundamental importance, this problem has not been addressed in the open literature, neither for centralized systems nor for distributed systems.

In the next section we present the principal considerations, the setting under which the model has been developed, and the assumptions used for a mathematical formulation of the problem. These are followed by a detailed description of the cost factors that affect the design of a distributed computing system. The problem is formulated in Section IV as a quadratic integer programming problem followed by a linear integer programming formulation of the problem. Several relaxations of the problem are introduced in Section V. The problem addressed here belongs to the class of NP-complete problems. In Section VI, heuristics are suggested to obtain approximate solutions to the problem. The heuristics and the lower bounding procedures were tested on several numerical examples and have generated good initial designs for those cases. Some of the computational results and a numerical example are presented in Section VII. In Section VIII, we present possible extensions of this model. The last section contains conclusions and suggestions for further research.

The following terminology is used throughout the paper: transactions stand for update transactions, while reports stand for read-only transactions and include reports as well as query transactions.

## II. UNDERLYING ASSUMPTIONS

The problem of configuring a general-purpose large-scale distributed computing system is a complex task. Therefore we restrict the formulation and analysis by using simplifying assumptions. The following assumptions are used to formulate the distributed system architecture problem. In subsequent sections we will show how some of those assumptions can be relaxed and incorporated in the model, thus extending the range of cases that can be handled by this and related models.

Assumption 1—(No splitting) All the transactions that originate in a data source are routed to the same set of computer locations. This assumption highly simplifies the directory for routing transactions from their origination location to the database in which they are stored. Under this scheme each data source has a unique set of addresses to which all its transactions are forwarded.

Assumption 2—(No duplication) A single computer location and its associated database handle all the transactions that originate from the same data source. However, one or more data source locations can be assigned to the same computer.

Assumptions 1 and 2 exclude the possibility of having multiple copies

of the same file in the system. The fact that the model is restricted to single-copy databases eliminates the need to consider the nonnegligible overhead required for the synchronization and concurrency control for updating multiple copies of the same database. This highly simplifies the analysis and modeling of the problem. In the last section of the paper we discuss situations in which multiple copies of databases are possible.

Assumption 3—Databases can be assigned only to locations that contain computers with significant processing capacity. A distinction is made between I/O processors and processors whose main activity is arithmetic/logic operations. This implies that assigning a database to a location requires the assignment of a processor to the same location.

Assumption 4—The processing of a report is done in a single location. All the data that are needed for processing a report are sent on demand to that location. This assumption does not exclude the possibility that only a fraction of the transactions will be sent from the database to the report generation location. This fraction depends on the selectivity and compression/aggregation of the data retrieved at the database location. The portion of data that is actually transferred to the report generation location is not a collection of the original transactions. Typically it is only the compressed/aggregated content of the data selected from the databases.

Assumption 5—Reports can be generated only in locations that contain computers.

Assumption 6—Every report is generated in a separate report generation process. The input, intermediate, or final results of one report are not used as input for the generation process of another report (in the same location or some other location). In reality it is possible to save on processing or communication costs by combining a few report generation processes into a sequence of report generation processes. By doing it, it is possible to eliminate duplication in the retrieval, transfer, or editing steps of the report generation process and reduce the systems costs.

This assumption is clearly applicable to cases where reports can be initiated by users at any point in time. This eliminates the possibility of coordinating the report generation processes between the different users. However, it does not prohibit the possibility of combining the generation processes of reports that are always generated in the same run into a single report generation process. The model views the combination of such reports as a single report generation process with outputs being distributed to one or more final destinations. The only restriction that is imposed on such a combination is the requirement to specify beforehand the inputs and outputs from and to the combined process.

Using the above assumptions, four types of locations can be defined.

$D_1$—is the set of data sources. This is the set of locations from which data are collected and recorded to support the operations (decision making and control) of the organization that is being served by the distributed computer system. This is the set of locations in which people or automatic devices observe or detect changes in the state of the system; record them on forms or some other recording devices; and feed them through displays, data collection equipment, communication networks, optical scanners, or magnetic ink readers to the system databases.

Examples of transaction initiators are bank tellers in a bank branch, window clerks in a warehouse, sales personnel in a marketing group, cashiers in a supermarket, or surveying and testing devices located in central offices generating automatic alarms or responses to equipment and facility testing requests in the telephone system. It is unrealistic to assume that the model developed subsequently will be able to capture this level of detail. Therefore a data source location is defined as an aggregation of all the transactions that were initiated from the same geographical area. This might be the set of transactions originating from the same bank branch (which is an aggregation of the tellers in that branch), the set of banking transactions that originate from the same city/region (this is an aggregation of bank branches), the set of transactions that have originated in the same supermarket store, or all the transactions that originate in the same plant/laboratory or warehouse.

$D_2$—is the set of places in which end users are located. Those users request reports that are based on data and transactions that have originated from locations in $D_1$. Those are reports used by the headquarters, regional managers, or clerks. Depending on the organizational structure of the corporation, $D_2$ can include locations that are identical to locations in $D_1$, or locations that are not part of $D_1$.

$D_3$—is the set of locations in which databases are stored. We assume that the existence of a database in a location implies the presence of a processor in the same location.

$D_4$—is the set of locations at which reports are generated. This implies the availability of computing capacity in those locations. The intersection of $D_4$ and $D_3$ typically is not empty. A location that belongs to $D_3$ and $D_4$ is a location with a computer that maintains a database and is also used to generate reports for some users.

## III. COST STRUCTURE OF THE PROBLEM

The selection of a final configuration of a distributed computing system depends to a large extent on many cost factors. Some of those cost components are dependent on the amount of activity in the

system and could be nonlinear functions of the distance, number of transactions, or storage requirements. In this section we outline the major cost components that influence such a design. Distributed computer systems are expected to be operational over a long lifetime, with different expenses incurred at varying stages of the system life cycle. In order to bring those expenses into a common denominator, they are discounted to their present value, using appropriate discounting factors.

### *3.1 Setup and operational costs*

These are the setup and operational costs of placing a computer and appropriate storage devices in a location. These costs are composed of purchasing/rental, installation, and maintenance of hardware and software; the physical facilities in which they are installed; and the staffing requirements for production runs, maintenance, and development activities. They include only cost factors that are independent of the amount of activity from and to that site, in terms of update transactions or retrieval requests.

### *3.2 Data collection transfer and update costs*

Data collection transfer and update costs include:

- The costs for recording as transactions events that occur at data source locations
- Routing all the transactions that originate in a data source to their database location
- Editing, verifying, validating, and updating the database with correct transactions
- Sending a transaction update confirmation response to the transaction originator and handling the errors discovered during that process.

Those costs are composed of processing, communications, and storage costs, and depend on

- The number of transactions processed over the planning period
- The length (in bytes) of an average transaction
- The duration in time that individual transactions are kept as active detailed on line data in the database
- The amount of storage required to store the stable part of the database that contains data on the entities that belong to its data sources.

The cost expressions make a clear distinction between the traffic intensity (measured by the number of transactions) from a given source to its database, and the amount of storage needed to store the relevant data pertaining to that source. The on-line storage requirements for a single data source are composed of two components. They

are tightly coupled to the amount of data held on the different entities in the data source, and are also a function of the number of transactions held as active transactions in the database.

Using a banking example, we can have bank branches that have relatively few accounts, but the average account holders in those branches are very active and therefore many transactions are generated from and to those accounts. On the other hand, we can have branches that have many accounts, with very little activity going on in those accounts. In this specific example, there is very little relationship between the number of accounts (which are the dominant factor in determining the on-line storage requirements) and the average activity in those accounts (which determines the communication, processing, and update costs).

The data collection, transfer, and update costs between points $i$ and $j$ are given by the following cost expressions:

$$C_{ij}^{(1)}(A_i,E_i) = f_i^{(1)}(A_i,E_i) + f_{ij}^{(1)}(A_i,E_i) + g_j^{(1)}(A_i,E_i),$$

where:

$A_i$—is the number of transactions per time period that originate from data source $i$, $i \in I$.

$E_i$—is the expected number of entities in data source $i$, $j \in I$ for which data are kept in the system databases.

$f_i^{(1)}(A_i,E_i)$—captures the costs for data recording, collection, and handling of correct and rejected transactions, in data source $i$. These costs are primarily dependent and proportional to $A_i$, in some cases it is also dependent on $E_i$.

$f_{ij}^{(1)}(A_i,E_i)$—captures the cost of transferring transactions from data source $i$ to a database in location $j$; this also includes the cost of sending responses to the transaction originators on updated transactions, and the resubmission of transactions that have been rejected by the system. The expression $f_{ij}^{(1)}(\cdot)$ consists mainly of communication costs between points $i$ and $j$, and is proportional to the number of messages transmitted and to the average message length.

$g_j^{(1)}(A_i,E_i)$—captures the processing costs for editing, verifying, validating, and updating in location $j$ transactions that have originated from source $i$. The expression $g_j^{(1)}(\cdot)$ also includes the storage costs for storing in location $j$ data that have originated or are relevant for processing transactions that have originated from data source $i$. The storage costs depend mainly on the amount of storage needed to store information on entities in $E_i$

and to keep readily accessible in location $j$ a subset of the transactions that are submitted from $i$ to $j$.

### 3.3 Data retrieval and transfer costs

These include the costs for retrieving from a database in location $j$ data that are needed for generating report $r$, $r \in R$, and transferring the retrieved data to the location in which report $r$ is generated. The data retrieval costs over the planning horizon depend on the frequency in which report $r$ is generated and consist of:

- Data processing costs for retrieving data from the databases, and sorting and aggregating the retrieved data at the data retrieval location (in order to reduce the amount of data that has to be transferred to the report generation location)
- Data communication costs for transferring the retrieved data from the data retrieval location to the report generation location. The amount of data that has to be transferred depends on the selectivity of the retrieved data (the ratio between the amount of data collected in the database and the portion that is actually retrieved for a specific report). In addition, the retrieved data are also aggregated, which further reduces the volume of data that has to be transferred from the system databases to the report generation locations.

Using the assumption that those costs are separable over data sources, the following expressions give the cost of retrieving from a database in location $j$ data that have originated in data source $i$, $j \in S_r$, are stored in location $j$, and are used as input to the generation process of report $r$, $r \in R$. Given that report $r$ is generated in location $k$,

$$C^{(2)}_{ijkr}(A_i, E_i, S_{ir}) = f^{(2)}_{ijr}(A_i, E_i, S_{ir}) + f^{(2)}_{ijkr}(A_i, E_i, S_{ir})$$

$S_{ir}$ = the selectivity factor between the amount of data entered into the database from data source $i$, and the amount of data that is retrieved from the same database for report $r$.

$f^{(2)}_{ijr}(A_i, E_i, S_{ir})$ = the data retrieval and processing costs (in location $j$) for retrieving from a database in location $j$, data that have originated from source $i$, $i \in S_r$, and are needed as input to the report generation process of report $r$.

$f^{(2)}_{ijkr}(A_i, E_i, S_{ir})$ = the communication costs for transferring the retrieved data from the data retrieval location $j$ to the report generation location $k$. These costs are proportional to the amount of data that has to be transferred between the two locations ($j$ and $k$),

and the characteristics of the data source $i$ and the requested report $r$.

### *3.4 Report processing editing and distribution costs*

These are costs for generating report $r$ in location $k$ and for distributing the edited report to the end users of that report. They are incurred from the moment that all the data that are needed to generate report $r$ are available at the report generation location ($k$). They are composed of processing costs in location $k$ for combining the different data sets that were transferred to location $k$ from database locations that contain data needed for report $r$. These data are sorted, aggregated, and edited to form report $r$. Communication costs are incurred when the edited reports are transferred to the end users. Since those cost components are not dependent on the locations from which the input data have been retrieved they are expressed as:

$$C_{kr}^{(3)}(\bar{A}_r, \bar{E}_r, N_r) = f_{kr}^{(3)}(\bar{A}_r, \bar{E}_r) + g_{kr}^{(3)}(N_r),$$

where:

$\bar{A}_r$ = the volume of data that is used as input to report $r$. It is based on transactions collected from all the data sources that provide input for report $r$.

$\bar{E}_r$ = the amount of data that is related to all the entities whose data are being used as input to report $r$.

$N_r$ = the set of locations that contain end users of report $r$. This is the set of locations to which the report will be distributed.

$f_{kr}^{(3)}(\bar{A}_r, \bar{E}_r)$ = the generation costs (I/O, processing, and memory) of report $r$ in location $k$. They are a function of $\bar{A}_r, \bar{E}_r$ and the frequency with which report $r$ is being generated over the planning horizon.

$g_{kr}^{(3)}(N_r)$ = the distribution costs of report $r$, from location $k$ in which it is generated, to the set of end users of this report.

Large amounts of data have to be collected and analyzed in order to provide input to a system configuration model. This is a time-consuming and costly effort that might preclude the use of such a design tool owing to excessive data processing and preparation costs. Fortunately, under appropriate assumptions, some of those cost components are identical for different system configurations and are irrelevant for cost comparisons, and therefore need not be collected. For example, if the cost of generating report $r$ does not depend on the location in which the report is being generated, then there is no need to specify $f_{kr}^{(3)}(\bar{A}_r, \bar{E}_r)$ in the design model. Similarly, if the costs for recording, collection, and handling of data in data source $i$ are independent of

the location in which those data are finally stored, then there is no need to specify the values of $f_i^{(1)}(A_i, E_i)$ in $C_{ij}^{(1)}(A_i, E_i)$. By paying careful attention to which cost components are included in the model, the amount of effort needed for data collection and analysis can be reduced significantly.

## IV. MATHEMATICAL FORMULATION OF THE PROBLEM

In this section we present two mathematical programming formulations of the distributed system architecture design problem. The first formulation is an integer programming problem with a linear constraint set and a quadratic objective function. This class of problems is extremely difficult to solve. Therefore, in the second formulation, the objective function is linearized by defining additional variables and constraints.

To formulate the problem, we define the following decision variables:

$Z_j$ = a binary variable equal to one if a computer and database are placed in location $j$, and equal to zero otherwise.

$X_{ij}$ = a binary variable equal to one if all the transactions that originate from a data source $i$ are routed to a database in location $j$, $X_{ij}$, and equal to zero otherwise.

$Y_{kr}$ = a binary variable equal to one if report $r$ is assigned to be generated in location $k$, and equal to zero otherwise.

The distributed system architecture problem is formulated as:

*Problem QIP:*

Find binary variables $Z_j$, $X_{ij}$, $Y_{kr}$ that satisfy:

$$Z_{\text{QIP}} = \text{Min} \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} X_{ij} V_{ij} + \sum_{r \in R} \sum_{k \in J} Y_{kr} U_{kr} + \sum_{r \in R} \sum_{i \in S_r} \sum_{j \in J} \sum_{k \in J} X_{ij} Y_{kr} W_{ijkr} \right\}$$

subject to:

$$\sum_{j \in J} X_{ij} = 1 \qquad i \in I \tag{1}$$

$$X_{ij} \leq Z_j \qquad i \in I, j \in J \tag{2}$$

$$\sum_{k \in J} Y_{kr} = 1 \qquad r \in R \tag{3}$$

$$Y_{kr} \leq Z_k \qquad r \in R, k \in J \tag{4}$$

$$X_{ij}, Y_{jr}, Z_j = 0 \text{ or } 1 \qquad i \in I, j \in J, r \in R. \tag{5}$$

The constraints in (1) ensure that all the transactions that originate

in data source $i$ will be routed to exactly one of the locations that are candidates for database (and processor) placement; the constraints in (2) ensure that transactions will be sent from source $i$ to location $j$ only if a database has been placed in location $j$. Similarly, the constraints in (3) guarantee that each report will be assigned to some report generation location, while the constraints in (4) prevent the assignment of a report generation process to a location that does not have processing capabilities.

The objective function consists of the following cost components:

$P_j$ = the cost of placing a computer and a database in location $j$. We assume here that placing a processor (or a database) in location $j$ implies that the capabilities for handling a database (or report generation) have also been acquired. Those setup costs are incurred only when $Z_j = 1$.

$V_{ij}$ = the marginal cost over the planning horizon for sending all the transactions that originate from data source $i$ to update a database that is placed in location $j$. This includes the communication costs for transferring the transactions from location $i$ to location $j$; the transaction processing costs in location $j$ for verifying, validating those transactions, and updating them in the database; storage costs for storing those transactions in the database; and the costs of sending rejection or acceptance responses from location $j$ to data source $i$ on the status of those transactions. Those costs are incurred only if $X_{ij} = 1$.

$U_{kr}$ = the marginal cost over the planning horizon for processing report $r$ in location $k$. This includes the costs for editing the report in location $k$ and for distributing its outputs to end users. $U_{kr}$ is incurred only when $Y_{kr} = 1$.

$W_{ijkr}$ = the marginal cost over the planning horizon of retrieving for report $r$, which is generated in location $k$, data that originated in data source $i$, $i \in S_r$, which is currently stored in a database in location $j$. This includes the cost of retrieving data from the database, selecting and aggregating it in location $j$ and transferring the aggregated data from location $j$ to location $k$, and preparing it as input to the process that generates report $r$ (in location $k$). This cost is incurred only if the multiplication of $X_{ij}$ by $Y_{kr} = 1$, i.e., the transactions collected from data source $i$ have been assigned to a database in location $j$, and report $r$ has been assigned to a processor in location $k$.

The mathematical programming formulation of *Problem QIP* is a quadratic integer programming problem. It has a linear constraint set and a quadratic objective function. The present state of the art in

solving nonlinear integer programming problems does not hold much promise of solving those types of problems in reasonable amounts of computing times. The objective function can be linearized by defining a new set of binary variables, $\Psi_{ijkr}$, $r \in R$, $i \in S_r$, $j \in J$, $k \in J$. $\Psi_{ijkr}$ is equal to one when report $r$ is assigned to be generated at location $k$ and it uses transactions that have originated from data source $i, i \in S_r$, and were stored in a database in location $j$. $\Psi_{ijkr}$ is zero otherwise.

The distributed system architecture problem is reformulated as:

*Problem ILP*:

Find binary variables $Z_j$, $X_{ij}$, $Y_{kr}$, $\Psi_{ijkr}$ that satisfy:

$$Z_{\text{ILP}} = \text{Min} \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} X_{ij} V_{ij} + \sum_{r \in R} \sum_{k \in J} Y_{kr} U_{kr} + \sum_{r \in R} \sum_{j \in J} \sum_{k \in J} \sum_{i \in S_r} \Psi_{ijkr} W_{ijkr} \right\}$$

subject to:

$$\sum_{j \in J} X_{ij} = 1 \qquad i \in I \tag{6}$$

$$X_{ij} \leq Z_j \qquad i \in I, j \in J \tag{7}$$

$$\sum_{k \in J} Y_{kr} = 1 \qquad r \in R \tag{8}$$

$$Y_{kr} \leq Z_k \qquad r \in R, k \in J \tag{9}$$

$$\Psi_{ijkr} \leq X_{ij} \qquad j \in J, k \in J, r \in R, i \in S_r \tag{10}$$

$$\Psi_{ijkr} \leq Y_{kr} \qquad j \in J, k \in J, r \in R, i \in S_r \tag{11}$$

$$\sum_{j \in J} \sum_{k \in J} \Psi_{ijkr} = 1 \qquad r \in R, i \in S_r \tag{12}$$

$$Z_j, X_{ij}, Y_{kr}, \Psi_{ijkr} = 0 \text{ or } 1 \qquad i \in I, j \in J, k \in J, r \in R \tag{13}$$

The constraints in eqs. (6) through (9) have the same interpretation as constraints (1) through (4) in the quadratic programming formulation of the problem. The constraints in (10) ensure that reports that use data originating from data source $i$ will be able to retrieve it from a database in location $j$, only if the transactions originating from location $i$ have been assigned to update a database placed in location $j$. The constraints in (11) ensure that the data needed for report $r$ are sent to location $k$ only if report $r$ is actually generated in location $k$. By definition, report $r$ uses transactions that originate in location $i, i \in S_r$. These transactions must first be routed and update a database. When report $r$ is requested, the data are retrieved from the database

and sent to the location in which report $r$ is generated. The constraints in (12) ensure that such a routing to and from a database actually takes place for each combination of $r \in R$ and $i \in S_r$.

## V. GENERATING LOWER BOUNDS ON THE OPTIMAL SOLUTION

The problem of optimally configuring a distributed computing system is a combinatorial optimization problem. It belongs to the class of NP complete problems. This statement can be proved by setting $W_{ijkr} = 0 \forall i, j, k, r$. The quadratic terms in *Problem QIP* are eliminated and the problem is reduced to an uncapacitated plant location problem, which has been proven to be in the NP complete class.[20] This observation implies that it is unlikely that an algorithm will be developed that is capable of solving every instance of the problem to optimality. Moreover, using similar arguments it can be shown that unless $P = NP$, it is not possible to develop a polynomial time algorithm which produces approximate solutions that have a guaranteed absolute error bound. The existence of such an approximation scheme implies the ability to generate feasible solutions that have an objective function value within a user-predefined error tolerance from the optimal solution. Given this situation, the only alternative is for a system designer to resort to heuristics. Those heuristics can generate feasible solutions for the problem. However, they are not guaranteed to generate an optimal solution. Many heuristics have been developed for a variety of combinatorial optimization problems. The practical experience gained in using those heuristics is quite encouraging. Extensive computational experimentation with many of those heuristics shows that in many cases they generate solutions that are very close to the optimal solution.

The solutions generated by the heuristics are feasible to the problem of configuring distributed systems, and as such provide an upper bound on the value of the (unknown) optimal solution. Since the value of the optimal solution to the system configuration problem is unknown, to evaluate the quality of the solutions generated by the heuristics, we develop in this section methods for computing lower bounds on the value of the optimal solution. The difference between the value of the best upper bound generated by the heuristic and the tightest lower bound generated by the developed method is clearly an upper bound on the gap between the value of the heuristic solution and the true optimal solution. Keeping in mind that the data supplied to those models contain errors that in many cases exceed the errors introduced by the heuristic, these heuristics seem to provide a plausible option for a system designer.

We present two methods for computing lower bounds. The first method is based on a Lagrangian relaxation of *Problem ILP*, which,

together with a multiplier updating method, can produce tight lower bounds on the optimal solution value. This procedure requires extensive computational resources, which limits its applicability to cases in which the system designer is willing to incur those expenses. As an alternative to this time-consuming procedure, a second method is developed for computing lower bounds. This procedure generates lower bounds that are not as tight as the Lagrangian-based procedure, but are easier to implement and require only a modest amount of computing resources. It is a preferable alternative for design problems in which a rough estimate on the quality of the solution suffices.

### *5.1 The Lagrangian relaxation procedure*

The Lagrangian relaxation of *Problem ILP* is formed by multiplying the constraints in (12) by a vector $\{\lambda_{ir}\}$ of Lagrange multipliers, and the constraints in (11) with the vector $\{\beta_{ijkr}\}$, and adding them to the objective function, forming the following Lagrangian problem:

$$L(\lambda, \beta) = \min \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} X_{ij} V_{ij} + \sum_{r \in R} \sum_{k \in J} Y_{kr} U_{kr} \right.$$

$$+ \sum_{r \in R} \sum_{j \in J} \sum_{k \in J} \sum_{i \in S_r} \Psi_{ijkr} W_{ijkr} + \sum_{r \in R} \sum_{i \in S_r} \lambda_{ir} \left( 1 - \sum_{j \in J} \sum_{k \in J} \Psi_{ijkr} \right)$$

$$\left. + \sum_{r \in R} \sum_{j \in J} \sum_{k \in J} \sum_{i \in S_r} \beta_{ijkr} (Y_{kr} - \Psi_{ijkr}) \right\}$$

subject to eqs. (6) through (10) and (13).

After rearrangement and collection of terms that correspond to identical variable types, the objective function is reduced to:

$$L(\lambda, \beta) = \min \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} X_{ij} V_{ij} \right.$$

$$+ \sum_{r \in R} \sum_{k \in J} Y_{kr} \left( U_{kr} + \sum_{i \in S_r} \sum_{j \in J} \beta_{ijkr} \right) + \sum_{r \in R} \sum_{i \in S_r} \lambda_{ir}$$

$$\left. + \sum_{r \in R} \sum_{j \in J} \sum_{k \in J} \sum_{i \in S_r} \Psi_{ijkr} (W_{ijkr} - \beta_{ijkr} - \lambda_{ir}) \right\},$$

leading to the following optimization problem.

*Problem LR*

Find binary variables $X_{ij}$, $Y_{ij}$, $Z_j$, $\Psi_{ijkr}$ that satisfy:

$$L(\lambda, \beta) = \sum_{r \in R} \sum_{i \in S_r} \lambda_{ir} + \min \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} X_{ij} V_{ij} \right.$$

$$\left. + \sum_{r \in R} \sum_{k \in J} Y_{kr} \hat{U}_{kr} + \sum_{r \in R} \sum_{j \in J} \sum_{k \in J} \sum_{i \in S_r} \Psi_{ijkr} \hat{W}_{ijkr} \right\}$$

subject to the constraints in eqs. (6) through (10) and (13), where

$$\hat{U}_{kr} = U_{kr} + \sum_{j \in J} \sum_{i \in S_r} \beta_{ijkr},$$

and

$$\hat{W}_{ijkr} = W_{ijkr} - \beta_{ijkr} - \lambda_{ir}.$$

*Problem LR* is relatively easy to solve, if we note that for a fixed vector $\hat{X}_{ij}$ of $X_{ij}$ variables, the $\Psi_{ijkr}$ variables in the Lagrangian problem must satisfy:

$$\hat{\Psi}_{ijkr} = \begin{cases} 0 \text{ if } \hat{X}_{ij} = 0, \\ 0 \text{ if } \hat{X}_{ij} = 1 \quad \text{and} \quad \hat{W}_{ijkr} \geq 0, \\ 1 \text{ if } \hat{X}_{ij} = 1 \quad \text{and} \quad \hat{W}_{ijkr} < 0, \end{cases}$$

where $\hat{\Psi}_{ijkr}$ is the optimal solution to *Problem LR* for a fixed set of Lagrange multipliers. From the above relationship it is easy to verify that:

$$\hat{\Psi}_{ijkr} \hat{W}_{ijkr} = \hat{X}_{ij} \min\{0;\ \hat{W}_{ijkr}\}.$$

Using this relation, *Problem LR* can be rewritten as:

$$L(\lambda, \beta) = \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} X_{ij} \hat{V}_{ij} + \sum_{r \in R} \sum_{j \in J} Y_{jr} \hat{U}_{jr} \right\} + \min \sum_{r \in R} \sum_{i \in S_r} \lambda_{ir} \tag{14}$$

subject to:

$$\sum_{j \in J} X_{ij} = 1 \qquad i \in I, \tag{15}$$

$$\sum_{j \in J} Y_{jr} = 1 \qquad r \in R, \tag{16}$$

$$X_{ij} \leq Z_j \qquad i \in I, j \in J, \tag{17}$$

$$Y_{jr} \leq Z_j \qquad j \in J, r \in R, \tag{18}$$

$$Z_j,\ X_{ij},\ Y_{jr} = 0 \text{ or } 1 \qquad i \in I, j \in J, r \in R, \tag{19}$$

where

$$\hat{V}_{ij} = V_{ij} + \sum_{k \in J} \sum_{r \in R} \min\{0;\ \hat{W}_{ijkr}\}.$$

The problem given by eqs. (14) through (19) is convertible to a simple uncapacitated plant location problem, in which the set of candidate plant locations is equal to $J$, and the set of customer locations is given by the union of the sets $I$ and $R$. Uncapacitated plant location problems are optimization problems that belong to the class of NP complete problems. As such, it is unlikely that an algorithm that is capable of solving every instance of the problem to optimality in polynomial time will be developed. However, ample empirical evidence exists showing that several algorithms that have been developed by several researchers for solving uncapacitated plant location problems can solve without too much difficulty many occurrences of the problem. Erlenkotter has developed such a code.[21] It is a branch and bound dual-based procedure for solving plant location problems. The procedure was implemented and tested on a large set of cases and was able to solve without much difficulty many plant location problems to optimality.

Given a fixed vector of Lagrange multipliers, it can be shown that $L(\lambda, \beta)$ is a lower bound on the value of the optimal solution to the distributed system configuration problem, i.e., $L(\lambda, \beta) \leq Z_{ILP}$. Different multiplier values will generate different values for $L(\lambda, \beta)$. We are interested in obtaining a set of multipliers $(\hat{\lambda}, \hat{\beta})$ that generate the tightest possible bounds on $Z_{ILP}$, i.e., they satisfy:

$$L(\hat{\lambda}, \hat{\beta}) = \max_{(\lambda,\beta)}\{L(\lambda, \beta)\}.$$

Obtaining the optimal multiplier values is not a simple task. $L(\lambda, \beta)$ is a nondifferentiable function, making a straightforward optimization of the multiplier values a difficult task. Several methods have been suggested in the literature for computing good approximations to $(\hat{\lambda}, \hat{\beta})$. Those procedures compute multiplier values that generate solutions that are very close to $L(\hat{\lambda}, \hat{\beta})$. Those procedures include methods such as column generation, dual ascent, multiplier adjustment, or subgradient optimization procedures.[22,23]

The subgradient optimization procedure is an iterative method that starts with an initial set of multiplier values and uses a multiplier updating scheme to improve the lower bound value. The procedure is initiated with an initial set of multiplier values and uses the following multiplier updating scheme when moving from one iteration to the next:

$$\lambda_{ir}^{p+1} = \lambda_{ir}^{p} - t_p \gamma_{ir}^{p},$$

$$\beta_{ijkr}^{p+1} = \beta_{ijkr}^{p} - t_p \gamma_{ijkr}^{p},$$

where $\lambda_{ir}^p$ and $\beta_{ijkr}^p$ are the multiplier values used in iteration $p$. The subgradient directions are given by:

$$\gamma_{ir}^p = 1 - \sum_{j \in J} \sum_{k \in J} \Psi_{ijkr}^p,$$

$$\gamma_{ijkr}^p = Y_{kr}^p - \Psi_{ijkr}^p.$$

$Y_{kr}^p$ and $\Psi_{ijkr}^p$ are the optimal variable values for the Lagrangian problem at iteration $p$, and $t_p$ is a step size. Poljack[24] proved that the sequence $(\lambda^p, \beta^p)$ converges to $(\hat{\lambda}, \hat{\beta})$ if the step size $t_p$ satisfies the conditions:

$$\lim_{p \to \infty} t_p = 0 \quad \text{and} \quad \sum_{p=1}^{\infty} t_p = \infty.$$

Unfortunately, such a sequence is not practical for a computer implementation. Therefore, a heuristic is used for computing the step size $t_p$. It is given by:

$$t_p = \delta_p \frac{\bar{Z} - L(\lambda^p, \beta^p)}{|\gamma^p|^2},$$

where $\bar{Z}$ is an overestimate on $L(\hat{\lambda}, \hat{\beta})$ and $\delta_p$ is a scalar in the range of zero to two. $\delta_p$ is initially set to two and is divided by two if, in a sequence of $n$ iterations, there was no improvement in the Lagrangian value. The procedure is terminated when one of the following termination conditions is satisfied:

1. The number of iterations has exceeded an upper limit.
2. The difference between the upper and lower bound limits is below a given tolerance $\epsilon$, where

$$\left| \frac{\bar{Z} - L(\lambda, \beta)}{L(\lambda, \beta)} \right| \leq \epsilon.$$

3. The step size $t_p$ is small, $t_p \leq \epsilon_1$.
4. The scalar $\delta_p$ is small, $\delta_p \leq \epsilon_2$.

Extensive computational experience with the subgradient optimization procedure reveals that it is not sensitive to the overestimate $\bar{Z}$ value, or to the initial multiplier values. The procedure is sensitive to the value of the parameter $n$, which determines the rate of change in the $\delta_p$ value. Setting $n$ to a high value implies that a large number of iterations will be needed before the procedure converges to a good solution. Setting $n$ to a low value may lead to termination before the procedure had the time to converge. Appropriate $n$ values can be determined only through computational experimentation.

The combination of Lagrangian relaxation and subgradient optimization procedures has been successfully applied to a variety of

combinatorial optimization problems. Those include problems such as the traveling salesman problem,[23] multiple traveling salesman problems,[25] topological design of computer communication networks,[19,26] or routing in computer networks.[27] The application of the same procedure to the distributed system configuration problem is practical only for problems with a small number of reports and database locations. When applied to *Problem ILP*, the procedure requires the storage and update of $(|J|^2 \sum_{r \in R} |S_r|)$ multiplier values, leading to large storage requirements and to a relatively slow convergence.

### *5.2 A simple lower bounding method*

The Lagrangian relaxation-based procedure for computing lower bounds to *Problem ILP* is a computationally expensive procedure, which might preclude its use for many system design problems. An alternative method for computing lower bounds that requires significantly less computing resources, but is expected to produce inferior bounds, is based on the quadratic programming formulation of the problem. The method is based on the following argument. If the quadratic terms in the objective function of *Problem QIP* are dropped, the optimization problem is reduced to a simple uncapacitated plant location problem. The cost of this plant location problem is clearly a lower bound on the value of the optimal solution to *Problem ILP*. This bound can be further strengthened by adding to the $U_{kr}$ terms in the solution of the plant location problem an underestimate of the costs that have been neglected by eliminating the quadratic terms from the objective function. Those terms capture the costs for transferring from some unknown database locations to location $k$ data that originated in source $i$ that are needed for the generation of report $r$.

Letting $W_{ikr}$ be the minimal cost of sending data that originate in data source $i$, $i \in S_r$ to some database location, retrieving it from that location, and transferring it to location $k$ where report $r$ is generated; that cost is given by $\min_{j \in J}\{W_{ijkr}\}$. By summing up those individual costs over all data sources that originate data for report $r$, a lower bound $\tilde{U}_{kr}$ on the data retrieval and transfer costs for generating report $r$ in location $k$ is obtained:

$$\tilde{U}_{kr} = U_{kr} + \sum_{i \in S_r} \min_{j \in J}\{W_{ijkr}\} = U_{kr} + \sum_{i \in S_r} W_{ikr}.$$

Substituting $\tilde{U}_{kr}$ for $U_{kr}$ in the relaxed problem leads to the following optimization problem.

*Problem YLP*
Find binary variables $Z_j$, $X_{ij}$, $Y_{kr}$ that satisfy:

$$Z_{LB}^{Y} = \min \left\{ \sum_{j \in J} Z_j P_j + \sum_{i \in I} \sum_{j \in J} Z_{ij} V_{ij} + \sum_{r \in R} \sum_{j \in J} Y_{jr} \tilde{U}_{jr} \right\}$$

subject to eqs. (15) through (19).

This optimization problem is a simple, uncapacitated plant location problem. Its solution provides a lower bound on the value of the optimal solution.

*Theorem 1:* $Z_{LB}^{Y} \leq Z_{\text{QIP}}$.

*Proof*: Let $(\bar{Z}^*, \bar{X}^*, \bar{Y}^*)$ be the optimal solution to *Problem QIP*, and $(\bar{Z}^Y, \bar{X}^Y, \bar{Y}^Y)$ the optimal solution to *Problem YLB*. □

Letting $j_i$ be the index of the database location to which data are sent and stored from data source $i$, in the optimal solution to *Problem QIP*. From the definition of $W_{ikr}$, it is clear that:

$$W_{ij_ikr} \geq W_{ikr} = \min_{j \in J} \{W_{ijkr}\}.$$

The quadratic terms in the objective function of *Problem QIP* satisfy the following relation:

$$\sum_{r \in R} \sum_{j \in J} \sum_{k \in J} \sum_{r \in S_r} X_{ij}^* Y_{kr}^* W_{ijkr}$$

$$= \sum_{r \in R} \sum_{k \in J} Y_{kr}^* \sum_{i \in S_r} W_{ij_ikr} \geq \sum_{r \in R} \sum_{k \in J} Y_{kr}^* \sum_{i \in S_r} W_{ikr}.$$

From the above relations it follows that:

$$Z_{\text{QIP}} = \bar{Z}^* \bar{P} + \bar{X}^* \bar{V} + \bar{Y}^* \bar{U} + \bar{X}^* \bar{Y}^* \bar{W}$$

$$\geq \bar{Z}^* \bar{P} + \bar{X}^* \bar{V} + \bar{Y}^* \bar{U} + \bar{Y}^* \left\{ \overline{\sum_{i \in S_r} W_{ikr}} \right\}.$$

Since $(\bar{Z}^*, \bar{X}^*, \bar{Y}^*)$ is a feasible solution to *Problem YLB*, the following relation follows:

$$\bar{Z}^* \bar{P} + \bar{X}^* \bar{V} + \bar{Y}^* \bar{U} + \bar{Y}^* \left\{ \overline{\sum_{i \in S_r} W_{ikr}} \right\} \geq \bar{Z}^Y \bar{P}$$

$$+ \bar{X}^Y \bar{V} + \bar{Y}^Y \bar{U} + \bar{Y}^Y \left\{ \overline{\sum_{i \in S_r} W_{ikr}} \right\} = Z_{LB}^{Y}.$$

Thus $Z_{LB}^{Y} \leq Z_{\text{QIP}}$.

It is easy to see that this bounding method will provide a tight lower bound whenever

$$\sum_{i \in S_r} \max_{j \in J} \{W_{ijkr}\} \ll U_{kr} \qquad r \in R,\ k \in J,$$

i.e., the marginal cost contribution of $\sum X_{ij}Y_{kr}W_{ijkr}$ is small when compared to the other cost factors of the problem.

By applying similar arguments to the $X_{ij}$ variables in *Problem QIP*, a second bound is generated by the solution to the following problem.

*Problem XLB*

Find binary variables $Z_j$, $X_{ij}$, $Y_{kr}$ that satisfy:

$$Z^X_{\text{LB}} = \min \left\{ \sum_{j\in J} Z_j P_j + \sum_{i\in I}\sum_{j\in J} X_{ij}\tilde{\tilde{V}}_{ij} + \sum_{r\in R}\sum_{k\in J} Y_{kr}U_{kr} \right\},$$

where

$$\tilde{\tilde{V}}_{ij} = V_{ij} + \sum_{r\in T_i} \min_{k\in J}\{W_{ijkr}\},$$

and $T_i$ is the index set of the reports that use data from data source $i$.

The lower bound value is given by the maximum of those two bounds:

$$Z_{\text{LB}} = \max\{Z^X_{\text{LB}};\ Z^Y_{\text{LB}}\}.$$

The relationship between $Z_{\text{LB}}$ and $L(\hat{\lambda}, \hat{\beta})$ is established in the following theorem.

*Theorem 2*: $Z_{\text{LB}} \le L(\hat{\lambda}, \hat{\beta})$.

*Proof*: First we prove that $Z^Y_{\text{LB}} \le L(\hat{\lambda}, \hat{\beta})$. The feasible sets for *Problem LR* and *Problem YLB* are identical. Therefore, it suffices to show that the objective function of *Problem YLB* is a special case for *Problem LR*. The objective function for *Problem LR* can be written as:

$$\left[ \sum_{j\in J} Z_j P_j + \sum_{i\in I}\sum_{j\in J} X_{ij}V_{ij} + \sum_{r\in R}\sum_{k\in J} Y_{hr}\Big(U_{hr} + \sum_{i\in S_r}\sum_{j\in J}\beta_{ijhr}\Big) \right.$$

$$\left. + \sum_{r\in R}\sum_{i\in S_r}\lambda_{ir} + \sum_{i\in I}\sum_{j\in J} X_{ij}\left(\sum_{k\in J}\sum_{r\in R}\text{Min}\{0;\ W_{ijkr} - \beta_{ijkr} - \lambda_{ir}\}\right)\right].$$

When the multipliers are set to:

$$\lambda_{ir} = 0 \quad \forall r \in R,\ i \in S_r$$

and

$$\beta_{ijkr} = \begin{cases} 0 & \forall j \ne j_i,\ i \in S_R,\ r \in R,\ k \in J \\ W_{ikr} & \forall j = j_i,\ i \in S_r,\ r \in R,\ k \in J, \end{cases}$$

$j_i$ is an index of $j$ that satisfies $W_{ij_ikr} = W_{ikr}$. Under those conditions $\text{Min}\{0;\ W_{ijkr} - \beta_{ijkr} - \lambda_{ir}\} = 0$ and the objective function of the Lagrangian is reduced to

$$\left[\sum_{j\in J} Z_j P_j + \sum_{i\in I}\sum_{i\in J} X_{ij} V_{ij} + \sum_{r\in R}\sum_{k\in J} Y_{kr}\,\tilde{\tilde{U}}_{kr}\right],$$

which is the objective function to *Problem YLB*. Thus

$$Z^{Y}_{\mathrm{LB}} = L(0,\ \bar{W}_{ikr}) \leq L(\hat{\lambda},\ \hat{\beta}).$$

Using similar arguments it can be shown that $Z^{X}_{\mathrm{LB}} \leq L(\hat{\lambda},\ \hat{\beta})$, leading to $Z_{\mathrm{LB}} \leq L(\hat{\lambda},\ \hat{\beta})$ □

## VI. HEURISTICS FOR CONFIGURING DISTRIBUTED COMPUTER SYSTEMS

The problem of designing an optimal configuration of a distributed computing system belongs to the class of NP complete problems. As such it is unlikely that an algorithm will be developed that is capable of solving every instance of the problem to optimality. In lieu of this evidence, the only avenue that is open to a systems designer is to rely on heuristic procedures. Those are expected to generate good but not necessarily optimal solutions for the problem. In this section we develop such heuristics. Providing a theoretical bound on the expected quality of the solutions generated by those heuristics is itself a hard problem. Therefore, the performance of those heuristics is investigated in a set of computational experiments in which the heuristics are applied to a large number of problems and their performance is compared to the lower bound values that were generated by using the procedures developed in Section V. The sensitivity of those procedures to various design parameters is evaluated and can help in identifying the conditions under which those procedures can be expected to generate good solutions.

The heuristics developed in this section are of the greedy type. They start from an initial solution and attempt to improve it by reassigning data sources to database locations or reports to report generation locations. The solution improvement steps are based on the following observations.

Assuming that the decisions regarding the assignment of reports to report generation locations have been made, i.e., a vector $\{Y_{kr}\}$ of zero-one variables has been selected, the optimization problem, *Problem QIP*, is reduced to:

*Problem YIP*

$$Z_{YIP} = \min\left\{\sum_{j\in J_y} Z_j P_j + \sum_{i\in I}\sum_{j\in J} X_{ij}\,\ddot{V}_{ij}\right\}$$

subject to:

$$\sum_{j \in J} X_{ij} = 1 \qquad i \in I,$$

$$X_{ij} \leq Z_j \qquad i \in I, j \in \ddot{J}_y,$$

$$X_{ij}, Z_j = 0 \text{ or } 1 \qquad i \in I, j \in J,$$

where

$$J_y = \{j \mid Y_{jr} = 1 \text{ and } r \in R, j \in J\}$$

$$\ddot{J}_y = J - J_y$$

$$T_i = \{r \mid i \in S_r \text{ and } r \in R\}$$

$$\ddot{V}_{ij} = V_{ij} + \sum_{r \in T_i} W_{ijk_r r},$$

and where $k_r$ is the index of the location in which report $r$ is generated (i.e., $Y_{k_r r} = 1$).

Thus, when the assignment of reports to report generation locations is fixed, the problem is reduced to a simple plant location problem. Using similar arguments, one can show that, when the assignment of data sources to potential database locations is given, i.e., an $\hat{X}_{ij}$ vector has been selected, *Problem QIP* is reduced to:

*Problem XIP*

$$Z_{XIP} = \min \left\{ \sum_{j \in J_x} Z_j P_j + \sum_{r \in R} \sum_{j \in J} Y_{jr} \ddot{U}_{jr} \right\}$$

subject to:

$$\sum_{j \in J} Y_{jr} = 1 \qquad r \in R,$$

$$Y_{jr} \leq Z_j \qquad r \in R, j \in \ddot{J}_x,$$

$$Y_{jr}, Z_j = 0 \text{ or } 1 \qquad r \in R, j \in J,$$

where:

$$J_x = \{j \mid X_{ij} = 1 \text{ and } i \in I, j \in J\}$$

$$\ddot{J}_x = J - J_x$$

$$\ddot{U}_{jr} = U_{jr} + \sum_{i \in S_r} W_{ij_i kr},$$

and where $j_i$ is the index of the location to which the data originating from source $i$ has been assigned (i.e., $X_{ij_i} = 1$).

This problem is also a plant location problem.

*Problems XIP* and *YIP* are the basis for the development of the

following class of heuristics to the distributed system configuration problem. The heuristic consists of the iterative application of those two subproblems in order to improve an initial feasible solution to the problem. A detailed description of the heuristic for configuring a distributed computing system follows:

Step 1—Using some selection criteria, select an initial assignment of reports to report generation locations, i.e., form a $\bar{Y}_{kr}$ vector.

Step 2—Improving the assignment of data sources to database locations:

a. Using the vector $\bar{Y}_{kr}$ as an initial assignment of reports to report generation locations, solve *Problem YIP*.
b. From the solution to *Problem YIP* form an $\bar{X}_{ij}$ vector.

Step 3—Improving the assignment of reports to report generation locations:

a. Using the vector $\bar{X}_{ij}$, which was generated in Step 2b of the algorithm, solve *Problem XIP*.
b. From the solution to *Problem XIP* form a $\bar{Y}_{kr}$ vector.

Step 4—If in a full pass through Steps 2 and 3 of the algorithm, there was no improvement in the objective function value, Stop. Otherwise, go to Step 2.

In each step of the algorithm an attempt is made to improve the assignment of data sources to database locations or the assignment of reports to report generation locations. The process terminates when, in a full pass through the algorithm, there was no further improvement in the solution. Various methods can be used in Step 1 of the algorithm for generating an initial assignment of reports to report generation locations. Different initial assignments could lead to different final solutions by the heuristic. Some of the methods that have been considered are listed:

1. A single location is selected for generating all the reports. This could be the location that minimizes the system costs and involves the evaluation of up to $|J|$ different assignments of data sources to database locations. Each one of those evaluations requires the solution of a plant location problem and might be computationally expensive. Another possibility is to select the starting location as the center of gravity in an approximate gravity model, or to pick the starting node at random.

2. For each report $r$, $r \in R$, the location that minimizes the processing and distribution costs for that report is identified. The union of those locations is selected as a starting solution.

3. The simple procedure that was developed for generating a lower bound to *Problem ILP* provides as a byproduct a feasible assignment of reports to locations, or of users to database locations. Those assignments can be used as input to Step 1 of the heuristic.

A second set of heuristics for the problem is obtained when the role of the $\hat{Y}_{kr}$ and the $\hat{X}_{ij}$ variables is reversed in the suggested heuristic. The correct combination of starting procedures and heuristics can be decided based on their empirical performance in a set of computational experiments.

## VII. NUMERICAL EXAMPLES AND COMPUTATIONAL EXPERIMENTS

The methods that were developed in the previous sections provide a system designer with lower and upper bounds on the value of the optimal solution to the system configuration problem, and with a set of possible designs for such a system. To investigate the quality of the solutions that are generated by those procedures, they were programmed and tested on a set of simulated cases. In this section we present an example of using those procedures, and conduct a set of computational experiments in which the performance of those procedures is tested over a wide range of parameter values. These examples demonstrate the value of the design method and highlight the cases in which the procedure is expected to perform very well. First we present the data generation method for the base case. Then, by varying different parameters, we demonstrate how different cost factors influence the selection of an appropriate architecture for the system. By comparing the cost of the computed solution to the cost of a centralized solution, it is possible to identify the cases in which the distributed configuration clearly dominates the solution offered by a centralized system.

### *7.1 Data generation method for the base case*

The base case attempts to simulate the flow of data and reports in a large organization that is distributed over a large geographic area. The organization consists of a set of points representing manufacturing, distribution, and customer services centers. Those points originate the transactions needed for the management and control activities. The points are grouped into regions; each region has a regional center responsible for managing the entities that belong to its region. The organization has a general headquarters that receives reports and data from the organizational entities and is responsible for managing the organization.

The data generation method consists of the following steps. A rectangle of dimensions 1000 by 2500 was divided into $n$ by $m$ regions of equal dimensions. In each region, the coordinates of $p$ points were generated. Those points serve as data sources and can also be used for placement of databases as well as report generation processes. The geographical center of each one of those regions was set as the regional center location for that region. The geographical center of the rectangle

Table I—Test data for the computational example

| Point $i$ | Coordinates $X_i$ | Coordinates $Y_i$ | Transactions $A_i$ | Entities $E_i$ |
|---|---|---|---|---|
| 1 | 416 | 250 | 20543888 | 4430081 |
| 2 | 811 | 421 | 27995668 | 4661639 |
| 3 | 129 | 455 | 13852528 | 5601720 |
| 4 | 252 | 66 | 23314600 | 4835491 |
| 5 | 416 | 750 | 17652664 | 5938730 |
| 6 | 129 | 681 | 20787928 | 5319019 |
| 7 | 96 | 882 | 26454072 | 4510396 |
| 8 | 132 | 515 | 3892134 | 4835392 |
| 9 | 1249 | 250 | 22320524 | 5149213 |
| 10 | 1087 | 182 | 23819312 | 5394341 |
| 11 | 1579 | 276 | 27018756 | 5172949 |
| 12 | 1501 | 323 | 9757924 | 4795006 |
| 13 | 1249 | 750 | 3260475 | 5874032 |
| 14 | 1155 | 922 | 27545052 | 5244782 |
| 15 | 1364 | 847 | 15831062 | 5272542 |
| 16 | 1490 | 600 | 22183256 | 4646214 |
| 17 | 2082 | 250 | 2963550 | 4512458 |
| 18 | 2488 | 76 | 3077690 | 4119027 |
| 19 | 2417 | 100 | 24784832 | 4996060 |
| 20 | 1742 | 239 | 1123881 | 5207785 |
| 21 | 2082 | 750 | 2104029 | 5316656 |
| 22 | 2116 | 694 | 28796560 | 4985719 |
| 23 | 1891 | 989 | 10662968 | 5162839 |
| 24 | 1784 | 849 | 23130012 | 4246616 |
| 25 | 1250 | 500 | 4496444 | 4408979 |

was set as the location in which the general headquarters resides. In the numerical example that follows the parameters were set to $n = 2$, $m = 3$, and $p = 4$. Points 1, 5, 9, 13, 17, and 21 are regional centers, while point 25 represents the general headquarters. Regional center 1 controls and receives reports from points 1 to 4, 5 controls points 5 to 8, and so on.

For each point $i$ a tuple $(A_i, E_i)$ was generated, where: $A_i \sim U(10^6; 3 \times 10^7)$ and $E_i \sim U(4 \times 10^6; 6 \times 10^6)$. $A_i$ represents the amount of monthly update activity that originates in data source $i$, while $E_i$ represents the number of entities in data source $i$, on which data has to be stored in the system databases. Table I contains the coordinates and the $(A_i, E_i)$ values for the different points.

The set $R$ is partitioned into $R_1$-daily, $R_2$-weekly, $R_3$-monthly, $R_4$-quarterly, and $R_5$-yearly subsets of report types. Those reports are generated in a frequency of 365, 52, 12, 4, and one time per year. For each point $i$ a daily and a weekly report are generated (reports 1 to 50 in Table II). A copy of the weekly report is also distributed to the regional center that controls point $i$. For each regional center weekly, monthly, quarterly, and yearly reports are generated. Those are reports 51 to 74 in Table II. Monthly, quarterly, and yearly reports are generated for the general headquarters. Those are reports 75, 76, and

Table II—List of reports, their data sources, and end user locations

| Report ID $r$ | Generation Frequency | End User Locations | Set of Data Sources for Report $r$ — $(S_r)$ |
|---|---|---|---|
| 01–25 | Daily | $r$ | $r$ |
| 26–50 | Weekly | $r - 25, RC_r$ | $r - 25$ |
| 51–56 | Weekly | $(r - 51)\cdot 4 + 1$ | $(r - 51)\cdot 4 + 1, \cdots, (r - 51)\cdot 4 + 4$ |
| 57–62 | Monthly | $(r - 57)\cdot 4 + 1$ | $(r - 57)\cdot 4 + 1, \cdots, (r - 57)\cdot 4 + 4$ |
| 63–68 | Quarterly | $(r - 63)\cdot 4 + 1$ | $(r - 63)\cdot 4 + 1, \cdots, (r - 63)\cdot 4 + 4$ |
| 69–74 | Yearly | $(r - 69)\cdot 4 + 1$ | $(r - 69)\cdot 4 + 1, \cdots, (r - 69)\cdot 4 + 4$ |
| 75 | Monthly | 25 | 1–25 |
| 76 | Quarterly | 25 | 1–25 |
| 77 | Yearly | 25 | 1–25 |

($RC_{26} = 1$, $RC_{30} = 5$, $RC_{34} = 9$, $RC_{38} = 13$, $RC_{42} = 17$, $RC_{46} = 21$, $RC_{50} = 25$)

77 in Table II. Table II lists for each report type the set of data sources that provide data for that report, and the list of end user locations to which the report has to be distributed.

The cost entries for the numerical example are based on the yearly operations of the system, and the objective is to minimize the yearly costs of operating it. The cost factors are setup and computer operation costs:

$$P_j = vw500{,}000 \quad j \in J,$$

data collection, transfer, update, and storage costs:

$$V_{ij} = (E_i Q_i C_i + A_i C_2 + A_i \ell_i d_{ij} C_3 V_i)\cdot v,$$

where:

$Q_i = 500$ bytes

$C_1 = 10^{-4}$ \$/ebyte

$C_2 = 10^{-3}$ \$/etransaction

$C_3 - 5 \times 10^{-5}$ \$/ebyte

$\ell_i = 120$ bytes

$V_i = 12$

$v = 0.0005$

and

$d_{ij} = \max\{100;$ 10 times the distance between points $i$ and $j\}$.

The report generation and distribution costs are:

$$U_{kr} = \left(n_r C_{4r} \lambda_r \log_2 \lambda_r + C_{5r} \lambda_r n_r L_r + C_{4r} e_r \log_2 e_r + \sum_{k \in D_r} d_{kr} \Phi_r n_r C_3\right)\cdot u,$$

where:

$$\lambda_r = \sum_{i \in S_r} A_i V_i / n_r,$$

$$e_r = 0.25 \sum_{i \in S_r} E_i,$$

$L_r = 100,$

$D_r$ = the set of locations that receive the output of report $r$,

$n_r$ = the number of times per year that report $r$ is generated,

$\Phi_r$ = the amount of output generated by report $r$ (in bytes).

$\Phi_r$ is given by:

$$\Phi_r = \lambda_r L_{ir} + 4 e_r L_{2r}$$

$$L_{1r} = \begin{cases} 60 & r \in R_1 \\ 50 & r \in R - R_1 \end{cases}$$

$$L_{2r} = \begin{cases} 200 & r \in R_1 \\ 250 & r \in R_2 \\ 300 & r \in R_3 \\ 500 & r \in R_4 \text{ or } r \in R_5 \end{cases}$$

$$C_{4r} = \begin{cases} 10^{-6} \text{ when } r \text{ is a report that goes to a regional} \\ \quad \text{center or the headquarters} \\ 2 \times 10^{-6} \text{ otherwise} \end{cases}$$

$$C_{5r} = 3 \times C_{4r}$$

$$\mu = 0.0003.$$

Data retrieval and transfer costs are given by:

$$W_{ijkr} = [C_{4r} g_{ir} (A_i V_i / n_r) \log_2 (A_i V_i / n_r) + C_{4r} h_{ir} E_i \log_2 E_i + (C_1 s_1 g_{ir} A_i V_i / n_r + C_1 s_2 h_{ir} E_i) d_{jk}] \cdot w$$

where:

$g_{ir} = 100, \quad h_{ir} = 250, \quad s_1 = 0.1, \quad s_2 = 0.25$ and $w = 0.0015.$

$s_1$ and $s_2$ are the appropriate compression factors between the inputs and outputs from the databases.

The heuristic and the lower bounding procedures have been applied to the base case. The heuristic procedure was initiated with a solution in which all the databases were assigned to point 25 (the headquarters). After five iterations the heuristic stopped with a solution having a yearly cost of approximately *vw*9,090,000. Seven computers were assigned to the system; all of them are used for database assignment and

Table III—Assignment of data sources to database locations (for base case)

| Database Locations | Data Sources Assigned to those Databases |
|---|---|
| 1 | 1, 2, 3, 4 |
| 5 | 5, 6, 7, 8 |
| 9 | 9, 10, 11, 12 |
| 13 | 13, 14, 15 |
| 17 | 17, 18, 19, 20 |
| 21 | 21, 22, 23, 24 |
| 25 | 16, 25 |

for report generation processes. The 9 million dollars consist of 3.5 million dollars for computer setup and operational costs, *vw*1,655,200 for data collection transfer update and storage costs, *vw*3,509,350 for report generation and distribution costs, and *vw*426,340 for data retrieval and transfer costs. Tables III and IV present the assignment of data sources and report generation processes to computer locations.

The simple lower bounding procedure when applied to the base case has generated a lower bound value of *vw*8,662,100, assuring that the gap to optimality is at most *vw*422,000. The cost of a centralized solution in which only one computer is assigned to point 25 and all the databases and report generation processes are assigned to it is *vw*21,022,350, demonstrating that a distributed architecture is clearly preferable for the base case.

### *7.2 The impact of cost changes on system configuration*

One of the important aspects of having a model for configuring distributed computing systems is the ability to perform sensitivity analysis in which the impact of changes in different parameters that influence such a design are investigated. Many cost factors and activity parameters influence the final design of a computing system. Many of those parameters have to be estimated long before the system is actually developed and implemented. Therefore, it is important to be able to investigate well in advance what the impact of changes will be in parameter values and the assumptions used to derive them.

In this section we demonstrate the use of the model to illustrate how the final configuration changes with changes in parameter values. The method used in those investigations was to multiply a parameter by a factor that was varied during the experiments. A factor value of 1 is identical to the base case that was described in the previous section.

In Table V we demonstrate the impact of changes in setup and operational costs on system configuration. The table contains:

Table IV—Assignment of report generation processes to computer locations (for base case)

| Report $r$ | Assigned to | Report $r$ | Assigned to | Report $r$ | Assigned to |
|---|---|---|---|---|---|
| 1 | 1 | 27 | 1 | 53 | 9 |
| 2 | 1 | 28 | 1 | 54 | 13 |
| 3 | 1 | 29 | 1 | 55 | 17 |
| 4 | 1 | 30 | 5 | 56 | 21 |
| 5 | 5 | 31 | 5 | 57 | 1 |
| 6 | 5 | 32 | 5 | 58 | 5 |
| 7 | 5 | 33 | 5 | 59 | 9 |
| 8 | 5 | 34 | 9 | 60 | 13 |
| 9 | 9 | 35 | 9 | 61 | 17 |
| 10 | 9 | 36 | 9 | 62 | 21 |
| 11 | 9 | 37 | 9 | 63 | 1 |
| 12 | 9 | 38 | 13 | 64 | 5 |
| 13 | 13 | 39 | 13 | 65 | 9 |
| 14 | 13 | 40 | 13 | 66 | 13 |
| 15 | 13 | 41 | 13 | 67 | 17 |
| 16 | 25 | 42 | 17 | 68 | 21 |
| 17 | 17 | 43 | 17 | 69 | 1 |
| 18 | 17 | 44 | 17 | 70 | 5 |
| 19 | 17 | 45 | 17 | 71 | 9 |
| 20 | 17 | 46 | 21 | 72 | 13 |
| 21 | 21 | 47 | 21 | 73 | 17 |
| 22 | 21 | 48 | 21 | 74 | 21 |
| 23 | 21 | 49 | 21 | 75 | 25 |
| 24 | 21 | 50 | 25 | 76 | 25 |
| 25 | 25 | 51 | 1 | 77 | 25 |
| 26 | 1 | 52 | 5 | | |

Table V—The impact of changes in setup costs ($P$) on system configuration

| | System Costs (in Thousands) | | | | | Computer Locations | | |
|---|---|---|---|---|---|---|---|---|
| Factor | $\Sigma P_j Z_j$ | $\Sigma V_{ij} X_{ij}$ | $\Sigma U_{kr} Y_{kr}$ | $\Sigma W X_{ij} Y_{kr}$ | Total | Data-bases | Report Gener-ation | Total |
| 32 | 16000 | 5173.54 | 15346.17 | 2.64 | 36522 | 1 | 1 | 1 |
| 16 | 8000 | 5173.54 | 15346.17 | 2.64 | 28522 | 1 | 1 | 1 |
| 8 | 8000 | 3640.56 | 10847.64 | 246.36 | 22734 | 2 | 2 | 2 |
| 4 | 6000 | 2678.10 | 7264.51 | 409.21 | 16351 | 3 | 3 | 3 |
| 2 | 5000 | 2066.68 | 4886.43 | 358.52 | 12311 | 5 | 5 | 5 |
| 1 | 3500 | 1655.20 | 3509.35 | 426.34 | 9090 | 7 | 7 | 7 |
| 1/2 | 1750 | 1655.20 | 3509.35 | 426.34 | 7340 | 7 | 7 | 7 |
| 1/4 | 1125 | 1283.62 | 3509.35 | 483.64 | 6401 | 9 | 7 | 9 |
| 1/8 | 1000 | 331.98 | 3509.35 | 741.65 | 5582 | 16 | 7 | 16 |
| 1/16 | 561.6 | 147.83 | 3509.35 | 809.75 | 5028 | 18 | 7 | 18 |
| 1/32 | 343.2 | 22.4 | 3509.35 | 828.12 | 4703 | 22 | 7 | 22 |

- The total cost of the solution generated by the heuristic and the breakdown of the total cost into its major components
- The total number of computer locations used by that solution, how many of them are used for database placement, and how many for report generation processes.

Table VI—The impact of changes in setup costs ($P$) on the heuristic, lower bound, and centralized solution values

| Factor | Lower Bound on Optimal Solution | Cost of Heuristic Solution | Cost of Lower Bound Policy | Cost of Best Heuristic | Cost of Centralized Solution |
|---|---|---|---|---|---|
| 32 | 36517.0 | 36522.35 | 36522.35 | 36522.34 | 36522.35 |
| 16 | 28517.0 | 28522.35 | 28522.35 | 28522.35 | 28522.35 |
| 8 | 21939.8 | 22734.57 | 22351.82 | 22351.82 | 24522.35 |
| 4 | 15939.8 | 16351.82 | 16351.82 | 16351.82 | 22522.35 |
| 2 | 11950.5 | 12311.63 | 12311.64 | 12311.63 | 21522.35 |
| 1 | 8662.1 | 9090.90 | 9090.90 | 9090.90 | 21022.35 |
| 1/2 | 6912.1 | 7340.90 | 7340.90 | 7340.90 | 20772.35 |
| 1/4 | 5689.1 | 6401.62 | 6425.32 | 6401.62 | 20647.35 |
| 1/8 | 4776.2 | 5582.98 | 5597.27 | 5582.98 | 20584.85 |
| 1/16 | 4196.7 | 5028.54 | 5042.66 | 5028.54 | 20553.60 |
| 1/32 | 3868.9 | 4703.07 | 4708.95 | 4703.07 | 20537.95 |

Increases in setup costs lead to solutions that are closer to a centralized configuration. As the cost of computing is reduced we observe a decrease in system costs and a move towards a distributed solution. In Table VI we compare, for each one of the above cases, the values of two heuristics, one which is identical to the heuristic described in the previous section. The second one uses the solution generated by the simple lower bounding procedure, as a starting solution for the heuristic. The value generated by the best of those two heuristics is compared to the value of a centralized solution and to the lower bound value. The gap between a centralized solution and a distributed solution increases as computing costs decrease. It is also interesting to note that the gap between the feasible solution and the lower bound is small and is nonsignificant when compared to the gap to the centralized solution.

In Tables VII and VIII a similar analysis is made of the impact of changes in data collection, transfer, and update costs ($V$) on system configuration. We observe a trend towards a distributed architecture as those costs increase. The number of database locations increases with the increase in data collection costs. Setting the data collection and transfer costs to zero reduces the number of database locations opened in the system. However, this number is not reduced to one. The reason for this is the fact that report generation and distribution costs have not been reduced and require more than one database in the system in order not to increase further the report generation and distribution costs. In this case, too, the gap between the best heuristic and the lower bound is nonsignificant, assuring that the feasible solutions are very close to the optimal ones.

Tables IX and X summarize the results of changing the report generation and distribution costs ($U$) on system configuration. Here

Table VII—The impact of changes in data collection/transfer and update costs ($V$) on system configuration

| | System Costs (in Thousands) | | | | | Computer Locations | | |
|---|---|---|---|---|---|---|---|---|
| Factor | $\Sigma P_j Z_j$ | $\Sigma V_{ij} X_{ij}$ | $\Sigma U_{kr} Y_{kr}$ | $\Sigma W X_{ij} Y_{kr}$ | Total | Data-bases | Report Gener-ation | Total |
| 32 | 11000 | 604.80 | 3509.35 | 836.95 | 15951 | 22 | 7 | 22 |
| 16 | 10500 | 555.27 | 3509.35 | 842.35 | 15406 | 21 | 7 | 21 |
| 8 | 9000 | 1154.71 | 3509.35 | 818.58 | 14482 | 18 | 7 | 18 |
| 4 | 6000 | 2729.13 | 3509.35 | 733.68 | 12972 | 12 | 7 | 12 |
| 2 | 3500 | 3310.41 | 3509.35 | 426.34 | 10746 | 7 | 7 | 7 |
| 1 | 3500 | 1655.20 | 3509.35 | 426.34 | 9090 | 7 | 7 | 7 |
| 1/2 | 3500 | 827.60 | 3509.35 | 426.34 | 8263 | 7 | 7 | 7 |
| 1/4 | 3500 | 413.80 | 3509.35 | 426.34 | 7849 | 7 | 7 | 7 |
| 1/8 | 3500 | 580.09 | 3763.36 | 507.76 | 8351 | 4 | 7 | 7 |
| 1/16 | 3500 | 323.07 | 3800.57 | 534.65 | 8158 | 2 | 7 | 7 |
| 1/32 | 3500 | 161.53 | 3800.57 | 534.65 | 7996 | 2 | 7 | 7 |

Table VIII—The impact of changes in the data collection/transfer and update costs ($V$) on the heuristic, lower bound, and centralized solution values

| Factor | Lower Bound on Optimal Solution | Cost of Heuristic Solution | Cost of Lower Bound Pol-icy | Cost of Best Heuristic | Cost of Cen-tralized Solu-tion |
|---|---|---|---|---|---|
| 32 | 15112.0 | 15951.10 | 15951.10 | 15951.1 | 181402.25 |
| 16 | 14562.4 | 15406.98 | 15406.98 | 15406.98 | 98625.52 |
| 8 | 13661.9 | 14482.65 | 14482.65 | 14482.65 | 57237.16 |
| 4 | 12236.4 | 12972.17 | 12972.18 | 12972.17 | 36542.99 |
| 2 | 10317.5 | 10746.11 | 10746.12 | 10746.11 | 26195.90 |
| 1 | 8662.1 | 9090.90 | 9090.90 | 9090.90 | 21022.35 |
| 1/2 | 7834.6 | 8263.30 | 8263.30 | 8263.30 | 18435.58 |
| 1/4 | 7420.8 | 7849.50 | 7849.50 | 7849.50 | 17142.20 |
| 1/8 | 7214.0 | 8351.22 | 7642.60 | 7642.60 | 16495.50 |
| 1/16 | 7110.5 | 8158.29 | 7539.15 | 7539.15 | 16172.15 |
| 1/32 | 7058.9 | 7996.75 | 7487.42 | 7487.42 | 16010.48 |

again we observe a reduction in the number of databases and report generation locations as the value of $U$ is reduced. Reducing the report generation and distribution costs to a negligible level still requires two computer locations for an efficient solution. The reason for it is the fact that the data collection transfer and storage costs have not been changed. They are still significant in the total system costs and can be reduced only by having more than one database location in the system. The gap between a distributed and centralized solution increases as the value of $U$ is increased. Here again we observe a nonsignificant gap between the heuristic and the simple lower bounding procedure.

Tables XI and XII summarize the impact of changes in interprocess communication costs $W$ on system configuration. A decrease in $W$

Table IX—The impact of changes in report generation and distribution costs ($U$) on system configuration

| | System Costs (in Thousands) | | | | | Computer Locations | | |
|---|---|---|---|---|---|---|---|---|
| Factor | $\Sigma P_j Z_j$ | $\Sigma V_{ij} X_{ij}$ | $\Sigma U_{kr} Y_{kr}$ | $\Sigma W X_{ij} Y_{kr}$ | Total | Data-bases | Report Gener-ation | Total |
| 32 | 3500 | 1655.20 | 112299.27 | 426.34 | 117880 | 7 | 7 | 7 |
| 16 | 3500 | 1655.20 | 56149.63 | 426.34 | 61731 | 7 | 7 | 7 |
| 8 | 3500 | 1655.20 | 28074.81 | 426.34 | 33656 | 7 | 7 | 7 |
| 4 | 3500 | 1655.20 | 14037.40 | 426.34 | 19618 | 7 | 7 | 7 |
| 2 | 3500 | 1655.20 | 7018.70 | 426.34 | 12600 | 7 | 7 | 7 |
| 1 | 3500 | 1655.20 | 3509.35 | 426.34 | 9090 | 7 | 7 | 7 |
| 1/2 | 2500 | 2066.68 | 2443.21 | 358.52 | 7368 | 5 | 5 | 5 |
| 1/4 | 1500 | 2678.10 | 1816.12 | 409.21 | 6403 | 3 | 3 | 3 |
| 1/8 | 1000 | 3640.56 | 1355.95 | 246.36 | 6242 | 2 | 2 | 2 |
| 1/16 | 1000 | 3640.56 | 677.97 | 246.36 | 5564 | 2 | 2 | 2 |
| 1/32 | 1000 | 3640.56 | 338.98 | 246.36 | 5225 | 2 | 2 | 2 |

Table X—The impact of changes in the report generation and distribution costs ($U$) on the heuristic, lower bound, and centralized solution values

| Factor | Lower Bound On Optimal Solution | Cost of Heuristic Solution | Cost of Lower Bound Policy | Cost of Best Heuristic | Cost of Centralized Solution |
|---|---|---|---|---|---|
| 32 | 117452.2 | 117880.8 | 117880.8 | 117880.8 | 496753.5 |
| 16 | 61302.9 | 61731.2 | 61731.2 | 61731.2 | 251214.9 |
| 8 | 33227.7 | 33656.3 | 33656.3 | 33656.3 | 128445.5 |
| 4 | 19190.3 | 19618.9 | 19618.9 | 19618.9 | 67060.8 |
| 2 | 12171.6 | 12600.2 | 12600.2 | 12600.2 | 36368.5 |
| 1 | 8662.1 | 9090.9 | 9090.9 | 9090.9 | 21022.3 |
| 1/2 | 6851.7 | 7368.4 | 7284.3 | 7284.3 | 13349.2 |
| 1/4 | 5766.7 | 6403.4 | 6199.8 | 6199.8 | 9512.7 |
| 1/8 | 5042.4 | 6242.8 | 5409.6 | 5409.6 | 7594.4 |
| 1/16 | 4596.7 | 5564.9 | 5340.0 | 5340.0 | 6635.3 |
| 1/32 | 4308.8 | 5225.9 | 4832.6 | 4832.6 | 6155.7 |

increases the incentive to move towards a distributed solution. As $W$ increases the gap between the heuristic and the centralized solution is decreased. However, the number of computer locations is not reduced to one, mainly due to the nonnegligible data collection and report distribution costs. When interprocess communication costs are significant, we also observe a significant gap between the lower bound value and the heuristic solution value. This might be one of the cases in which it is worthwhile to apply the Lagrangian-based lower bounding procedure.

In the last set of experiments we investigate the impact of uniform changes in communication costs on system configuration. The method by which those changes were entered into the data was to multiply the distance measures $\{d_{ij}\}$, which were defined in the previous section, by

Table XI—The impact of data retrieval and transfer costs ($W$) on system configuration

| | System Costs (in Thousands) | | | | | Computer Locations | | |
|---|---|---|---|---|---|---|---|---|
| Factor | $\Sigma P_j Z_j$ | $\Sigma V_{ij} X_{ij}$ | $\Sigma U_{kr} Y_{kr}$ | $\Sigma W X_{ij} Y_{kr}$ | Total | Data-bases | Report Gener-ation | Total |
| 32 | 2500 | 5173.54 | 9258.75 | 1908.21 | 18840 | 1 | 5 | 5 |
| 16 | 2500 | 5173.54 | 6222.17 | 3032.75 | 16298 | 1 | 5 | 5 |
| 8 | 3000 | 5173.54 | 5479.41 | 1694.65 | 15437 | 1 | 6 | 6 |
| 4 | 3000 | 5173.54 | 4848.00 | 1281.55 | 14303 | 1 | 6 | 6 |
| 2 | 3500 | 1655.20 | 3509.35 | 852.68 | 9517 | 7 | 7 | 7 |
| 1 | 3500 | 1655.20 | 3509.35 | 426.34 | 9090 | 7 | 7 | 7 |
| 1/2 | 3500 | 1655.20 | 3509.35 | 213.17 | 8877 | 7 | 7 | 7 |
| 1/4 | 3500 | 1655.20 | 3509.35 | 106.58 | 8771 | 7 | 7 | 7 |
| 1/8 | 3500 | 1655.20 | 3509.35 | 53.29 | 8717 | 7 | 7 | 7 |
| 1/16 | 3500 | 1655.20 | 3509.35 | 26.64 | 8691 | 7 | 7 | 7 |
| 1/32 | 3500 | 1655.20 | 3509.35 | 13.32 | 8677 | 7 | 7 | 7 |

Table XII—The impact of changes in data retrieval and transfer costs ($W$) on the heuristic, lower bound, and centralized solution values

| Factor | Lower Bound on Optimal Solution | Cost of Heu-ristic Solu-tion | Cost of Lower Bound Policy | Cost of Best Heuristic | Cost of Cen-tralized Solu-tion |
|---|---|---|---|---|---|
| 32 | 8744.7 | 18840.51 | 22307.59 | 18840.51 | 21104.26 |
| 16 | 8701.7 | 16928.46 | 15486.08 | 15486.08 | 21061.98 |
| 8 | 8680.7 | 15347.61 | 12075.32 | 12075.32 | 21040.85 |
| 4 | 8670.9 | 14303.10 | 10369.94 | 10369.94 | 21030.28 |
| 2 | 8664.9 | 9517.25 | 9517.25 | 9517.25 | 21024.99 |
| 1 | 8662.1 | 9090.90 | 9090.90 | 9090.90 | 21022.35 |
| 1/2 | 8660.8 | 8877.73 | 8877.73 | 8877.73 | 21021.03 |
| 1/4 | 8660.4 | 8771.14 | 8771.15 | 871.14 | 21020.37 |
| 1/8 | 8660.0 | 8717.85 | 8717.85 | 8717.85 | 21020.04 |
| 1/16 | 8659.9 | 8691.20 | 8691.21 | 8691.20 | 21019.87 |
| 1/32 | 8659.6 | 8677.88 | 8677.88 | 8677.88 | 21019.79 |

a factor that was changed during the experiments. A change in distances implies similar changes in

- Data collection transfer and storage costs
- Interprocess communication costs
- Report generation and distribution costs.

The results of those experiments are summarized in Tables XIII and XIV. As expected, a uniform reduction in communication costs leads to a centralized solution, while an increase in communication costs leads towards a distributed solution. The gap between a centralized and the distributed solution increases as communication costs increase. A similar trend is observed between the lower bound and the heuristic solution.

Table XIII—The impact of uniform changes in communication distances ($D$) on system configuration

| | System Costs (in Thousands) | | | | | Computer Locations | | |
|---|---|---|---|---|---|---|---|---|
| Factor | $\Sigma P_j Z_j$ | $\Sigma V_{ij} X_{ij}$ | $\Sigma U_{kr} Y_{kr}$ | $\Sigma W X_{ij} Y_{kr}$ | Total | Data-bases | Report Gener-ation | Total |
| 32 | 11000 | 494.4 | 111923.3 | 26480.6 | 149898.9 | 22 | 7 | 22 |
| 16 | 9000 | 2276.1 | 55967.7 | 12944.1 | 80187.9 | 18 | 7 | 18 |
| 8 | 8000 | 2619.4 | 27989.9 | 5926.1 | 44535.5 | 16 | 7 | 16 |
| 4 | 5000 | 4364.7 | 14000.9 | 2156.9 | 25522.6 | 10 | 7 | 10 |
| 2 | 3500 | 3309.3 | 7006.5 | 850.7 | 14666.6 | 7 | 7 | 7 |
| 1 | 3500 | 1655.2 | 3509.3 | 426.3 | 9090.9 | 7 | 7 | 7 |
| 1/2 | 2500 | 1033.6 | 2447.5 | 180.3 | 6161.5 | 5 | 5 | 5 |
| 1/4 | 1500 | 535.8 | 1457.7 | 83.5 | 3577.1 | 3 | 3 | 3 |
| 1/8 | 500 | 517.1 | 1536.5 | 2.6 | 2556.3 | 1 | 1 | 1 |
| 1/16 | 500 | 7.3 | 24.8 | 2.6 | 534.7 | 1 | 1 | 1 |
| 1/32 | 500 | 7.3 | 24.8 | 2.6 | 534.7 | 1 | 1 | 1 |

Table XIV—The impact of uniform changes in communication distances ($D$) on the heuristic, lower bound, and centralized solution values

| Factor | Lower Bound on Optimal Solution | Cost of Heuristic Solution | Cost of Lower Bound Policy | Cost of Best Heuristic | Cost of Centralized Solution |
|---|---|---|---|---|---|
| 32 | 123303.1 | 149898.9 | 150070.3 | 149898.9 | 657069.9 |
| 16 | 66915.5 | 80187.9 | 80387.2 | 80187.9 | 328787.4 |
| 8 | 38099.9 | 44535.5 | 44646.2 | 44535.5 | 164646.1 |
| 4 | 22717.2 | 25522.6 | 25651.4 | 25522.6 | 82575.2 |
| 2 | 13813.0 | 14666.6 | 14666.6 | 14666.6 | 41539.9 |
| 1 | 8662.1 | 9090.9 | 9090.9 | 9090.9 | 21022.35 |
| 1/2 | 5978.4 | 6161.5 | 6161.5 | 6161.5 | 10763.4 |
| 1/4 | 3491.1 | 3577.1 | 3577.1 | 3577.1 | 4608.3 |
| 1/8 | 2450.2 | 2556.3 | 2479.4 | 2479.4 | 2556.3 |
| 1/16 | 529.8 | 534.7 | 534.7 | 534.7 | 534.7 |
| 1/32 | 529.8 | 534.7 | 534.7 | 534.7 | 534.7 |

## VIII. POTENTIAL APPLICATIONS AND EXTENSIONS OF THE MODEL

The optimization models for configuring distributed computer systems, which were formulated in the previous sections, were based on simplifying assumptions. Those assumptions restrict the use of the model to a subset of the organizations in which we expect that distributed computing systems will be widely implemented. Some of those restrictive assumptions can be relaxed, and the model can be extended to a wider range of application areas. In this section we present a few of those extensions and provide examples to existing or planned systems that are prime candidates for using this model. Each of these systems has unique characteristics that are exploited in the models and the suggested solution methods.

### *8.1 Allowing for different transaction classes*

The model developed in Section IV has assumed that all the transactions that originate in a data source are assigned to the same database location. This assignment is regardless of the set of entities and functions that are being served by the data contained in those transactions. This is rather a highly restrictive assumption. In many of today's organizations it is difficult to associate a single data source with a unique set of functions that are served by the transactions that originate in that data source. Situations in which the same data source, and sometimes the same individual, provides input data to many databases, and to a variety of sometimes conflicting functions, are quite common. For example, a salesperson in a company sales office can report orders for final products that were received from customers, which will update a database that supports the order handling subsystem. The same salesperson can also enter data on payments made by customers, fill out presence or absenteeism reports that will update the personnel and employee history database, or can enter customer reports on problems that they have encountered with equipment installed by the company (those transactions will typically update an engineering or technical support database). When larger entities are used as data sources (such as departments, branch offices, plants, or divisions), then the association between data sources and functions becomes less apparent. For example, a job shop in a large manufacturing company originates and feeds transactions that deal with different activities in the organization. Transactions can report on the progress of jobs in the production process, the consumption of inventory items in those jobs, transfer of final goods to inventory, machine and equipment failure, presence or absence of employees. Those transactions will be routed to different databases and will be used by completely different sets of end users in the organization. The events reported by those transactions occur at different rates, and they are of interest to completely different sets of end users that might be located in different regions of the country, where each one of those groups might have their own reporting requirements.

The restriction that all the transactions that originate in a single location will be routed to the same database location, can, in many instances, increase the data processing costs to the organization that is being served by the data processing department and, at the same time, lengthen the response times to user requests for reports. Organizations that have such characteristics can benefit from relaxing this restriction. It might be possible to reduce the system's data processing costs and improve its response time by routing transactions that serve identical functions to database locations that are different from those that serve other functions even if the transactions originate from the

same data source. Relaxing this assumption slightly complicates the routing of transactions in the network, but is justified by the potential reduction in the system costs.

Defining by $F$ the index set of the various functions in the organization, it is possible to partition the set of transactions that originate in source $i$, $i \in I$, into up to $|F|$ subsets of transactions, where each subset corresponds to a specific function in the organization. Letting $i_f$ denote the subset of transactions in source $i$ that are used by function $f$, $f \in F$ in the organization, this problem can be formulated as follows.

Find binary variables $Z_j$, $X_{mj}$, $Y_{kr}$ that satisfy

$$Z_{FIQP} = \min \left\{ \sum_{j \in J} Z_j P_j + \sum_{j \in J} \sum_{f \in F} \sum X_{mj} V_{mj} + \sum_{r \in R} \sum_{k \in J} Y_{kr} U_{kr} + \sum_{r \in R} \sum_{k \in J} \sum_{j \in J} \sum_{f \in F} \sum_{m \in i_f} X_{mj} Y_{kr} W_{mjkr} \right\}$$

subject to eqs. (3) through (5) and

$$\sum_{j \in J} X_{mj} = 1 \quad m \in i_f, f \in F,$$

$$X_{mj} \leq Z_j \quad m \in i_f, f \in F, j \in J,$$

where $V_{mj}$ and $W_{mjkr}$ are the appropriate cost factors for those data sources and functions. This optimization problem has the same characteristics as the single-function/multiple-sources system configuration problem. The only difference between the two problems is in the increase in the number of data sources introduced by the splitting of single data sources into multiple data sources. The same methods that have been developed for solving *Problem ILP* are applicable to the above optimization problem.

### *8.2 Different setup costs for report generation than for data storage*

The hardware and software requirements of a node in the distributed system could be dependent on the activities performed in the node. For example, different setup costs might be incurred for supporting a database versus supporting a report generation process, or for supporting both activities.

Letting $P_{jt}$, $t = \{1, 2, 3\}$ be the setup costs for placing in location $j$ a computer that can support a: database $\{1\}$, report generation process $\{2\}$, or both activities $\{3\}$. We redefine the $Z_j$ variables in *Problem QIP* and *Problem ILP* as the sum of three variables, $Z_{jt}$, $t = \{1, 2, 3\}$. $Z_{jt}$ are binary variables that specify what type of capabilities will be provided in location $j$.

This problem has the same formulation as *Problem QIP* or *Problem*

*ILP*, with the $Z_j$ variables replaced by the sum $\sum_{i=1}^{3} Z_{jt}$ and the addition of the constraints

$$\sum_{i=1}^{3} Z_{jt} \leq 1 \quad j \in J.$$

Techniques similar to the ones used for solving the previous models can be used to generate solutions for this problem.

### *8.3 Transactions that update multiple databases*

A situation that arises in many systems is one where a transaction is used to update two or more functionally different databases that might be located in different locations. For example, a transaction that reports on the transfer of inventory items from one regional warehouse to another warehouse might have to update two fragments of the inventory database that are located in different locations. Transfer of employees from one region to another, transfer of items from inventory to the production process, transfer of final products from the production process to the distribution system, or third party charging of long distance phone calls are other examples in which there might be a need to simultaneously update more than one fragment of a database.

This problem can be handled in a similar fashion to the one used for modeling different transaction classes by splitting the flow of transactions according to the databases that they have to update and introducing a new set of variables and costs associated with the synchronization of simultaneous updates to multiple databases.

### *8.4 Allowing for multiple copies of databases*

An underlying assumption of the model was that there is only a single copy of each fragment of the database in the system. In some cases it is possible to save on retrieval costs by allowing for a partial or full overlap between databases. An example in which such duplication can be useful is in a videotex system in which we can have full or partial overlap between databases. Models that select the fragments to be duplicated, decide which portion of each fragment will be duplicated, and assign each fragment to a computer location are not easy to formulate and will require further research.

## IX. CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

We have developed models and algorithms for designing an architecture of distributed computer systems. The algorithms have been tested on a few examples and have performed well by generating good initial designs for those systems. Research is now in progress on

extending those methods by developing better procedures for computing lower bounds on the optimal solution and improved heuristics for generating feasible solutions for the problem. Despite the restrictive assumptions that have been used for developing the models, we believe that the developed procedures are valuable tools for helping the preliminary design of distributed computing systems.

We have already pointed out the possible extensions of the formulations and the models. Another extension that might be worthwhile to pursue includes multiperiod versions of the problem. The models developed in this paper are single-period versions of the problem, which are well suited for the design of stable systems. In reality some systems go through transitional expansion periods for which multiperiod versions of the problem have to be developed. The models presented here are also uncapacitated versions of the problem. Again, in reality we expect to have capacitated or capacity-dependent setup and operational cost versions of the problem.

## X. ACKNOWLEDGMENT

I wish to express my appreciation to Dr. Liebert for raising many of the issues involved in the adoption of a distributed computing system to commercial organizations and for providing helpful comments on earlier versions of the paper; to Dr. Wakid for introducing me to TCAS and to testing, maintenance, and provisioning operations in the telephone system; Dr. Ormsby for handling the editing and typesetting of the paper, Dr. Friedel for many helpful discussions and comments on earlier versions of the paper; and to Mr. Altinkemer for programming the algorithms and performing the reported computational experiments.

## REFERENCES

1. C. A. Ellis, "A Robust Algorithm for Updating Duplicate Data Bases," Berkeley Workshop on Distributed Data Management and Computer Networks, University of California at Berkeley (May 1977), pp. 146–58.
2. M. Hammer and D. Shipman, "An Overview of Reliability Mechanisms for a Distributed Data Base System," IEEE Tutorial: Centralized and Distributed Data Base Systems (1979), pp. 645–7.
3. L. Lamport, "Time Clocks and the Ordering of Events in a Distributed System," IEEE Tutorial: Centralized and Distributed Data Base Systems (1979), pp. 588–95.
4. R. J. Ramirez and N. Santoro, "Distributed Control of Updates in Multiple-Copy Data Bases: A Time Optimal Algorithm," Proc., 4th Berkeley Workshop on Distributed Data Management and Computer Networks, University of California at Berkeley, May 1979.
5. P. M. G. Apers, *Query Processing and Data Allocation in Distributed Data Base Systems*, Amsterdam: Free University, 1982.
6. R. M. Shapiro and R. E. Millstein, "Failure Recovery in a Distributed Data Base System," Proc. COMPCOM (Spring 1978), pp. 66–70.
7. R. L. Rivest, A. Shamir, and L. Adelman, "A Method of Obtaining Digital Signatures and Public Key Cryptosystems," Comm. ACM, *21*, No. 2 (February 1978), p. 120.

8. R. G. Casey, "Allocation of Copies of a File in an Information Network," AFIPS 1972 SJCC Conf. Proc. *40* (1972), pp. 617–25.
9. P. Chen and J. Akoka, "Optimal Design of Distributed Information Systems," IEEE Trans. Comput., *C-29,* No. 12 (December 1980), pp. 1068–80.
10. W. W. Chu, "Optimal File Allocation in Multiple Computer Systems," IEEE Trans. Comput., *C-18,* No. 10 (October 1969), pp. 885–9.
11. K. D. Levin and H. L. Morgan, "A Dynamic Optimization Model for Distributed Data Bases," Oper. Res. *26,* No. 5 (September 1978), pp. 824–35.
12. H. L. Morgan and K. D. Levin, "Optimal Program and Data Locations in Computer Networks," Comm. ACM, *20,* No. 5 (May 1977), pp. 315–21.
13. B. Gavish and H. Pirkul, "Allocation of Data Bases and Processors in a Distributed Computing System," *Management of Distributed Data Processing,* J. Akoka, ed., Amsterdam: North-Holland, 1982, pp. 215–31.
14. S. Mahmoud and J. S. Riordan, "Optimal Allocation of Resources in Distributed Information Networks," ACM Trans. Data Base Syst., *1,* No. 1 (March 1976), pp. 66–78.
15. H. Moortgat, "Processor and File Allocation in Distributed Data Base Systems," Ph.D. Dissertation, University of Washington, Seattle, 1979.
16. B. Gavish and A. Segev, "Query Optimization in Distributed Computer Systems," *Management of Distributed Data Processing,* J. Akoka, ed., Amsterdam: North-Holland, 1982, pp. 233–52.
17. B. Gavish and A. Segev, "Set Query Optimization in Horizontally Partitioned Distributed Data Base Systems," working paper, The Graduate School of Management, The University of Rochester, Rochester, N.Y., 1983.
18. A. R. Hevner and S. B. Yao, "Query Processing in Distributed Data Base Systems," IEEE Trans. Software Eng., *SE-5,* No. 3 (March 1979), pp. 177–87.
19. B. Gavish, "Topological Design of Centralized Computer Networks-Formulations and Algorithms," Networks, *12* (April 1982), pp. 355–77.
20. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* New York, NY: W. H. Freeman and Company, 1979.
21. D. Erlenkotter, "A Dual Based Procedure for Uncapacitated Facility Location," Oper. Res., *26,* No. 1 (January 1978), pp. 992–1009.
22. M. L. Fisher, "Lagrangean Relaxation Method for Solving Integer Programming Problems," Manage. Sci., *27* (January 1981), pp. 1–18.
23. M. Held and R. M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II," Math. Programming, *1* (January 1971), pp. 6–25.
24. B. T. Poljack, "Minimization of Unsmooth Functionals," U.S.S.R. Computational Math. Math. Phys., pp. 14–29. [Translation of "Zurnal Vycislitel 'noi Matematiki i Matematiceskoi Fiziki," *9* (1969), pp. 509–21.]
25. B. Gavish and S. K. Srikanth, "Optimal Solution Methods for Large Scale Multiple Salesman Traveling Salesman Problem," working paper, The Graduate School of Management, The University of Rochester, Rochester, N.Y. (1980).
26. B. Gavish, "Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem," J. ACM, *30,* No. 1 (January 1983), pp. 118–32.

## AUTHOR

**Bezalel Gavish,** B.Sc. (Industrial Management and Engineering), M.Sc. and Ph.D. (Operations Research), Technion–Israel Institute of Technology, Haifa, in 1967, 1970, and 1975, respectively. From 1968 to 1973 he was Department Head of Systems Analysis and Scientific programming in charge of developing large-scale on-line logistics and transportation management computerized systems. From 1973 to 1976 he was Project Head of developing and testing a system for computer-assisted medical diagnosis at the IBM Scientific Center. He joined the Graduate School of Management, University of Rochester, Rochester, NY, in 1976, where he is an Associate Professor in Computers and Information Systems. During that time he was a visiting faculty member at the IBM T. J. Watson Research Center, Bell Laboratories, and the Technion. His research interests span the design and analysis of computer communication networks, design and analysis of distributed computing systems, systems

analysis and design, combinatorial optimization, and scheduling and routing in logistic systems. Member, Association for Computing Machinery, the Operations Research Society of America, The Institute of Management Sciences, IEEE Computer Society. Recipient, 1972 IPA Award, 1982 AIIE Transactions Development and Applications Award.

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Inverted Decision Tables and Their Application: Automating the Translation of Specifications to Programs

By L. S. LEVY* and H. T. STUMP†

(Manuscript received August 24, 1983)

Code generation techniques are used to program an application characterized by complexity arising from many special cases, and rapid changes due to advances in the state of the art. A formal notation—an inverted decision table written in a propositional logic form—is developed as a means for allowing expert users to describe the application in a knowledge base that code generators then can use to create production code. The complete system described in the paper automatically transforms a one thousand-page specification into a running program. The development of this system is an example of the formalization of the specification of a complex application. In this case the application is a part of the Job Management Operations System, an operational support system to aid regional Bell Operating Company construction and engineering processes. The techniques described, however, can be generalized.

## I. INTRODUCTION

A complex applications program that involved enumerating and analyzing many special cases was successfully developed using the notation, tools, and techniques described in this paper. Such enumerative complexity is characteristic of many applications. In such cases a detailed knowledge of the application is in the minds of experts who

*AT&T Bell Laboratories. †AT&T Bell Laboratories; present affiliation Bell Communications Research, Inc.

are best able to state the rules of the game, but are not able to program them. Computer scientists, on the other hand, can program but do not know the application in depth. Thus, the skills of these two groups are complementary. Formal notation and tools enable them to cooperate by allowing them to focus on their respective skills.

In addition many applications have a *knowledge base* or set of rules that changes with advances in the state of the art. Since the most current information about the application is in the hands of expert users, it is desirable for them, and not programmers, to maintain the knowledge base of the program. This is possible when an appropriate notation exists for stating the application information, and when programming tools are available for exploiting this repository of information.

In this paper we discuss the development of a module of the Job Management Operations System (JMOS) application software. This development made a deliberate attempt to incorporate both (1) notation to allow expert users to describe their knowledge base in a formal specification, and (2) programming tools to generate the application code directly from this specification. Updating the programs to reflect changes in the state of the art then consists primarily of modifications to the knowledge base driving the program, and these updates can be performed directly by the expert users.

In the software development method that evolved, the specification is developed using *UNIX*™ operating system shell and editing tools. This specification is in a form that can be used as a basis for a document for users, and also as a basis for mechanical translation into programs. From the specification, decision tables are derived by shell programs. Then these decision tables are translated into PL/1 programs by a decision table processor developed for this application. If the specification has been properly composed, it then becomes a directly compilable program and is the *primary program description.* Although we have not achieved 100 percent of this goal, we feel that it is a realizable objective.

In Section II of this paper we describe the application and those characteristics motivating the present work. In Section III, the evolution of our approach is presented as it proceeded through several iterations. Section IV details the current approach to the specification-design process. The last section is an assessment of this methodology.

Our intended audience is programmers, specifiers (analysts and systems engineers), and managers involved in applications development.

### *1.1 Related work*

Decision tables are well known and have an extensive literature.

Their use in specifications as a methodology relevant to correctness is described in Gerhart and Goodenough.[1] Our inverted decision tables resemble production systems.[2,3] *Computer* contains a survey of recent work in specification languages.[4] Our own efforts in this area are somewhat more special purpose. The emphasis on productivity of programmers is part of the broader concern for productivity gains in the general economy. Jones is a collection of articles describing work in this area.[5]

Code generation, long a part of compiler technology,[6] rises to a higher level in application code generators. Reference 7 is a broad survey of the state of the art in 1977, and specifically discusses decision tables. The work of one of the authors in an earlier application has been reported in Levinson, Levy, and Salisbury.[8] Other recent work is described in Refs. 9 through 14.

## II. PROBLEM DESCRIPTION

### *2.1 The application*

The JMOS system is a predominantly on-line operations support system that tracks and manages construction work in the outside telephone plant.* Its major functions are to

- Analyze data from engineering drawings and compute the work content of the construction job;
- Schedule such jobs to meet due dates, taking into account material availability, job priorities, and construction resources;
- Use daily reports of work progress to track the status of jobs;
- Analyze and report the ongoing performance of the construction forces;
- Interface with accounting, budgeting, and inventory systems to maintain company records, produce payrolls, and track operations and costs.

The application addressed in this paper deals with the first of the functions listed above. In particular it decomposes each work operation specified on an engineering drawing into two sets of components. One set comprises the physical tasks that must be undertaken to complete the work operation. The second set comprises theoretical elements that are used to compute a standard company work measurement index.

Associated with each of the tasks in the former set is a Standard Time Increment (STI) allotted to perform the task. These STIs are

* The outside telephone plant is the physical part of the telecommunications network that extends from the local central office to the customer's premises. It comprises cables, service wires, interconnection facilities, signal regeneration equipment, etc., as well as supporting structures such as poles, guys, anchors, strand, manholes, and conduit.

later accumulated and used as the basis for planning, scheduling, and tracking the construction jobs. Associated with each of the elements in the latter set is a quantity of Work Units (WUs), which are a relative expression of the value and complexity of the element. This measure allows construction work to be compared to dissimilar work in other areas of the telephone business. For example, craft performance is measured in terms of WUs per hour and cost performance is measured as dollars per WU.

To identify the STI components, the application program must derive and correlate information about such things as the nature of the work to be performed (e.g., install a cable), the location (e.g., buried in a trench), the field conditions (e.g., rocky soil), and the equipment to be used (e.g., backhoe). Derivation of the WU elements is similar in concept, but differs significantly in terms of analytic detail. Throughout this paper we will use STIs as illustrations, since they are generally the more complex of the two work components.

As an example, Fig. 1 is an annotated engineering drawing that calls for the installation of a new buried cable from a manhole to a pedestal (splice point). Note that there are four construction work operations, called job steps, in this example. Step 1 involves placing 500 feet of cable, 100 feet in a duct extending from the manhole, and the remaining 400 feet in a trench. Step 2 involves the placement of a terminal,

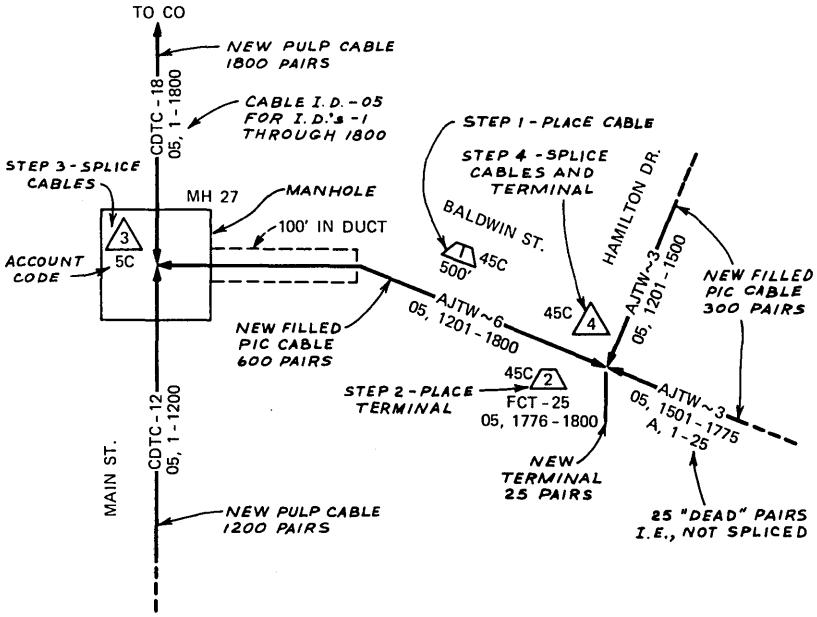



Fig. 1—Sample engineering drawing.

| STEP | STI CODE | DESCRIPTION | QUANTITY | TIME |
|---|---|---|---|---|
| 1 | 5P01 | SITE SETUP – UNDERGROUND PLACING | 1 | 0.9 |
| | 5P14 | PUMP MANHOLE | 1 | 0.3 |
| | 5P10 | PLACE CABLE IN LATERAL DUCT | 1 | 0.8 |
| | 4P01B | SITE SETUP – BURIED CABLE PLACING | 1 | 0.4 |
| | 4P08 | DIG TRENCH WITH TRENCHING MACHINE | 400 | 1.9 |
| | 4P09 | ADD FOR ROCKY SOIL | 400 | 1.0 |
| | 4P13 | PLACE CABLE IN TRENCH | 400 | 0.6 |
| | 4P15 | BACKFILL TRENCH BY MACHINE | 400 | 1.0 |
| | | | TOTAL | 6.9 |
| 2 | 4P17 | PLACE PEDESTAL | 1 | 0.5 |
| | | | TOTAL | 0.5 |
| 3 | S09 | SITE SETUP – UNDERGROUND SPLICE | 1 | 0.7 |
| | S10 | PUMP MANHOLE | 1 | 0.3 |
| | S11 | SET MANHOLE PLATFORM | 1 | 0.2 |
| | S13A | PREPARE CABLE ENDS | 3 | 0.9 |
| | S13C | CLEAN FILLED PAIRS | 600 | 0.6 |
| | S13D | PREPARE CABLE ENDS | 3600 | 2.2 |
| | S22A | CLOSE Y SPLICE – NEW CABLE | 1 | 0.8 |
| | S39 | JOIN PULP PAIRS | 3000 | 4.8 |
| | S41 | JOIN FILLED PIC PAIRS | 600 | 1.3 |
| | S54 | IDENTIFY AND TAG PAIRS | 600 | 4.8 |
| | S55 | PROVIDE TONE FOR TAGGING | 600 | 4.8 |
| | | | TOTAL | 21.4 |
| 4 | S06 | SITE SETUP – BURIED SPLICE | 1 | 0.4 |
| | S13A | PREPARE CABLE ENDS | 3 | 0.9 |
| | S13C | CLEAN FILLED PAIRS | 1200 | 1.2 |
| | S13D | PREPARE CABLE CORE | 1225 | 0.7 |
| | S40 | JOIN AIRCORE PIC PAIRS | 25 | 0.1 |
| | S41 | JOIN FILLED PIC PAIRS | 1175 | 2.6 |
| | S60 | PLACE GRAVEL IN PEDESTAL | 1 | 0.1 |
| | | | TOTAL | 6.0 |

Fig. 2—STI components derived for sample job.

or pedestal, in which the cable will be spliced to other cables and to a termination block, from which service wire can be extended into a customer's premise. Step 3 involves splicing (i.e., connecting) one end of the cable in the manhole to two other new cables installed on another portion of the drawing. Step 4 involves splicing the other end of the cable to two smaller cables and to a distribution terminal.

The STI components of these steps are listed in Fig. 2. Figure 3 illustrates the information supplied by the user, typically a clerk in the engineering office. The function of the application program is to analyze these input data and derive the correct set of work tasks.

### *2.2 Problem complexity*

The complexity of the problem arises from a combination of static and dynamic factors. The static complexity is due to the inherent nature of telephone plant construction. The dynamic complexity of the problem is due to rather rapid changes in construction methods.

| STEP | DATA ITEM NAME | ITEM 1 | ITEM 2 | ITEM 3 | ITEM 4 |
|---|---|---|---|---|---|
| 1 | WORK ENVIRONMENT | B | | | |
| | MATERIAL DESCRIPTION | AJTW-6 | | | |
| | PLANT ACCOUNT CODE | 45C | | | |
| | MATERIAL QUANTITY | 500 | | | |
| | PLACING ACTION | LDUC | STAN | | |
| | ACTION QUANTITY | 100 | 400 | | |
| 2 | WORK ENVIRONMENT | B | | | |
| | MATERIAL DESCRIPTION | FCT-25 | | | |
| | PLANT ACCOUNT CODE | 45C | | | |
| | MATERIAL QUANTITY | 1 | | | |
| | PLACING ACTION | STAN | | | |
| 3 | WORK ENVIRONMENT | U | | | |
| | PLANT ACCOUNT CODE | 5C | | | |
| | FIXTURE FLAG | N | | | |
| | CABLE DESCRIPTION | CSDC-18 | AJTW-6 | CSDC-12 | |
| | CABLE STATUS | N | N | N | |
| | CABLE CATEGORY | C | F | F | |
| | CABLE IDENTITY | 05 | 05 | 05 | |
| | PAIR RANGE | 1-1800 | 1201-1800 | 1-1200 | |
| 4 | WORK ENVIRONMENT | B | | | |
| | PLANT ACCOUNT CODE | 45C | | | |
| | FIXTURE FLAG | Y | | | |
| | CABLE DESCRIPTION | AJTW-6 | AJTW-3 | AJTW-3 | FCT-25 |
| | CABLE STATUS | N | N | N | N |
| | CABLE CATEGORY | C | F | F | F |
| | CABLE IDENTITY (1) | 05 | 05 | 05 | 05 |
| | PAIR RANGE (1) | 1201-1800 | 1201-1500 | 1501-1775 | 1776-1800 |
| | CABLE IDENTITY (2) | | | A | |
| | PAIR RANGE (2) | | | 1-25 | |

Fig. 3—Input data for sample job.

### *2.2.1 Static factors*

Telephone plant construction involves the use of diverse materials, methods, field conditions, equipment, and types of operations. In particular this work employs over five thousand different material items in over 50 classifications. Some 265 different work tasks and 80 work-unit elements characterize just two major classes of work operations—placing and splicing.

Placing work includes the installation, removal, and rearrangement of materials and entails the use of many different types of specialized equipment and methods. There are also four distinct field environments to consider:

1. Aerial—cable and other materials supported by poles;
2. Buried—cable buried directly in the ground;
3. Building—cable and other materials placed on and within buildings; and
4. Underground—cable installed in underground conduits extending between manholes.

The following illustrates the complexity just of the installation of buried cable:

There are two common installation methods—plowing and trench-

ing. With plowing, one of two types of vehicles may be used—one with a static plowshare or one with a vibrating plowshare. With the trenching method, either a backhoe or a specialized trenching machine may be used to dig the trench. In some cases trenches are even dug by hand. For each of these methods and types of equipment the system must assign work tasks and associated times. Additional time is required if the soil is rocky. Furthermore, there are some ten specialized operations, used in conjunction with or in place of these standard methods, that must be selected on a case by case basis for each job step.

Splicing is the second major class of work performed by the construction forces. Here there are fewer methods and types of equipment involved than with placing. However, the normal splicing process is more detailed and complex than its placing counterpart, so that a given step often involves many more work tasks. Consequently, the analysis of splicing input is inherently more difficult and requires the use of rather esoteric algorithms.

For example, consider Step 4 in Fig. 1. As Fig. 1 shows, the input for this splice includes a description of each cable, along with the identification of its component pairs. Consider the cable at the top right side of the splice. It is designated as AJTW-3, which means it contains 300 pairs, has color-coded plastic insulation on the copper wires (known as PIC cable), and is filled with a petroleum jelly compound to render it waterproof so that it can be buried directly in the ground. The pairs in the cable are numbered 05, 1201 through 05, and 1500. Algorithms in the splicing module examine the pairs in each cable and determine which are being connected, disconnected, or rearranged. Subtotals, computed by type of cable (e.g., filled PIC), are used to derive the quantities of STIs and WUs for the splice. For this splice a total of 1175 filled PIC pairs and 25 regular PIC pairs are being joined. Similar analyses are performed to derive the other STIs and WUs for the step.

Describing the work operations outlined above requires the use of five different data-entry formats. The original word specification describing the application was over 150 pages long. The new specification described in this paper is over one thousand pages long.

#### *2.2.2 Dynamic factors*

The application faces additional complexity due to dynamically changing construction operations. New materials, tools, and techniques are constantly emerging. The application must be able to be modified frequently to keep abreast of these changes. For example, lightguide cable is currently being introduced very rapidly. This requires new placing and splicing procedures.

In addition to this normal evolution, the problem was recently complicated further by the reissuance of completely revised sets of both work tasks and work-unit elements. These revisions were triggered by the recent completion of extensive statistical studies of telephone construction methods.

## III. EVOLUTION OF APPROACH

### *3.1 Informal word specification*

Development of the prototype version of JMOS was started in late 1980. At that time the preliminary specifications described the derivation of work tasks (and associated standard times) to be associated with different outside plant construction operations. The document was over 150 pages long and its interpretation required extensive discussions between the software developers and the system engineer.

An example of the style of the original specification is the following:

Original Specification Style

If the step is entered on a Placing Work mask, the work environment is "buried," the material class is "cable," and one of the following conditions is met:

- The placing action is "trench with backhoe," "trench by machine," or "trench by hand."
- The placing action is "standard," the account code classification is "c," and the Profile Table specifies trenching with a backhoe or a trenching machine as the standard buried cable placing method.

Then for each foot of cable, generate:

1 4P13 Plc Ca in Trench

Although this example captures the flavor of the original specification, it does not indicate the level of complexity of the specification or of the changes that resulted from either errors in specification or typing. Suffice it to say that there were pages of the original specification in which the marginal notes and corrections were comparable to the typing on the page. Indeed, the changes were so extensive, and the specification so complex, that it was never reissued with corrections.

### *3.2 Manual translation to design*

At that time it seemed clear that the amount of detail in the specification document, and the need for interpretation, would probably induce a fairly extensive debugging effort once the programs were written. When coupled with the fact that there were also undoubtedly errors in the specification itself, that some parts of the specification were incomplete, and, finally, that the specification would be extensively revised in 1982, we decided to use decision tables.

### 3.3 *Decision table and processor*

Decision tables are a well-known schema for specifying algorithms. They are, in effect, a deterministic version of the production systems currently popular in artificial intelligence applications. The advantages of decision tables are that they are quite change tolerant and fairly readable. Their disadvantage is that they represent a natural solution to only a part of the problem—there are algorithms that do not readily fit into decision table form.

A search of various software resources failed to turn up any decision table processor, both available and supported, that could generate PL/1 code—although we did find several that met two of these three criteria. Accordingly, a decision table processor was built that has been used in JMOS for the development of an operational prototype.

The format of the decision table that was used is shown below:

```
!D-TABLE:sti-p-bu
!VERSION:%I%,DATE,%D%
!=============================================
%logical
  buriedpla
  cable
  backhoe
  trencher
  hand
  standard
  pacc
  opf_backhoe
  opf_trencher
%conditions
  buriedpla
    masktype = 'p' & workenv = 'b'
  cable
    mattype = 'cable'
  backhoe
    plaatn = 'ctrb'
  trencher
    plaatn = 'ctrm'
  hand
    plaatn = 'ctrh'
  standard
    plaatn = 'stan'
  pacc
    paccls = 'c'
  opf_backhoe
    opf_tbl('bu_cbl_mth') = 'backhoe'
```

```
  opf_trencher
    opf_tbl('bu_cbl_mth') = 'trencher'
%actions
  s4p13
    CALL add_sti('4p13',matqty,jobstep_ptr,index,
      total);
    $indexer
%specifications
  buriedpla & cable & (backhoe | trencher | hand |
    (standard & pacc & (opf_backhoe | opf_trencher)))
    % 4p13
```

The interpretation of this decision table specification is as follows:

- Lines beginning with '!' are comment lines and are ignored by the decision table compiler.
- Lines beginning with a '%' are syntactic markers, which denote the headings of sections of the decision table.

There are four sections to the decision table.

1. Logical—This section consists solely of a list of the logical conditions occurring in the decision table. It is used to compile declarations in the object program.

2. Conditions—This section assigns values to the logical variables. In the object program, `buriedpla` will be true if `masktype` is `'p'` and `workenv` is `'b'`. The decision table will compile this into an assignment statement initializing the logical variable.

3. Actions—This section identifies action stubs of the decision table. In general, the actions are a sequence of PL/1 statements and PL/1 preprocessor statements. In the example, `s4p13` consists of a procedure call, and a preprocessor statement.

4. Specifications—This section relates conditions and actions. A set of specific actions to the right of the '%' symbol will be invoked if the Boolean expression to the left of the '%' symbol is true.

The decision table processor is a very straightforward implementation in the *UNIX* system, written in a combination of shell-level tools, primarily `awk`. Although there was no optimization performed in generating object code from the decision table, the performance was quite acceptable supporting our thesis that, in general, application code is not performance limited. However, had performance been a problem, it would have been possible to include optimization in the decision table processor without changing the decision table source. (The source code of the decision table is almost two orders of magnitude larger that the source code of the decision table processor, so clearly the processor would have been the place to optimize.)

The decision table approach did, in fact, accomplish what we thought it would. It was change tolerant, and more readable if only because it

was more concise than the PL/1 code. The decision table source resides in some 17 decision tables, each of which translates into a PL/1 program. These 17 PL/1 programs comprise in excess of ten thousand lines of code.

But once extensive debugging was under way, the decision table did not satisfy our needs completely, since there was no direct way to relate outputs to inputs. The same work task output could be generated by any one of a number of decision tables since these were organized by generic categories of the application. Thus, the problem was to develop a means of tracing an output back to its input.

### *3.4 Decision table inverter*

The format that evolved to solve this problem of correlating outputs to inputs was an *inverted decision table.* These tables, which are generated by shell programs, list each output, the programs that generate each output, and, for each program, the conditions under which the output arises. This inverted decision table, which is an extended form of cross-reference listing, made the debugging easier and became an invaluable aid in this phase of the project.

The concept of a decision table inverter can best be visualized by considering the tabular form of a decision table:

| | | | | | |
|---|---|---|---|---|---|
| Conditions | | | | | |
| | C1 | T | F | T | T |
| | C2 | T | — | F | T |
| | C3 | F | — | — | T |
| Actions | | | | | |
| | A1 | x | x | — | x |
| | A2 | — | x | — | x |

In this standard form of decision table, the relationship between conditions and actions is specified by the vertical columns. Each vertical column corresponds to a given set of input conditions. To determine what output actions apply, one starts with the specification of input conditions and searches for a column that matches those input conditions. The corresponding action portion of that column defines the output actions in that case.

Suppose that one wanted to know what input conditions could generate a given output. Then, even though the information is contained in the decision table, it is not in a convenient form. On the other hand, if one *inverts* the decision table, then the actions appear as the entry points that one uses to search for conditions that could generate those actions. The inverted decision table for our example is

| | | | | | |
|---|---|---|---|---|---|
| Actions | | | | | |
| | A1 | x | x | | |
| | A2 | | | x | x |
| Conditions | | | | | |
| | C1 | F | — | F | — |
| | C2 | — | T | — | T |
| | C3 | — | — | — | T |

Although the inverted decision table is the current conceptual model of our methodology, it was first conceived of as a generalized cross-reference listing to assist in debugging the prototype software. This generalized cross-reference listing, or inverted decision table, was produced automatically from the decision table source listings.

The major problem still remaining was that the translation from the specifications to decision tables was very labor intensive, entailing considerable human interaction between programmers and systems engineers to interpret the specifications. To solve this problem, the inverted decision table was used as a model for the next iteration of the specification documents. In this next iteration, the specification is used as an input to a processor that directly generates the decision tables, as an intermediate step, and then the PL/1 code. The specification itself then acts as a cross reference and is accurately embodied in the code.

It would, of course, be possible to produce the object code directly from the formal specification without producing the intermediate form of decision table. However, there were at least three reasons for not doing this:

1. In generating software it is desirable to decompose the process into a sequence of simple transformations such that each transformation is easy to implement efficiently and correctly.
2. The intermediate form of the code represented by the decision table can be used to provide several different versions of the final program. Some of these versions may have more extensive diagnostics to be used in the debugging phase. In fact, some of the versions were so used.
3. We had at this stage of the project a working decision table processor that did a significant part of the transformation for us.

## IV. CURRENT APPROACH

The development of the prototype system began in the fall of 1980, and at that time the decision table processor was written. During the first half of 1981, the decision table processor was used to produce PL/1 code. In the late summer of 1981, the decision table inverter was written to assist in the debugging of the prototype. In planning for the

production release, we decided to use the inverted decision tables as a model for the software specification itself.

The current approach requires that a systems engineer write a set of specifications, which are then automatically translated into production software. The specification must conform to a rigid documentation format and use a propositional-calculus-type language with a limited vocabulary and a simple syntax. The specification is machine readable so that the two-stage code generation program can transform it first into decision table format and finally into PL/1 code.

Section 4.1 details the specification generation process. This includes a description of the specification structure and the language developed for the application. Also described are several software tools designed to aid in the preparation of the specifications themselves.

Section 4.2 describes the process of translating the specifications into finished code. This includes the two stages of the translation process as well as the handling of several special problems, such as the addition of program elements not expressly included in the specification.

Section 4.3 discusses the performance improvements that result from this current approach. These include programming efficiency, streamlined debugging, and vastly simplified maintenance.

### *4.1 Specification: language and preparation*

#### *4.1.1 Document structure*

The specification document contains three parts—logic modules, data structures, and function definitions. A logic module defines a set of conditions required to generate a single STI task or WU element. There are about 650 of these modules, which constitute over 90 percent of the total specification document. The data structures define the vocabulary of the specification language. There are five parts to the data structure section, corresponding to the five data-entry formats. The function definition sections define 20 special functions referenced in the logic modules. These functions perform numerical computations or evaluate logical conditions that either cannot be handled by the logic modules or that are common to many logic modules.

Presently our code generation tools work only on the logic modules. The data structures are used to manually produce the PL/1 data declarations. The 20 functions are manually coded.

*4.1.1.1 Logic modules.* Each logic module comprises six parts (see Fig. 4, which defines the conditions for STI task `4P13`).

1. Title—The first line of the module is the title. It indicates whether the module applies to an STI task or to a WU element and identifies the specific task or element involved. Thus if the conditions

```
STI   4P13
-------------------------------------------------------------
Description:
    Place cable in a trench.
-------------------------------------------------------------

Level:   job step
Factor:  matqty

Conditions:

         masktype  =  epw   &
         <like_cable>  =  `true`  &
         {  plaatn  =  ctrb    |
            plaatn  =  ctrm    |
            plaatn  =  ctrh    |
            plaatn  =  ctr     |
           {  wrkenv  =  b   &
              plaatn  =  stan   &
              pacclass  =  c   &
              OPF(bu_cbl_plc_mth,arg1)  ~=  plow
           }
         }



Notes:

   1.  No work environment check against actions ctrm,
       ctrb, ctrh, ctr, since these apply
       unambiguously to buried plant.

   2.  Actions bore and push are not allowed here
       (i.e., to generate cable placement), since they imply
       only that structure work is to be done.  If cable is
       placed in conjunction with this other work, then the
       EPW mask should be used to enter multiple actions.
```

Fig. 4—Logic module for STI 4P13.

specified in the remainder of the module are satisfied, then some quantity of this task is required to work the job step being analyzed.

2. Description—The second section is a word description of the STI task or WU element denoted by the title. In cases where more than one logic module applies to a given task, the description differentiates among the cases. The purpose of the word description is simply to aid persons reading the specification.

3. Level—The third section is a one-line entry that specifies the operating level of the logic module. The set of possible levels that may apply are defined by the data structures document and correspond to the hierarchical layers of data defined for each of the five entry forms. For example, for a placing step entry form, the possible levels are job step, supplementary placing action, and plant account code.

The level factor controls the processing of the logic module. The conditions in the module are evaluated once for each appearance in a job step of the data denoted by the level parameter. Thus if the level is job step, the module is evaluated once for a step. In the placing step example cited above (see also Fig. 4), the level is supplementary placing action, so the module is evaluated once for each action entered for the

step. (From 0 to 20 of these entries are allowed for a single placing step.)

The reason for varying levels is that different tasks apply at each of these levels. For example, since the task of setting up a work site applies to a job step as a whole, the logic module for a site set-up task is evaluated only once (job-step level) per step. On the other hand, a task associated with a placing action (e.g., STI `4P13` as shown in Fig. 4) must be evaluated for each action entered for a step in order to determine how often it applies.

4. Factor—The fourth section is another one-line entry. It specifies the quantity of the associated task to be generated if the conditions of the module are satisfied. This value must be numerical. It may be a fixed number (a constant), a variable name corresponding to an input-data item, or the result of a function computation. In the example in Fig. 4, the factor is the quantity associated with the placing action and represents the length in feet of the trench to be dug.

5. Conditions—The fifth section is the main body of the logic module. It defines the logical conditions that must be satisfied in order to generate the associated task. The conditions are written in a propositional-calculus-language form developed specifically for this application (see Section 4.1.1.2 for a detailed description).

The conditions specify the various combinations of data values that indicate the need for a given STI task or WU element. They may include references to data entered by the user, parametric data derived from these entries, the output of functions applied to these entries, or fixed parametric data extracted from tables that define the standard operating methods used by the local construction organization.

6. Notes—The last section contains an optional set of notes. Notes are provided to explain certain aspects of the conditions that may not be obvious to the reader. They may also be used to further differentiate a given module from others that apply to the same STI or WU.*

***4.1.1.2 Data structures.*** Each data structure lists the set of data items used in the logic modules and functions associated with a particular data-entry form. Figure 5 illustrates the data structure for the placing-step entry form.

Note that the data items are partitioned by operating level (see Section 4.1.1.1). For placing steps there are three such levels. The job-step level contains all those data items that apply universally to the step (e.g., only one material description—`matdsc`—is entered for a placing step). The second level is supplementary placing action. Two

---

* Typical reasons for specifying multiple logic modules for a given task are that the task can be generated from more than one input form, at more than one data level, or with more than one quantity factor.

```
Structure:  EPW
------------------------------------------------------------
Description:
    Data structure for EPW (placing work) job steps.
------------------------------------------------------------

Layers:  step[1] supp_plc_action[2] plt_acct_code[2]

Contents:


         STEP

         jobstp#    --  job step number
         rskeystp   --  RS key step flag
         rsgrpid    --  RS group i.d.
         masktype   --  mask type
         pacclass   --  stppac(1) class
         wrkenv     --  work environment
         matdsc     --  material description
         matclass   --  material class
         mattype    --  material type
         matsize    --  material size
         plaby      --  placed by indicator
         factstub   --  factory stub flag
         matqty     --  material quantity
         plaatn     --  placing action
         rdsd       --  road side flag
         hivltpro   --  high voltage protection flag
         #suppatn   --  number of supplementary placing actions
         #stppac    --  number of plant account codes


         SUPPLEMENTARY PLACING ACTION

         suppatn    --  supplementary placing action
         atnqty     --  supplementary placing action quantity


         PLANT ACCOUNT CODE

         stppac     --  plant account code
         pacqty     --  plant account code quantity
         paccls     --  plant account code class
         meas       --  measured account flag
         plttype    --  plant type



Notes:

  1.  Note that the SUPPLEMENTARY PLACING ACTION & PLANT ACCOUNT CODE
      portions of the data structure are parallel.  They each relate
      directly to the STEP level.
```

Fig. 5—Data structure for placing data-entry forms.

data items are associated with each supplementary placing action in a step. Up to 20 pairs of such data may be entered for a placing step. The third level is plant account code. Two data items are also associated with each plant account code and up to three pairs of such data are permitted per step. Note that the second and third levels in this case are parallel, in that each homes on the job-step level. For other types of steps, the data-structure hierarchy may be three levels deep.

The location of a given data item in the data structure denotes its

availability to the logic modules. The general rule is that a data item may be used in a logic module or function for which the specified level is the same as or lower than the level at which the data item is defined. For example `matdsc` may be used in any placing-step logic module or function, since it is defined at the highest level of the data structure. However, `suppatn` may only be used in those logic modules or functions that apply to the supplementary-placing-action level.

The data items listed in the data structures define most of the variables used by the specification language. These data items may be entered directly by the user via an entry form (e.g., `matdsc`), or they may be derived from an entered value (e.g., `matclass` is a parametric data element associated with `matdsc` and extracted from a table of material descriptions). Other than items in the data structures, the only data variables allowed in the logic modules or functions are function references (e.g., ⟨`like_cable`⟩ in Fig. 4) or references to the Operations Profile Table, which defines the standard operating methods used by each local construction organization (e.g., `OPF(bu_cbl_plc_mth)` in Fig. 4 defines the standard method for placing buried cable).

***4.1.1.3 Function definitions.*** A function is used either to perform numerical computations, which cannot be handled by the propositional calculus language, or to evaluate complicated logical conditions that appear in numerous logic modules or functions. The function definitions are structured much like the logic modules. Each has seven parts (see Fig. 6, which defines the logical function ⟨`like_cable`⟩ and Fig. 7, which defines the numerical function ⟨`#reg_pr_trans_sp`⟩).

1. Title—Name of the function.
2. Description—Word description of the function.

```
Function:  <like_cable>
------------------------------------------------------------
Description:
    Determine whether material item for step is cable
    or some item similar to cable (eg, ground wire,
    air pipe, innerduct, or lightguide).
    Used on EPW steps and with RS logic.
------------------------------------------------------------

Level:     job step
Step Type: epw
Data Type: logical

Conditions:

        matclass  =  cable     |
        mattype   =  ground_wire    |
        mattype   =  electrolys_wire     |
        mattype   =  air_pipe    |
        {  matclass  =  fiber_op_eqp     &
          {  mattype  =  innerduct    |
             mattype  =  lightguide
          }
```

Fig. 6—Definition for logical function ⟨`like_cable`⟩.

```
Function:  <#reg_pr_trans_sp>
------------------------------------------------------------
Description:
    Calculates the total number of regular pairs transferred for
    an SP step.
------------------------------------------------------------

Level:     job step
Step Type: sp
Data Type: numeric

Algorithm:

         total = 0 ;

         for ( i = 1; i <= #cblsht; i++ )

               total = total + #trspr ;

         return ( total - #spcpr ) ;
```

Fig. 7—Definition for numeric function ⟨ #reg_pr_trans_sp⟩.

3. Level—Operating level of the function. It is basically the same as defined for the logic modules (see Section 4.1.1.1). There is one slight difference, however: In numerical functions it is possible to selectively operate at levels lower than the one specified for the function as a whole by means of an iteration segment. These segments are denoted as *for* loops in the function algorithm.

4. Step type—Denotes the type(s) of data-entry form(s) to which the function applies.

5. Data type—Denotes whether the function computes and returns a numerical value, or evaluates a set of logical conditions and returns a true or false value.

6. Conditions/algorithm—The main body of the function. If the function is logical, this section contains the conditions that must be satisfied in order to return a value of true (if the conditions are not satisfied, false is returned). The conditions are written in the same propositional-calculus-language form used in the logic modules (see Section 4.1.1.1). If the function is numerical, this section contains the algorithm used to compute the numerical value. The algorithm is written in a predicate calculus form and includes iteration capabilities (patterned after the C programming language).

7. Notes—An optional set of notes to provide an expanded explanation of the function to the reader.

### *4.1.2 Specification language*

As noted previously, the conditions section of a logic module or function is written in a propositional-calculus-language form. This section describes some of the grammatical rules of that language, incuding the syntax and vocabulary.

***4.1.2.1 Syntax.*** The primary syntactic rules are:

1. A condition statement comprises one or more logic expressions

```
<expression>    ::= <expr1> | <expr1> or <expression>
<expr1>         ::= <expr2> | <expr2> & <expr1>
<expr2>         ::= <variable name><logical operator><data value> |
                    { <expression> }
<variable name>     ::=     any variable listed in the data structure |
                    <function> | <opf ref>
<opf ref> ::=  OPF(<opf op name><opf arg position>)
<opf op name>   ::= any operation listed in the OPF table
<opf arg pos>   ::= arg1 | arg2 | arg3
<function>      ::=     < <function name> >
<function name>     ::=     any function listed in the function definitions
<logical operator> ::= <lexical op> | <numerical op>
<lexical op>    ::= = | ~=
<numerical op>  ::= = | ~= | < | <= | > | >=
<data value>    ::=     <variable name> | <constant>
<constant>      ::= <number> | <string> | `true` | `false`
<number>   ::=  any integer or decimal number
<string>   ::=  any character string
```

Fig. 8—BNF syntax of specification.

joined by the connectors "and" (&) or "or" (|), and possibly grouped by curly brackets, { }.

2. A logic expression is a comparison between a data variable and a data value of the form

⟨variable name⟩ ⟨logical operator⟩ ⟨data value⟩

3. Permissible logical operators include = and ~=, which may be used with any variable, regardless of type. The other operators, ⟨, ⟨=, ⟩, and ⟩=, may be used only with numerical variables.

4. By convention, each logic expression is stated on a separate line.

5. By convention, curly bracket groups are indented to show the level of imbedding. Also, the closing curly bracket is positioned on a separate line directly below the opening bracket in a given group.

6. By convention, connectors (&, *or*)* are placed at the end of the last line containing the first of the two expressions being joined.

Figure 8 formally specifies the syntactic rules of the language. The Backus-Naur Form (BNF) definition given here leaves out the format of the specification that was incorporated as essentially a syntactic component. It would be possible, by extending the BNF notation to include horizontal and vertical positioning symbols, to include the format as part of the grammar. For example, if CR is used as the symbol for a carriage return, then the second part of the first definition would read ⟨expr 1⟩ or CR ⟨expression⟩. A full discussion of the considerations of including formats as part of the grammar would be tangential to the main concerns of this paper.

***4.1.2.2 Vocabulary.*** Other than the connectors and logic operators noted in the previous section, the specification language vocabulary is limited to two classes of terms: variables and constants.

---

* In the actual object language, "or" is denoted by |. Here we have reserved the use of | for the metalanguage disjunction.

The variables have already been defined (see Sections 4.1.1.1 and 4.1.1.2). They include the items listed in the data structures, the functions, and the references to the Operations Profile Table. Variables may appear on either side of the logical operator in a logic expression, though they usually appear on the left.

Constants may be numbers or character strings and may appear only on the right side of a logic expression. Common examples are

```
plaatn = plac
matsize <= 400
```

A constant must match in type and value one of the acceptable values of the variable to which it is being compared.

Two special constants are true and false. These correspond to the binary values of logical variables. They are used to make the specification more readable.

#### *4.1.3 Derivation*

Several programming tools were developed to aid in preparing the specifications. These enable the systems engineer to write a source file using severely abbreviated terminology and a very simple, loose format. The tools translate and expand the abbreviations into finished vocabulary and syntax. They also produce a finished format that adheres to all of the necessary conventions. For example, Fig. 9 is the source file corresponding to the logic module shown in Fig. 4.

Because of the magnitude of the specification, this tool provides several major benefits.

- It reduces typing significantly and enables much of the specification to be drafted directly on-line from working notes;
- It eliminates the need for word-processing support to convert the draft into a finished document—a finished specification is produced within minutes;
- It reduces typing errors;
- It guarantees that formatting is consistent and adheres to all conventions; and
- It simplifies correction and maintenance.

In addition to the translating and formatting programs, the *UNIX* operating system screen-editing tools were used to create templates of recurring logical conditions. These templates were then inserted where appropriate, usually requiring only minor editing changes or no changes at all. This eliminated the need to retype these sections. Nearly half of the specifications for placing steps were written using such templates.

```
gn GENERIC 1.0
tl SPECIFICATION FOR STI LOGIC -- BURIED PLANT PLACING WORK
id task STI 4P


hd 13
dsc
Place cable in a trench.
enddsc
lv step matqty
cn
epw &
<like_cable> = `true` &
{ pa ctrb |
pa ctrm |
pa ctrh |
pa ctr |
{ wb &
pa stan &
pc &
OPF(bu_cbl_plc_mth,arg1) ~= plow
}
}



Note
1.  No work environment check against actions  ctrm,
    ctrb, ctrh, ctr, since these apply
    unambiguously to buried plant.


2.  Actions "bore" and "push" are not allowed here
    (i.e., to generate cable placement), since they imply
    only that structure work is to be done.  If cable is
    placed in conjunction with this other work, then the
    EPW mask should be used to enter multiple actions.
endnote
```

Fig. 9—Specification source file for STI 4P13.

### *4.2 Mechanical translation to design*

The objective of a mechanical translation to design is to take the specification from a file on the *UNIX* system Programmers' Workbench and after performing the appropriate transformations, submit the PL/1 source code corresponding to the specification for compilation via a Remote Job Entry (RJE) link. Once such a mechanical translation is realized, many problems at the object level can be resolved at the metalevel where they are often easier to deal with.

The complete process is a shell procedure called `spec to pli`. `Spec_to_pli` performs the following steps in sequence:

- Retrieve files—Retrieves the specification files.
- Preprocess specifications—A buffer step to filter out any notational variations in the specifications, and ensure that the input to the rest of the code generation process is under control of the code generator.
- Split out the component files—A single specification file may specify several decision tables. These are broken out in this step.
- Alphabetize the Booleans—A housekeeping routine. To make the

object code more systematic, the logical variables corresponding to the condition codes in the decision tables are alphabetized. (There are often close to one hundred such variables.)

- Shorten the lines—The code generator may use very long lines internally in its list of logical variables, but the PL/1 environment limits the line length to 72 characters.
- Rename files—An interface to match the file names to those expected by the decision table processor.
- Assemble files into decision tables—The four component parts of the decision table, described above, are assembled here.
- Generate PL/1 from decision tables—The decision table processor itself.
- Add JCL to PL/1 code—The job control statements needed by Time-Sharing Option (TSO) are added here.
- Cleanup—Remove miscellaneous working files that have been generated in previous steps.
- Send to TSO—Submit to RJE.
- Save the decision table and the PL/1 code—These items are stored in a separate directory, where they may be inspected, or printed out.

Translation from a specification to a design requires the addition of those elements that are present in a design but are not present in a specification, or that are present in both but in a different form:

- Variable declarations, defining the types of the variables, are generally not present in a specification;
- Organization of sets of variables into a larger structure is present in a design because of programming or database considerations, but is not usually present in the specification.
- If the design uses components that are referred to in the specification, but not described there, the design must solve the problem of resolving these references by including, or linking, the appropriate code.
- Occasionally, aliasing problems arise because of the preceding item, and the mechanical translation must deal with these.

The solution to these problems is a program that constructs a structure to which the code in the decision table refers. In this structure, the organization, types, and specific names of the variables as seen by the decision table are all under our control. While it would be possible to write the code manually for the procedure to populate this mediating structure, it is rather simple to generate both the declaration of the structure to be populated and the program to fill it from a small data dictionary maintained for this purpose. This data dictionary can be a rather simple, ad hoc facility since it is designed for a rather limited purpose.

#### 4.2.1 Current status of the code generator

The current version of the code generator accepts as input the specification files, in the formatted version that is the customer copy of the specification. This is a propositional logic description of the conditions that generate the appropriate work credits—and, as noted above, is essentially an inverted decision table. The 20 specification files comprise 850 pages. The outputs of the code generators are 41 decision tables and their associated PL/1 programs, since most of the specifications produce more than a single decision table. The PL/1 code is approximately 16,500 lines of source code, the lines being relatively densely populated. Both the decision tables and the PL/1 code are stored in files, with the PL/1 being sent for compilation as well.

The code generator running time is about 0.1 second/line of system time, and 0.3 second/line of user time. In off hours the elapsed real time is about 0.6 second/line. Thus it is quite reasonable to regenerate the PL/1 code overnight if necessary.

#### 4.2.2 Productivity

The literature on programming productivity describes several high-productivity methods for generating programs. Foremost among these are two techniques: reusable code and code generators. The reuse of code is applicable when one is developing a set of programs for similar applications, and common functions can be identified among the various applications. In that case the first application in the set to be developed pioneers the code, and subsequent applications reuse it. If the subroutines cannot be reused without modification, then they should be generalized in the hope of anticipating future requirements. Thus after several iterations, a considerable library of common functions and subroutines can be developed. The foremost example of reusable code is the extensive scientific subroutine package of Fortran.

In the present application, the cases of enumerative complexity are sufficiently specialized that it does not seem likely that any other application would have need for these particular specifications. For this reason we have chosen to use the techniques of code generation here.

As far as programmer productivity, a productivity rate in excess of ten thousand lines of tested code per year is readily achievable using code generation techniques. Indeed, the goal of code generation techniques is to ultimately put the programmer in the position of only developing and maintaining the code generators. In that case, applications experts would retain all of the knowledge about the application and would generate formal specifications that could be compiled directly. Indeed, the purpose of formalization is to abstract the seman-

tics of the problem so that the programmer can deal with it as a purely syntactic problem.

The code generation time required to produce all of the decision tables and associated PL/1 code for one 62-page specification was 1.5 minutes of system time—4.5 minutes of user time—in a *UNIX* system on a *VAX 11-780* computer.* The four decision tables generated in this case were 350 to 400 lines each.

Using this code generator, it was possible to generate a completely fresh set of decision tables and PL/1 code each time that the specifications were changed. In this way the specifications did indeed become the primary program description.

## V. ASSESSMENT

The methodology and tools described in this paper were effective in producing a complex product on schedule, with the assurance that the program corresponded to the specification. Indeed, most of the specification files were revised, some of them extensively, within the final week prior to delivery. When such revisions were necessitated because the specification did not accurately state the conditions for various work credits, the production code was regenerated from the revised specification. This has the advantage that the last-minute changes to the code are not patches, and avoids the problem of introducing new bugs when fixing old ones.

The development of the module described in this paper supports the thesis that increasingly large roles in the software-development process should be played by code generation techniques. Indeed, the major problems encountered in the process of producing the code arose because we did not have some tools that should be part of the code generation facility. Examples are

- Syntax checking—There was no syntax check of the specification. Syntax errors in the specification showed up only as compile-time or run-time errors.
- Semantic checking—No tests were run to determine the correctness of the specification, until it was translated into production code. Most of the errors that were detected in the operational tests of the software were errors that could have been found by a diagnostic tool capable of executing the specification.

The code generation process also provided severe tests of other software. The symbols and data structures that were generated exceeded the capacity of some of the available tools. Examples are

- Variable names are constructed automatically by the code gener-

* Trademark of Digital Equipment Corporation.

ator and are semantically significant as an aid in diagnosis. Thus the statement in the specification:

```
pacclass = 'c'
```

generates the logical identifier:

```
pacclass_eq_c.
```

In some cases, this translation yields symbols that exceed the 31-character PL/1 limit on identifiers. (In those cases, provision was made in the generator to shorten the names. It was necessary in those cases to take special precaution to avoid name conflicts where two distinct names might have the same 31-letter prefix.)

- Lines constructed by the code generator often contained the entire specification of the conditions for a work task. In unabbreviated form these lines might be as long as 1,500 characters. Since `awk` is one of the primary tools used by the code generator and has a line length limit just under 512, this caused a problem. The solution was to use data compression and expansion at different stages in the generation process. (The primary tool used to effect the data compression and expansion is `sed`—the *UNIX* system stream editor.)

In summary, the development of formal specifications and their automatic processing by computer yielded significant dividends in the JMOS project. Considering that the technology of application software generation is still quite new, we expect that much greater dividends can be obtained as we learn how to manage and organize software-development projects to more fully exploit this technology. In addition, new and improved tools and techniques will arise from aspects of the code generation process that are different from the manually produced code.

Although we attempted to quantify the productivity gains that result from the processes that we have described, we feel that the technology is too new and there are too many variables to support particular claims. Much of the time of the authors was spent in developing tools and learning how to use them. Still a significant improvement was obtained in both the quality and quantity of the specifications and the delivered code, even when no allowances are made for the time spent developing the tools and learning how to use them.

## REFERENCES


1. J. Goodenough and S. L. Gerhardt, "Toward a Theory of Test Data Selection," IEEE Trans. Software Eng., *SE-1,* No. 2 (June 1975), pp. 156–73.
2. R. Davis and J. King, "An Overview of Production Systems," *Machine Intelligence, Vol 8,* E. W. Elcock and D. Michie, Eds., New York: Wiley, 1977, pp. 300–32.
3. M. D. Rychener, "Production Systems as a Programming Language for Artificial

Intelligence Applications," doctoral dissertation, Carnegie-Mellon University, Pittsburgh, December 1976.
4. *Computer—Special Issue on Application Oriented Specifications, 15,* No. 5 (May 1982).
5. C. Jones, *Programming Productivity: Issues for the Eighties*, New York: IEEE Computer Society, 1981.
6. A. V. Aho and J. D. Ullman, *Principles of Compiler Design*, Reading, MA: Addison-Wesley, 1977.
7. A. F. Cardenas, "Technology for Automatic Generation of Application Programs—A Pragmatic View," MIS Quarterly, *1*, No. 1 (September 1977), pp. 49–72.
8. E. Levinson, L. S. Levy, and J. B. Salisbury, "CARL—Experience of an Application Using Clusters," Proc. NCC, Chicago, 1981, pp. 241–8.
9. J. C. Zolnowski and P. D. Ting, "An Insider's Survey on Software Development," Proc. Sixth Int. Conf. Software Eng., Tokyo, 1982, pp. 178–87.
10. J. G. Rice, "Build Program Techniques: Objectives, Processes, and Processes," Interactive Systems, 1975 European Computing Congress, Online, Uxbridge, UK, September 26, 1975.
11. A. F. Cardenas and W. P. Grafton, "Challenges and Requirements for New Application Generators," Proc. NCC, Houston, 1982, pp. 341–9.
12. J. M. Grochow, "Application Generators: An Introduction," Proc. NCC, Houston, 1982, pp. 389–92.
13. R. L. Roth, "Program Generators and Their Effect on Programmer Productivity," Proc. NCC, Houston, 1982, pp. 351–8.
14. A. M. Goodman, "Application Generators at IBM," Proc. NCC, Houston, 1982, pp. 359–62.

## AUTHORS

**Leon S. Levy**, B.A. (Physics), 1952, Yeshiva College; S.M. (Applied Science), 1955, Harvard; M. E. (Applied Mathematics), 1958, Harvard; Ph.D. (Computer and Information Science), 1970, University of Pennsylvania; AT&T Bell Laboratories, 1979—. Mr. Levy's work at Bell Laboratories has been in the field of application code generation. In 1983 he received a Distinguished Staff Award for this work. He was on leave for the academic year 1983–84 as a visiting Full Professor of Computer Science at Ben Gurion University in Beer Sheba, Israel. In 1984 he returned to AT&T Bell Laboratories. Member, ACM, ACL, and an affiliate member, IEEE Computer Group.

**H. Theodore Stump**, B.A. (Mathematics), 1967, Franklin and Marshall College; M.S. (Applied Mathematics), 1970, Stevens Institute of Technology; Bell Laboratories, 1967–1976; New Jersey Bell Telephone, 1976–1979; Bell Laboratories, 1979–1983. Present affiliation Bell Communications Research, Inc. When Mr. Stump returned to Bell Laboratories in 1979, he worked as a systems engineer in the area of construction operations systems. In 1983 he was appointed Supervisor of a group responsible for systems engineering of new loop technology and services maintenance. In 1984 he joined Bell Communications Research as District Manager for Construction Systems and New Technology Maintenance. Member, IEEE.

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Proposed Specification of BX.25 Link Layer Protocol

By R. P. KURSHAN*

(Manuscript received October 3, 1983)

The BX.25 link-layer protocol is a standardized procedure for establishing and maintaining a connection across a data link. Virtually all such standardized communication protocols worldwide are defined through English-language specifications. While the English-language BX.25 specification was developed with exceptional care and in fact represents an improvement over the international standard X.25, it is argued here that it, or any English-language specification, can be expected to fall short of the implicit objective: to define implementations that will be mutually compatible and satisfy given performance criteria. An alternative formal specification is presented. Among its attributes are that it is by its nature mathematically precise; it provides a medium for formal structured development and formal proof of performance of a task (validation) independent of any implementations; and the formal specification may be implemented into software or hardware in an automated fashion, reducing the need for laborious and repetitive implementations and virtually eliminating the need for "certification" of implementations. The specification given here can also be used to resolve issues of vagueness, ambiguity, and contradiction as they arise in the English-language BX.25 specification standard.

## I. INTRODUCTION

The BX.25 link-layer (level 2) protocol is for data packet interchange between DTE (Data Terminal Equipment) customer equipment and DCE (Data Circuit-Terminating Equipment) network equip-

* AT&T Bell Laboratories.

ment, or between two DTEs, over a single data link (a general reference for the terminology used here is Ref. 1). The protocol is designed with the expressed intention to be compatible with the International Standards Organization (ISO) Standard High-level Data-Link Control (HDLC) and International Telegraph and Telephone Consultative Committee (CCITT) Recommendation X.25 Link Access Procedure B (LAPB).

### *1.1 Function of level 2*

The functional purpose of level 2 is to provide end-to-end transmission of "frames" (packets) with the capabilities of error and flow control. Error control is provided through error detection and error correction. Error detection is accomplished by means of a Cyclic Redundancy Check (CRC) for detection of erroneous frames, and by level 2 issued sequence numbers for detection of lost frames. Error correction is accomplished by retransmission of lost or erroneous frames. Flow control is provided through interstation signaling, which interrupts the generation of frames at the remote end. The protocol is defined for use on a synchronous, full-duplex link.

### *1.2 OSN specification of level 2*

This paper is based upon the following document, hereinafter referred to as OSN: "Link-Layer Logical Interface," Section 2, *Operations System Network Protocol Specification: BX.*25, Issue 2, Part II, published by Bell Laboratories in March 1980. (Since the writing of this paper, a third issue has appeared; however, the Part II, Section 2 of concern here remains substantially the same in the new issue.)

#### *1.2.1 Purpose*

The purpose of OSN, as described elsewhere in the BX.25 document, is to provide a standard that will serve two goals. The first is to assure compatibility of two stations adhering to the standard. The second is to assure the performance of any implementation adhering to the standard.

#### *1.2.2 Means*

The means by which OSN sought to fulfill the above purpose is a very carefully formulated English-language description with numbered paragraphs and associated figures and tables (see OSN) that covers "every aspect" of operation of the protocol. The description is meant to describe the functions of the protocol, independent of any particular implementation.

### *1.2.3 Critique*

The BX.25 link-layer protocol is a standardized procedure for establishing and maintaining a connection across a data link. The protocol is defined through an English-language specification, the way virtually all communication protocols are defined worldwide. In fact, the BX.25 link-layer protocol is a refinement of X.25 LAPB, a protocol adopted by the international telecommunications standards organization, CCITT. BX.25 is compatible with X.25, and corrects certain problems associated with X.25.

Why write a protocol specification? One answer might be to provide implementors with a conceptual sense of the facility to be implemented. This is a reasonable use for such a document, and many of the carefully worded English-language specifications previously adopted as standards, including BX.25, serve fairly well in this capacity.

However, there is no guarantee that an implementation whose design is guided by such a specification will be compatible with any other such implementation. Nor is there a guarantee that any such implementation will maintain any level of performance whatsoever. When stated this way, there are few who would disagree. Nonetheless, there is an implicit inference held by many people that if a communication protocol implementation "adheres" to a specification, then it will be compatible with all other "adhering" implementations and, furthermore, will maintain a certain level of performance. An overriding difficulty with this inference, as we shall see, is that in practice there is virtually never a reasonable criterion for "adherence" of an implementation to an English-language specification.

At the root of the difficulty is the semantic meaning of "specification." Actually, specification would seem to entail a three-fold purpose, which goes far beyond the (certainly useful and viable) role of presenting a conceptual sense of a facility. According to this expanded purpose, a specification first of all should define (logically) a class of implementations (namely, those which adhere to it). Within the context of the specification, it should be possible to state a task that each implementation is intended to perform (e.g., maintain some level of performance). The specification should be such that it is possible, at least in theory, to determine whether it is logically consistent with the stated task (e.g., if part of the required performance is to detect all sequence number errors, it should be possible to determine by theoretical means whether every adhering implementation necessarily will detect all sequence number errors, without actually testing each of what is most likely an infinite number of such possible implementations). If it is not presumed that a task can be stated, or that the question of whether a task always will be performed is logically

meaningful, then adherence to a specification loses all practical significance.

The question is whether an English-language specification can satisfy these three points. The claim here is that it cannot. The basic problem with English-language specifications is that they generally lack any concrete form or structure (such as one does find in "formal" specifications). This amorphous quality of English-language specifications leads to a generality which engenders vagueness and ambiguity. Furthermore, a lack of structure fosters the appearance of inconsistencies.

For example, the BX.25 specification is the result of careful thinking by many talented individuals. It is well organized and formulated, probably as much so as any English language specification can be. It represents an improvement over the X.25 specification, which itself was the result of careful thinking by other individuals at the international level; BX.25 itself has undergone two major revisions. All this effort notwithstanding, BX.25 is nonetheless vague (e.g., the conditions under which a poll may be sent are never made explicit), ambiguous (e.g., precedence of the rejection of an out-of-range N(R) described in OSN 2.4.7, over the packet acceptance described in OSN 2.4.5.2, while probably intended, is never stated), and contradictory (e.g., OSN 2.3.4.3 provides for retransmissions of REJ [Reject] while OSN Table 2.1c uses RR [Receiver Ready] consistently in its place). Given the quality effort that went into this specification, it might seem unreasonable to place the blame for these deficiencies upon the individuals who created the specification; an alternative culprit is the specification medium: the nonformal, unstructured English language. This view is reinforced by the (now widely accepted) observation that such difficulties are not the exception but the rule in English-language specifications of communicative protocols.

Definitions that are vague and ambiguous have limited use. It is not surprising to hear reports that different implementations of OSN (and virtually all other standardized communication protocols, for that matter) have been incompatible and of varying performance, even though this was the very situation OSN was intended to prevent.

Let us suppose for the sake of argument that a protocol specification such as OSN were free of imprecision. Where would that leave it with respect to its purpose? How would one determine whether a particular implementation adheres to the specification? Since the specification is nonformal, a formally defined adherence criterion would have no meaning beyond its informal interpretation. Possibly the best adherence criterion would be that several experts examine the implementation and declare that it adheres. The issue of stability of such a criterion aside, however, the consensus of experts is a patently im-

practical criterion, as many implementations involve optimized code of an arcane nature understood only by its designer. More likely, one would skirt the adherence criterion altogether, relying instead upon testing the implementation. What about testing (which is commonly called "certification"?[2] It is a notoriously difficult problem to design a test for performance of a task. At best, such a test can only say with a certain level of confidence, assuming the implementation has certain properties (which, in fact, we can never know for sure that it does have), that it performs the task. In practice, however, it is virtually never the case that a testing procedure is sufficiently well analyzed that one can assign to the result a qualitatively defined degree of confidence. Thus, the test itself can only suggest that an implementation is okay. Moreover, such a test certainly can say little or nothing about the overall integrity of the specification, since the relationship between the specification and an implementation is not quantized. (As a case in point, there have been satisfactory implementations of BX.25 level 2, the above-mentioned deficiencies in the specification notwithstanding.)

There are, in fact, models that are used for protocol validation, such as Refs. 3 and 4 (more about them later). However, to use such a model, a given specification such as OSN must be translated into the context of the model. As there is no unique way to do this, one might end up with a validated translation of OSN, but certainly not with a validated OSN (other translations may not be valid).

Thus, the status of the English language as a specification medium is this: it does not lend itself to precise, unambiguous, or internally consistent specifications; and there is no meaningful way either to test (abstractly) the properties of a specification or to determine whether a particular implementation adheres to a specification. It does provide a conceptual sense of a facility. However, if this is the only requirement placed on a specification, the specification could be considerably simplified. For example, the concept of the link layer of X.25 is presented in Ref. 1 in a mere four pages (as compared to 27 pages for OSN). When one speaks privately to implementors who have implemented a protocol from an English-language specification, one often learns either they did not concern themselves with the precise details of the specification, or if they did, then these details have led them astray.

#### *1.2.4 This paper*

This paper presents an example of an alternative approach to protocol specification. The approach is to present a specification in a formal, mathematically based context that, by its very nature, renders the specification free of vagueness, ambiguity, and inconsistency.

Furthermore, this formal context not only provides a medium for precise specifications, but also provides a medium for a formal, structured development of the protocol and for statement and formal proof of the performance of a task (independent of any implementations). The formal specification may be stored on-line as a database in a computer, and tested and altered quite easily, prior to any implementations. Finally, the specification medium is such that a protocol specification may be implemented into software or hardware in an automated fashion.

This paper presents a formal specification of BX.25 level 2, presented through such a formal specification medium (called the *selection/resolution model for concurrent processes*). The specification is intended to illustrate that such a formal specification can be quite readable (the implementors with whom I have interacted have been able to follow such a specification after about an hour of coaching). A production quality, formal specification of the protocol should be preceded by an informal introduction that presents the concept of the protocol (e.g., the above-mentioned four pages in Ref. 1). It is proposed that such a formal specification with a suitable introduction replace currently accepted English-language specifications. Not only would this permit true validation (formal proof of performance of task); it could also eliminate the need for laborious and repetitive implementations of the same protocol, through application of automated implementation. This would save time and effort, virtually eliminate the need for certification of implementations, and reduce the problem of interimplementation compatibility to the level of hardware interfaces.

The specification given here can also be used to resolve issues of vagueness, ambiguity, and contradiction as they arise in OSN. Deviations from OSN occur only for the purpose of resolving inconsistencies as they appear in that specification; they are indicated by a comment in the right margin of the formal specification. In every case that the formal specification clarifies an ambiguity or resolves a vague issue in OSN, this clarification has been cleared with at least one expert on BX.25 level 2 (such clarifications were too numerous to indicate them in the formal specification).

### *1.3 Other possible level 2 specifications*

There is no dearth of models for specification and analysis of concurrent processes [see the Special Issue on Computer Networks and Protocols of IEEE Trans. Commun., *28,* No. 4 (1980)]. However, there is something unsatisfying about each such model I have examined. Without trying in any way to give a critique of the literature, I will voice my complaints about two such models. The complaints (not the models) may be considered to be representative.

Bochmann and Chung use a Petri net model for a synchronous environment in which transitions are determined by enabling predicates "specified in terms of a high-level programming language such as Pascal."[3] However, this makes no sense in an asynchronous environment as the lower-level interactions of synchronization, interruption, message exchange, and other "critical section" problems are hidden from view by the programming language. This aside, they describe no algorithm for combining several processes, a necessity for any analysis.

Zafiropulo et al. use a finite state model based upon message exchange.[4] While this is (theoretically) capable of specifying processes in asynchronous environments and an algorithm for combining several processes is described, transitions are based simply upon the receipt or transmission of a message. There is no provision for complex interdependencies among processes such as exist in BX.25.

Other specification formats such as Ref. 5 provide actual software to establish a database containing a well-defined, formal specification. However, no tools exist to analyze the resulting protocol.

### *1.4 The selection/resolution model*

The format used here to specify level 2 is based upon a general model for concurrent processes.[6,7] It is arithmetic in nature and has the property that the collective specification and joint behavior of several processes is computed by a fixed algorithm from the specifications of the component processes alone. Processes are described in terms of states and enabling predicates. The states need not be listed exhaustively, but may be represented by one or more parameters. For analysis, a task is formally defined. To facilitate validation (proof that a given task is performed) reduction algorithms are available that substitute for a given system a simpler system with the (provably) same performance. Such reductions are necessary to combat intractability produced by "state explosion."

#### *1.4.1 Methodological attributes*

The approach of the selection/resolution model has several methodologically desirable attributes. First, all algorithms may be applied mechanically without understanding the meaning of the protocol specification or task. Second, the procedure for defining the formal specification requires a complete systematic detailing of process behavior, reducing the opportunity, which exists in a less organized approach, to overlook events. The significance of this is not negligible. Most approaches to specification suffer not so much through incorrect assertions as through what is overlooked. While specification mistakes and oversights are still possible in the model described here, they are

easily caught (see Section 1.4.2). The given specification is guaranteed to be logically complete in the sense that there can be no unexpected event.

Not only does the specification provide a vehicle for analysis validation, it provides the basis for implementation as a structured program. This is helpful not only to the implementor, whose programming job becomes straightforward if not trivial through automated implementation, it is essential to assure the carry-over of compatibility and performance to each implementation.

### *1.4.2 Quick review of the model*

A system is decomposed into many small interacting automaton-like processes. Suppose process **A** is in an OFF state and process **B** is in its READY state. The process **C** comprised of the two processes **A** and **B** taken together is then in a joint state (OFF, READY). Process **A**, let us suppose, has an ON state and a labelled directed edge from OFF to ON. The label, say, corresponds to the assertion, "**A** is in its OFF state and **B** is in its READY state." This label encodes the condition under which the transition from OFF to ON may occur. It is denoted thus:

$$(\mathbf{A}{:}\mathrm{OFF}) \cdot (\mathbf{B}{:}\mathrm{READY}) .$$

Suppose **B** has a labelled directed edge from READY to its state WILLING with the label

$$(\mathbf{B}{:}\mathrm{READY}) \cdot (\mathbf{A}{:}\mathrm{ON}) .$$

| **A** | **B** |
|---|---|
| OFF | READY |
| $\downarrow(\mathbf{A}{:}\mathrm{OFF}) \cdot (\mathbf{B}{:}\mathrm{READY})$ | $\downarrow(\mathbf{B}{:}\mathrm{READY}) \cdot (\mathbf{A}{:}\mathrm{ON})$ |
| ON | WILLING |

Then the joint process **C** has an "edge" from its state (OFF, READY) to its state (ON, WILLING) with the label

$$(\mathbf{A}{:}\mathrm{OFF}) \cdot (\mathbf{B}{:}\mathrm{READY}) \cdot (\mathbf{B}{:}\mathrm{READY}) \cdot (\mathbf{A}{:}\mathrm{ON})$$

(The Boolean product of the label of **A** and the label of **B**).

When interpreted as a Boolean expression, this has the logical value 0 (always false) since (**A**:OFF) and (**A**:ON) can never be true simultaneously. Thus, in fact, **C** has no edge between (OFF:READY) and (ON:WILLING). Had **B**'s label been rather

$$(\mathbf{B}{:}\mathrm{READY}) \cdot (\mathbf{D}{:}?)$$

then **C**'s edge would have had the (reduced) label

$$(\mathbf{A}{:}\mathrm{ON}) \cdot (\mathbf{B}{:}\mathrm{READY}) \cdot (\mathbf{D}{:}?) .$$

In this example, each process selects (broadcasts to the other processes) the name of its current state. This is a special case of the more general presentation but is typical of the processes defined for level 2.[6]

The joint behavior of all the processes that comprise level 2 taken together is modelled by taking all such possible products as illustrated above. This procedure is curtailed (so as to avoid the need to consider the labels on some $10^{12}$ possible edges) through formal reductions of the system.[6]

### *1.5 How the structural organization of the protocol was established*

The division of an environment (in this case, level 2) into component processes (see Fig. 1) is part discipline and part art. The general approach involves several passes over a founding document such as OSN, a set of notes or evolving ideas. In the first pass, the so-called explicit processes are identified. These are objects, parameters, routines, and the like that are explicitly named, such as timers, counters, messages, and channels. Some are more obviously translated into processes than others. For example, a counter is a conceptually simple process that advances from state $N$ to state $N + 1$ each time the thing it is counting arrives. A message is produced by a sending process whose states correspond to the message it sends. (The Control Primitive process **CP** sends the primitive REJ when it is in state REJ.) A channel process (such as level 1) is represented as a buffer process which stores the message $M$ it is transmitting while in its state $M$. Delay is modelled by the length of time the channel process remains in the state $M$. In this pass, each such explicit process is named and its states are defined (to the extent possible).

Once the explicit processes have been identified, another pass is made over the founding document and the list of explicit processes and associated states. In this pass, implicit processes are identified. These are mainly processes which keep track of what is going on (and hence are dubbed place-keeper processes). For example, if the founding document says, "Transmit SABM and wait for acknowledgement," a place-keeper process must be defined that will (say) start in state 1 and then move to state 2 when SABM (Set Asynchronous Balanced Mode) is sent. This enables the protocol to determine if an appropriate reception is in fact an acknowledgement and not simply out of the blue.

Having identified all the component processes (explicit and implicit) and their respective states, a third pass is made, this time over the processes, to define the selections of each process. A selection is a signal that each process broadcasts from a given state. In level 2 almost every process simply selects the name of the state it is in. In

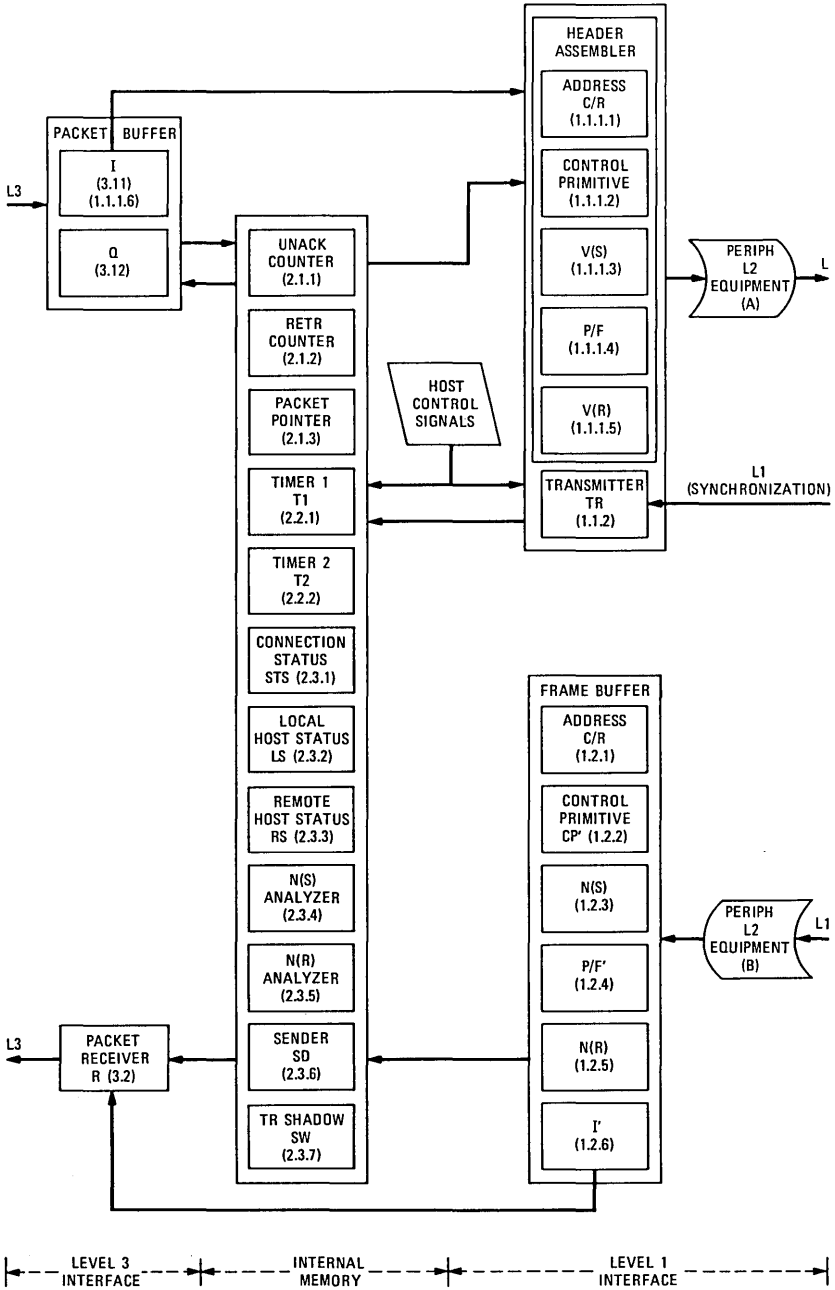



Fig. 1—BX.25 link-layer (level 2) protocol structural organization local end.

other words, it broadcasts the name of its state. (An exception is the transmission process, **TR**.) Finally, a fourth pass is made in which the edge labels of each process are defined in terms of the selections of the various processes. These edge labels are Boolean functions in the selections of various processes that describe the conditions under which the associated transitions occur.

### *1.6 Sources of errors*

Having completed the specification of all the component processes comprising a protocol, one begins a refining procedure to eliminate errors. This is best facilitated when the specification resides in an on-line database. Four types of errors are possible:

1. Errors in the founding document (OSN)
2. Errors of inference from the founding document
3. Errors in transcription (typographical errors)
4. Errors in interpretation of implementation considerations.

The first three types of errors will be caught (insofar as they hinder the formally stated task of the protocol) during the validation phase, during which one applies algorithms to the specification database to prove that the given task is satisfied (or not satisfied). If it is determined that the task is violated, the source of the difficulty is tracked down, corrected, and the validation procedure repeated. This may be continued until a satisfactory specification is found. (Alternatively, there are algorithms for synthesizing a satisfactory specification.[6] However, in the present case, the result would be quite different from OSN.) Aside from verification that the task is performed, it is often useful to examine possible histories of parts of the protocol to determine whether there are any behavior abnormalities (whose exclusion was omitted from the scope of the task).

The fourth type of error is caught upon implementation by the programmer or chip designer who notices that it is impractical or impossible to be faithful to the given specification. This is rectified by altering the specification accordingly and repeating the validation procedure. (An error of the fourth type would not arise if the specification were derived from an actual implementation rather than a conceptual design.)

### *1.7 Synchronization*

The local end (level 2) and remote end are each assumed to be controlled by separate clocks driven by the respective clock of the associated incoming level 1 (see Fig. 2). Thus each level 2 is synchronized to its incoming level 1. The two ends are assumed to synchronize at the interface of level 2 and the outgoing level 1, but are otherwise asynchronous. (This is an inference, nowhere discussed in OSN.)

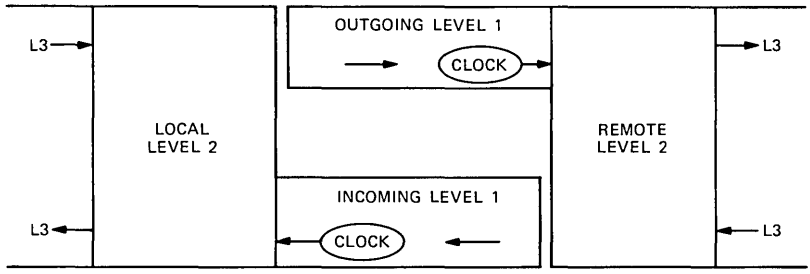



Fig. 2—BX.25 link-layer (level 2) protocol structural organization end-to-end. The two asynchronous components of an end-to-end link layer.

In each synchronous end, time is divided into (not necessarily uniform) intervals. During each time interval (defined separately for each end by the centralized scheduler at each end), each process makes a selection (which in most cases is simply an announcement of its current state). Once each process has done this, and before the end of the time interval, each process resolves the selections of the other processes by moving to a new state along an edge whose label is enabled by the joint selections of the other processes. Once this is done, the time interval is ended. Since different edge labels have varying degrees of complexity, transitions will require varying lengths of time (and the associated time intervals will vary). All this is controlled by the scheduler for the corresponding synchronous end.

However, unlike the specification format,[5] the state transitions all will be relatively quick, and thus it may be desirable to implement the time intervals so all have uniform length. This is especially true if the component processes are implemented in hardware.

### *1.8 Purpose of this paper*

This paper serves to illustrate a specification of BX.25 level 2 using the selection/resolution model.[6,7] The specification was derived almost exclusively from OSN. While every attempt was made to avoid errors in inference, transcription, and interpretation, the greatest likelihood is that all three types will be found here. They, as well as errors in OSN, could be ironed out during validation, which will not be discussed further here.

## II. HIERARCHY OF PROCESS GROUPS

The processes that comprise level 2 may be grouped according to function into a hierarchy of classes. At the highest level of the hierarchy, I identify three classes:

1. Level 1 frame interface
2. Internal memory

3. Level 3 packet interface.

The level interface processes transmit or receive, and buffer signals or packets/frames between levels (i.e., packets between level 3 and level 2, and frames between level 2 and level 1). Processes in the internal memory class include counters, timers, and place-keeper processes to remember what is happening, such as "link setup in progress." The three main classes are further subdivided as follows, with process name acronyms indicated in boldface.

1. Level 1 interface
   - 1.1 Outgoing
     - 1.1.1 Header Assembler
       - 1.1.1.1 **C/R** (Address)—identifies frame as command or response.
       - 1.1.1.2 **CP** (Control Primitive)—sets one of I; RR, RNR, REJ, SABM, DISC, DM, UA, FRMR.
       - 1.1.1.3 **V(S)**—sets the "send" sequence number N(S) (generated locally) of the next (information) frame to be transmitted.
       - 1.1.1.4 **P/F**—sets the poll/final bit.
       - 1.1.1.5 **V(R)**—sets N(R), the receive sequence number (generated remotely) of the next expected information frame to be received.
     - 1.1.2 **TR** (Transmitter)—notifies outgoing level 1 and "internal memory" processes of onset of frame transmission.
   - 1.2 Incoming (frame buffer)
     - 1.2.1 **C/R′** (Incoming address buffer)
     - 1.2.2 **CP′** (Incoming control primitive buffer)
     - 1.2.3 **N(S)** (Incoming N(S) sequence number buffer)
     - 1.2.4 **P/F′** (Incoming poll/final bit buffer)
     - 1.2.5 **N(R)** (Incoming N(R) receive sequence number buffer)
     - 1.2.6 **I′** (Incoming packet number buffer)

2. Internal memory
   - 2.1 Counters
     - 2.1.1 **UNACK**—the next expected receive (acknowlegement) sequence number N(R) (sequence generated on this end) in a received (information or supervisory) frame ($0 \leq N(R) < 8$).
     - 2.1.2 **RETR**—number of (re-) transmissions $N$ of a frame subsequent to the expiration of timer **T1** ($0 \leq N \leq N2$).
     - 2.1.3 **PP** (Packet Pointer)—points to packet in packet buffer that was last transmitted to level 1.
   - 2.2 Timers
     - 2.2.1 **T1**—times unacknowledged frame, pending initiation of retransmission procedure.
     - 2.2.2 **T2**—times period of link idleness pending declaration that remote station is unresponsive or inoperative.
   - 2.3 Place-keepers
     - 2.3.1 **STS** (Connection Status—States) S1 = disconnected
       S2 = link setup
       S3 = frame rejected
       S4 = disconnect request
       S5 = information transfer

S6 = REJ primitive sent
S7 = waiting acknowledgement

2.3.2 **LS** (Local host Status)—indicates status of local end.
2.3.3 **RS** (Remote host Status)—indicates busy/not busy status of remote end.
2.3.4 **N(S)A** (N(S) Analyzer)—indicates valid/invalid status of received N(S).
2.3.5 **N(R)A** (N(R) Analyzer)—indicates valid/invalid status of received N(R).
2.3.6 **SD** (Sender)—indicates to-be-sent/already-sent status of header assembler.
2.3.7 **SW**(TR Shadow)—marks the instant that the transmitter TR turns on or off.

3. Level 3 interface
   3.1 Packet Buffer—holds packets from local level 3; packet is represented by (local) level 3-assigned packet number.
      3.1.1 **I**—indicates number of oldest (local level 3) packet in buffer; in sequence 1, 2, ⋯.
      3.1.2 **Q**—indicates queue length in packet buffer.
   3.2 **R**(Packet Receiver)—receives packet number of packets received and passed by level 2.

## III. PERIPHERAL LEVEL 2 EQUIPMENT

The following pieces of level 2 equipment have no effect on protocol performance (relative to the level 2 task) and hence are not modelled here. Conceptually (and likely physically, too), they sit between the level 2 processes defined above and level 1. Those marked (A) (respectively (B)) sit between the level 2 outgoing (incoming) level 1 interface and outgoing (incoming) level 1.

(A) Flag generator—prefixes every frame with a flag consisting of the sequence 01111110, and generates continuous flags between frames.

(A) Frame checking sequence generator—appends a 16-bit Frame Checking Sequence (FCS) field to the end of every frame that provides a CRC for error detection.

(A) Bit-stuffer—provides for transparency of the flag by inserting a 0-bit after any sequence of five contiguous 1-bits in the address, control and FCS fields.

(A) Frame-abort sequence generator—transmits seven contiguous 1-bits (without 0's) when a frame in the course of transmission is aborted. (See comment in specification of level-1 Transmitter **TR** given in Section VII.)

(B) Idle link detector—detects an input of 15 contiguous 1-bits; when detected, the incoming link is declared idle and timer, **T2**, is set; when a 0-bit is detected, the idle condition is cleared. If **T2** expires in this mode, level 2 **STS** returns to a disconnected state, S1. (This is modelled by a disconnect request DISC retransmitted $N2$ times.)

(B) Invalid frame discarder—discards a frame not bounded by flags,

containing an address field other than 00000001 or 00000011, or having fewer than 32 bits between flags.
(B) Bit-destuffer—reverses bit-stuffing procedure between flags.
(B) Flat sensor—identifies beginning of frame by sensing opening flag (exactly six contiguous 1's). This facilitates the definition of the address, control, and information fields whose components provide the elements of the frame buffer, and it serves to mark the arrival of a new frame.
(B) CRC decoder—discards a frame which contains errors detected by CRC.
(B) Syntax checker—discards a valid error-free frame which contains any of the following syntactical faults:

- Control field unknown
- Receipt of an information field in a noninformation frame
- Information field exceeds maximum established length
- Receipt of an information field not containing an integral number of octets, if this is not allowed.

When such a frame is detected and discarded, an FRMR primitive is sent in response. (Such faults have no way of being generated with a valid implementation of levels 2 and 3 [short of the presence of undetected errors introduced by level 1]. However, this process is simulated by the N(R) analyzer process **N(R)A** which causes an FRMR to be sent if the N(R) of an incoming frame is out of range [not between the last received N(R) and the last transmitted sequence number N(S)]. This fault also has no way of being generated by a valid implementation of levels 2 and 3, short of undetected level 1 errors, and serves to represent, for the purpose of end-to-end performance analysis, all the possible syntax faults which result in a FRMR response.)
(B) FRMR information field generator—generates the special information field which is returned with an FRMR frame. (This field has no function with respect to the operation of the protocol. It is used exclusively for diagnostic purposes external to the normal operation of the protocol.)
(B) Frame buffer—buffers the frame address/control fields as they arrive from level 1, via the level 2 peripheral equipment listed above. The syntactically valid address/control fields are separated (by the frame stripper) into the five components listed below, as the bits arrive. Information bits (if any) may also be buffered while level 2 evaluates the (earlier arriving) address/control fields of the frame and determines whether the information field should be passed to level 3 or discarded. (See the packet receiver process **R** described in Section VII.) The address/control fields cannot be evaluated until they have completely arrived and the frame stripper has set the values of the

following five memory elements of the frame buffer:

1. Address
2. Control primitive
3. N(S)
4. P/F
5. N(R).

Once the values of all five elements are set, the frame stripper signals level 2 to evaluate the frame. However, the final determination about the frame cannot be made until the FCS field has been received, the frame has passed the CRC (error check), and the closing flag is checked as valid. (The physical delay associated with filling the frame buffer and checking for frame validity and correctness [CRC] is modelled here as part of the delay in level 1. There is no conceptual difference between a physical implementation in which a frame buffer is sequentially filled and then presented in toto to level 2 for evaluation, and a level 1 that receives the frame, pauses, and then presents the five frame elements [simultaneously] to level 2 for evaluation.)

(B) Frame discarding by peripheral equipment—the effect of frame loss, which results when any of the (unmodelled above-described) peripheral equipment discards a frame, is modelled here by loss of the frame in level 1. Hence, each frame seen by level 2 as modelled here is assumed to be error-free, valid, and syntactically correct, except possibly for an out-of-range N(R).

## IV. LEVELS 1 AND 3

In order to analyze the end-to-end peer-level performance of level 2, the channel between the local and remote ends (the physical buyer or level 1) must be modelled. Furthermore, as the end-to-end performance is stated in terms of delivery of level 3 packets, level 3 must be modelled to the extent of packet delivery to the adjoining level 2.

### *4.1 Level 3*

Level 3 is modelled implicitly as presenting two signals to level 2:

- Transmitting packet
- Idle.

These signals are not explicitly represented, but are presented as a nondeterministic filling of the packet buffer. (The actual level 2/level 3 interface is user defined, and may vary from implementation to implementation. However, its definition is not required in order to analyze the end-to-end performance of level 2.)

### *4.2 Level 1*

The end-to-end performance is analyzed in terms of packet throughput in the context of a local level 2, remote level 2, outgoing level 1

(from local end to remote end) and incoming level 1 (from remote end to local end). Of course, it is enough to define one level 2 and one level 1.

Level 1 is modelled as a frame buffer with delay, in order to absorb the delay in level 1/level 2 interfacing devices such as the CRC checker (error detection) and frame format checker. (These are technically part of level 2, but are not specifically modelled here since they do not affect the end-to-end behavior, aside from producing delay). The capacity of level 1 could be modelled as infinity. However, it can be shown on a theoretical basis that it will never hold more than $k$ frames, where $k$ is the maximum number of transmitted I-frames permitted to be unacknowledged by remote level 2. Furthermore, all the delays attributed to level 2 are smaller than the delay of transmitting one frame. (One frame is at least five octets or 40 bits long. For the purpose of this model, the delay of certain level 2 overhead, including error detection and frame format checking, has been attributed to level 1. This contributes a noncumulative delay of no more than the time required to transmit three octets.) Hence, level 1 may be modelled as a buffer with a capacity of only one frame, which is pushed out of the buffer by a nondeterministic selection change after an arbitrary delay. (The model thus models a situation which, in this respect, is more general than reality. There is no harm in doing this if the desired conclusions are nonetheless obtained. Presumably, we do not want a level 2 which is critically sensitive to timing considerations.)

The processes of level 1 are designed as follows:

Outgoing level 1

1. **C/R1** address buffer
2. **CP1** control primitive buffer
3. **N(S)1** N(S) buffer
4. **P/F1** P/F buffer
5. **N(R)1** N(R) buffer
6. **I1** I buffer
7. **R1** Receive moderator (indicates whether level 1 is ready to receive)
8. **S1** Send moderator (indicates whether level 1 is ready to send).

Incoming level 1 component processes are denoted by acronyms followed by a prime (').

### *4.3 Synchronization*

Incoming level 1 provides clocking to level 2 through its bit stream and flag generation. This establishes bit and octet synchronization with level 2. Hence, the process comprising local levels 2 and 3, the local host and incoming level 1 are all assumed to be synchronized

(driven by a common clock), and are modelled as synchronous processes. The *local end* is defined to be (the product of) these processes. Remote levels 2 and 3, the remote host and outgoing level 1 comprise the *remote end.* Thus end-to-end behavior is modelled by these two processes (the local end and the remote end) taken together. The stabilization[7] of each of these two processes represents the two ends as asynchronous (general) processes. Intuitively, the "asynchronous connection" between the local end and the remote end is provided by the transmitter process **TR** of level 2, which synchronizes its transmission to level 1 by sensing the level 1 clock. This is modelled by permitting transmission to level 1 only in the presence of a level 1 selection, "level 1-is-ready" (process **R1**), seen by the transmitter of level 2.

## V. LOCAL HOST

The computer comprising the residence of local levels 2 and 3 (local host) may disconnect the node (for preventive maintenance, for example). This is done in an orderly fashion. Before the actual disconnection, local level 2 informs the remote end that a disconnection is in progress. The actual disconnection takes place only once this notice has been acknowledged by the remote end.

Similarly, to reestablish a link, the host causes the level 2 link setup procedure to be initiated.

Furthermore, if local difficulties cause a temporary inability to process incoming frames (for example, a temporary malfunction which causes a buffer overflow condition), the local host may suspend incoming packets by issuing a busy signal. The effect of this is that local level 2 discards any incoming information fields and notifies the remote end of this condition (using the RNR primitive). When the condition terminates, the host clears the busy condition, and level 2 in turn notifies the remote end (with an RR primitive).

The interaction of the local host with local level 2 is modelled by providing the local host (process **H**) with four L-selections visible to local level 2, defined below.

| Host L-selection | Interpretation |
|---|---|
| (**H**:GO) | Local host start command |
| (**H**:STP) | Local host stop command |
| (**H**:B) | Station becomes busy |
| (**H**:NB) | Busy condition clears |

## VI. SYSTEM PARAMETERS

There are four system parameters upon whose definition this specification depends. (There are other system parameters, such as maxi-

mum allowed size of information field or timer periods, upon which this specification does not depend.) The four parameters are:

- Sequence number modulus (set at eight here and in OSN description)
- $N2$—maximum number of (re-) transmissions of a frame subsequent to the exploration of timer **T1**
- $k$—maximum number of transmitted I-frames permitted to be unacknowledged by remote level 2 ($0 < k < 8$)
- $Q_{max}$—capacity (in number of packets) of level 3 packet buffer ($0 < Q_{max} < 8$).

## VII. PROCESS SPECIFICATION

The *link layer interface* (level 2) is specified as a product of 27 component processes. Additionally, the *physical layer* (level 2) is specified as a product of 16 component processes (8 incoming, 8 outgoing). The processes, outlined in the previous section, are formally specified here.

### *7.1 Format*

The format of specification is as follows. The comments column includes the source in OSN for the given edge labels.

**<process acronym>** (<process number/name>)

<verbal description>
states: <list of state acronyms followed by (names)>
<comments>

selections: <list of L-selection acronyms followed by (names)>

<state acronym (name) A>⟶
<state acronym a>: <edge label of the edge (A,a)>
⋮
<state acronym (name) B>⟶
<state acronym a>: <edge label of the edge (B,a)>
⋮

### *7.2 Special notation*

1. Unlisted edge labels are 0 (no edge).
2. When the list of L-selection acronym (names) is identical to the list of state acronym (names), the former is omitted.
3. When states are identified implicitly in terms of parameters, the range of the parameters is described in the comments column.
4. The word "otherwise" is used to denote the complement of the disjunction (inclusive OR) of the other labels on edges outgoing from

the state in question, multiplied by the sum of the L-selections at that state.

5. $[N]$ denotes the smallest nonnegative integer congruent to $N$ modulo 8 (in particular, $[8] = 0$); $[i \leq m \leq n]$ denotes $[i] \leq [m] < [n]$ if $[i] \leq [n]$, and denotes $0 \leq [m] \leq [n]$ or $[i] \leq [m] \leq 7$ otherwise.

6. If **A** is a process, its L-selections are expressed as (**A**:S) where S is the selection identifier. The standard relations[6] are satisfied, namely, (**A**:S)(**A**:S′) = 0 if S ≠ S′, (**A**:S) and (**B**:S′) are independent if **A** ≠ **B** and $\sum_S$ (**A**:S) = 1.

7. A prefix dot (•) in edge labels of edges outgoing from a state S of a process **P** denotes the L-selection (**P**:S).

8. We may denote $\sum_{i=l}^{n}$ (**A**:$S_i$) ≡ (**A**:{$S_l$, ⋯ ,$S_n$}) when convenient.

9. Each process has a single initial state, which is either the state listed first, or in the case of a parameterized state variable, the smallest value of that variable.

### *7.3 Common elements*

The following are definitions of expressions which occur in the edge labels of more than one process. They may be used in implementation to centralize the redundant computations they present. Each expression is defined in terms of a character $\alpha$(<acronym>), headed in parentheses by the list of the process in which it appears, and meaning, if relevant.

(**C/R, PF** Poll may be set)

$$\alpha(\text{poll}) = (\mathbf{N(R)A}{:}\text{OK})[(\mathbf{STS}{:}(5,6,7))[(\mathbf{CP'}{:}\text{I})(\mathbf{LS}{:}\text{NB})(\mathbf{RS}{:}\text{NB}) + (\mathbf{CP'}{:}\{\text{RR:REJ}\})]\cdot(\mathbf{C/R'}{:}\text{C})(\mathbf{P/F'}{:}0) + (\mathbf{C/R'}{:}\text{R})] + (\mathbf{H}{:}\text{GO}) + (\mathbf{LS}{:}\text{STP}) + (\mathbf{RETR}{:}N2) + (\mathbf{T2}{:}\text{EXP})(\mathbf{STS}{:}3)$$

(**CP, P/F** A final transmission is pending)

$$\alpha(\text{fp}) = (\mathbf{C/R}{:}\text{R})(\mathbf{P/F}{:}1)(\mathbf{SD}{:}\text{C})$$

(**CP, TR, SD** All packets in buffer have been sent)

$$\alpha(\text{aps}) = \sum_n (\mathbf{PP}{:}n)(\mathbf{Q}{:}n)$$

(**CP, RETR, T1, I** No new acknowledgment of an outstanding I frame)

$$\alpha(\text{nn}) = \sum_n (\mathbf{UNACK}{:}n)(\mathbf{N(R)}{:}n)$$

(**TR, T1** All sent I frames have been acknowledged)

$$\alpha(\text{aa}) = \sum_n (\mathbf{UNACK}{:}n)(\mathbf{V(S)}{:}n)$$

*7.4 Level 2 processes*

The 27 processes which comprise local level 2 are specified here. The processes which comprise remote level 2 are identical but that to each of their acronyms is adjoined the suffix "r". (This may be seen in the edge labels of incoming level 1.)

For the interconnection of these 27 processes, see Figs. 1 and 3.

**C/R** (1.1.1.1 address assembler)

Identifies a frame as a command or response by defining the address field. (For command [respectively, response], address is set to that of the remote [local] end.) The edge labels are taken from OSN Table 2.1 and from the need to ensure that a pending "final" (i.e., a response with P/F bit 1) is not superceded as a result of a subsequent frame arrival, as described in OSN sections relating to the poll bit (see description of **P/F** process).

states: C (command)
R (response)

C (command) $\rightarrow$
R: $\cdot\tilde{\alpha}$(poll)[($\mathbf{T}_1$:EXP) + (**STS**:3)]
C: otherwise
R (response) $\rightarrow$
C: $\cdot$[(**P/F**:0) + (**SD**:L)][$\alpha$(poll) + ($\mathbf{T}_1$:EXP)(**STS**:3)]
R: otherwise

**CP** (1.1.1.2 Control Primitive Assembler)

Identifies command/response control primitive in the frame control field of the next frame to be sent. This may be updated several times before it is actually sent, if several frames arrive from level 1 during the course of transmission of a long information frame. Level 3 packets and send sequence numbers appear exclusively in frames containing the information control primitive I. Acronyms are those of OSN. Edge labels are from OSN Table 2.1 and 2.4.5.2b.

states: I (Information command) Frames containing this control primitive are called information frames and contain a send sequence number N(S) and a receive sequence number N(R) in the control field and an information field containing a level 3.

RR (Receiver Ready) Indicates local station is ready to receive an I frame; frame contains N(R) in the control field (acknowledging

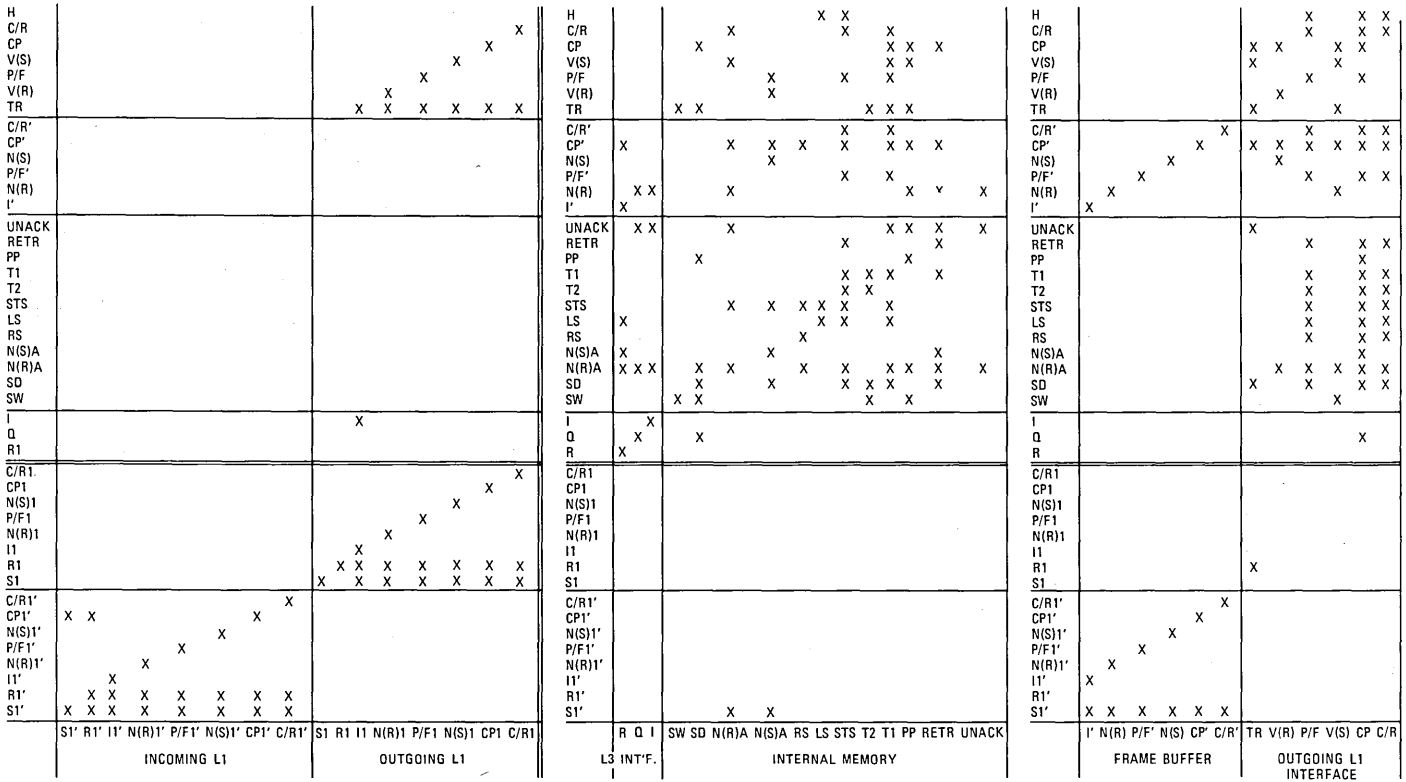



Fig. 3—In the following chart, a process **P** on the horizontal axis "sees" one or more selections of a process **Q** on the vertical axis if the grid point **(P, Q)** is marked by an X.

previously sent I frames). "Supervisory" frame.

| | | |
|---|---|---|
| RNR | (Receiver Not Ready) | Indicates host is busy (unable to accept packets); frame contains N(R). "Supervisory" frame. |
| REJ | (Reject) | Frame contains N(R) (acknowledging previously sent I frames); constitutes request to retransmit frames with send sequence numbers ≥ N(R). "Supervisory" frame. |
| SABM | (Set Asynchronous Balanced Mode Command) | Indicates host wants to set up link. |
| DISC | (Disconnect Command) | Indicates host is disconnecting from link (e.g., for prevention maintenance). |
| DM | (Disconnected Mode Response) | Indicates host is disconnected from link. |
| UA | (Unnumbered Acknowledgment Response) | Indicates receipt of SABM or DISC. |
| FRMR | (Frame Reject Response) | Indicates receipt of semantically bad frame (e.g., N(R) out of range). |

Set $\alpha$ = (**CP′**:I)(**P/F′**:0)(**STS**:{5,7})(**LS**:NB)(**RS**:NB)
+ (**CP′**:{RR,REJ})[(**C/R′**:C)(**P/F′**:0)
+ (**C/R′**:R)](**STS**:{5,6,7}),
$\beta$ = (**LS**:{NB,B})(**RETR**:N2)(**N(S)A**:OK)(**N(R)**A:OK)
$\gamma$ = [(**T1**:EXP) + (**C/R′**:R)(**P/F′**:1)](**T2**:EXP). cf: OSN 2.4.5.8

$\upsilon$(parameter) → $\upsilon \in$ {I.RR, ..., FRMR}

I: $\cdot\tilde{\alpha}(\text{fp})\tilde{\alpha}(\text{aps})\alpha\beta\gamma$

RR: $\cdot\tilde{\alpha}(\text{fp})\beta$(**STS**:{5,6,7})[$\gamma${(**CP′**:{I,RR,RNR,REJ})(**P/F′**:1)
(**LS**:NB) + (**CP′**:I)(**P/F′**:0)(**LS**:NB)(**RS**:B) + $\alpha$(aps)$\alpha$}
+ [(**T1**:EXP) + (**T2**:EXP)][(**STS**:5)(**RS**:NB) +
(**STS**:6)(**RS**:B)](**LS**:NB) + (**T1**:EXP)(**STS**:7)(**LS**:NB)]
The $\alpha$(aps)$\alpha$ term is from OSN 2.4.5.1.2.

RNR: $\cdot$(**RETR**:N2)[(**STS**:{5,6,7})[(**H**:B)(**LS**:NB) +
(**LS**:B)[(**CP′**:1) + (**P/F′**:1)(**CP′**:{RR,RNR,REJ}) +
(**T1**:EXP) + (**T2**:EXP)]]]

REJ: $\cdot\tilde{\alpha}(\text{fp})$(**LS**:NB)(**RETR**:N2){[(**T1**:EXP) +
(**T2**:EXP)](**STS**:{5,6})(**RS**:NB) +
(**N(S)A**:REJ)(**STS**:5)}

SABM: $\cdot${(**CP′**:DM)(**STS**:1) + (**CP′**:{DM,FRMR})(**STS**:3) +
(**CP′**:{DM,FRMR,UA})(**STS**:{5,6,7}) +

(**T1**:EXP)(**STS**:1) + [(**T1**:EXP) + (**T2**:EXP)](**STS**:2)
+ (**RETR**:N2)(**STS**:{3,5,6,7} + (**H**:GO)}

DISC: ·$\tilde{\alpha}$(fp){(**LS**:STP)(**STS**:{1,4}) + [(**T1**:EXP) +
(**T2**:EXP)](**STS**:4)}

DM: ·{(**STS**:1)[(**CP′**:{I,RR,REJ,RNR})(**C/R′**:C)(**P/F′**:1) +
(**CP′**:DISC)] + (**STS**:2)(**CP′**:DISC) +
(**STS**:4)(**CP′**:SABM)}

UA: ·[(**CP′**:SABM)(**STS**:4) + (**CP′**:DISC)(**STS**:{3,5,6,7})]

FRMR: ·(**N(R)A**:FRMR)

*v*: otherwise

**V(S)** (1.1.1.3 sequence number generator)

Indicates the sequence number N(S) (assigned here) in the frame control field of the next information frame to be transmitted. This is "seen" by level 1 (i.e., is part of the frame transmitted to level 1) in I frames only.

states: $N$ (parameter) $0 \leq N < 8$

$N \rightarrow$

[$N$ + 1] ·(**SW**:0)(**TR**:ON)(**CP**:I) OSN 2.3.1.4.1

$M$: ·(**CP′**:REJ)(**N(R)**:M)(**N(R)A**:OK) OSN 2.3.5.5, $0 \leq M < 8$

0: ·[(**CO′**:SABM) + (**CP**:SABM)] if $N \neq 0$; OSN 2.3.3.5

$N$: otherwise

OSN refers to initialization of N(S) only upon receipt of SABM; presumably, this must be done by sending end as well, as modelled here.

**P/F** (1.1.1.4 poll/final bit assembler)

Indicates value (0 or 1) of poll/final bit in the frame control field of the next frame to be transmitted. (OSN is very vague about this. According to Fred Berg [private communication], the purpose of a poll [P/F bit set to 1 in a command frame] is to force the other end to respond to the associated command, rather than to wait and respond possibly to a subsequent command [or I frame], as is allowed if there is no poll. The response is required to be a "final" [P/F bit set to 1 in a response frame], which indicates it is a response to the outstanding poll, of which there is allowed to be at most one [from each end] at any time. However, when to send a poll is, with one exception, nowhere specified in OSN and thus is left to the implementer. The exception is that when timer T1 expires, a poll is required [OSN 2.4.5.8]. Hence, it must be the intention of OSN that the protocol be equally valid, independent of the extent to which polls are ever used. Furthermore, this polling convention leads to certain inefficiencies. For example, the response during the information transfer to a supervisory poll

must be an RR [final], whereas an I frame would convey the same information [less the final bit] more efficiently.) The following sections of OSN mention the P/F bit: 2.3.2, 2.3.3.2, 2.3.3.3, 2.3.3.4, 2.3.3.8, 2.3.4.2, 2.4.3, 2.4.4.4, 2.4.4.6, 2.4.5.5, 2.4.5.7, 2.4.5.8, 2.4.7, 2.4.8.1.

states: 0 Outgoing frame not poll/final.
1 Outgoing frame is poll or final.

$v$ (parameter) $\rightarrow$ $v \in \{0,1\}$

1: $\cdot$[(**C/R′**:C)(**P/F′**:1)(**N(R)A**:OK) + $\alpha$(fp)
+ (**STS**:3)(**T1**:EXP) + $\alpha$(poll)]

0: $\cdot$[$\alpha$(fp)[(**C/R′**:R) + (**P/F′**:0)] + $\alpha$(poll)]

$\alpha$(poll) is added to both edges to model polling conventions for all legal implementations.

**TR** (1.1.2 transmitter)

Indicates to level 1 the on/off status of the frame transmission. When the transmitter is on, level I receives sequentially the bits comprising a frame, including those of the address/control fields and the information field if the frame is an I frame or FRMR frame. When the transmitter finishes to send a frame, it turns off and one or more flags are then (automatically, by unmodelled peripheral equipment) sent across level 1. The periods during which the transmitter is on can be of variable length, depending upon the length of the information field and the amount of bit-stuffing required. This is modelled by a nondeterministic state change from the "on" states to the OFF state. The lengths of the periods when the transmitter is off must be an integer multiple of the length of time required to send one flag, i.e., one octet (8 bits). This is modelled by a level 1 state change from the "not ready" state to the "ready" state which is assumed to occur at each instant that a complete flat has been sent.

The transmitter process also optionally aborts the transmission of an I frame when a REJ primitive is received concurrently (in an error-free, valid, syntactically correct frame). The delay associated with the transmission of the frame abortion signal by the peripheral frame-abort-sequence generator is modelled here as if absorbed in the delay until the "on" state is next reached.

states: OFF
ON/I (transmitting I frame)
ON (transmitting non-I frame)
AB (abort)

selections: (**TR**:OFF)
(**TR**:ON)
(**TR**:ON)
(**TR**:AB)

OFF →
ON/I: ·(**SD**:C)(**CP**:I)(**R1**:1)
ON: ·(**SD**:C)(**CP**:1)(**R1**:R)
OFF: ·otherwise

**V(R)** (1.1.1.5 received sequence number assembler)

Indicates the sequence number N(S) (assigned at remote end) in frame control field of the next expected information frame to be received. This is "seen" by level 1 (i.e., is a part of the frame transmitted to level 1) in I frames and supervisory frames (RR, RNR, REJ) only.

states: $N$ (parameter) $0 \leqslant N < 8$
$N$ →
[$N+1$] · (**CO**:I)(**N(S)**:N)(**N(R)A**:OK) OSN 2.3.1.4.3
0: · [(**CP**:SABM) + (**CP**:SABM)] OSN 2.3.3.5
$N$: otherwise

OSN refers to initialization of V(R) only upon receipt of SABM; presumably, this must be done by sending end as well, as modelled here.

ON/I (transmitting I frame) →
OFF: (**TR**:ON) OSN 2.4.5.5b
AB: (**TR**:ON)(**N(R)A**:OK) Abort is optional.
(**CP'**:{REJ,SABM})
ON/I: (**TR**:ON) OSN does not describe abort upon receipt of SABM: added here.

ON: (transmitting non-I frame) →
OFF: ·
ON: ·
AB: (abort) →
OFF: ·$\alpha$(aa)
AB: otherwise

**C/R'** (1.2.1 incoming address buffer)

Buffers the address field of a frame arriving from level 1.

states: C (command)
R (response)
C (command) →
R: ·(**C/R1'**:R)(**S1'**:1)
C: otherwise
R (response) →
C: ·(**C/R1'**:C)(**S1'**:1)
R: otherwise

**CP′** (1.2.2 incoming control primitive buffer)

Buffers the control primitive in the control field of a frame arriving frame level 1.

states: I — For names of states, see control primitive assembler process 1.1.1.2.
RR
RNR
REJ
SABM
DISC
DM
UA
FRMR

$v$ (parameter) → $v,w \in$ {I,...,FRMR}
$w$:·(**CP1′**:$w$)(**S1′**:1) $w \neq v$
$v$: otherwise

**N(S)** (1.2.3 incoming sequence number buffer)

Buffers the sequence number in the control field of a frame arriving from level 1.

states: N (parameter) $0 \leq N < 8$
N (parameter) → $0 \leq N,\mathrm{M} < 8$
$M$ (parameter):·(**N(S)1′**:M)(**S1′**:1) $M \neq N$
$N$: otherwise

**P/F′** (1.2.4 incoming poll/final bit buffer)

Buffers the poll/final bit in the control field of a frame arriving from level 1.

states: 0
1
$v$ (parameter) → $v,w \in \{0,1\}$
w (parameter):·(**P/F1′**:$w$)(**S1′**:1) $w \neq v$
$v$: otherwise

**N(R)** (1.2.5 incoming receive sequence number buffer)

Buffers "receive sequence number" in the control field of a frame arriving from level 1.

states: N (parameter) $0 \leq N < 8$
$N$ (parameter) → $0 \leq N,M < 8$
$M$ (parameter):·(**N(R)1′**:M)(**S1′**:1) $M \neq N$
$N$: otherwise

**I′** (1.2.6 incoming information field buffer)

Buffers the packet number of the packet in the information field of a frame arriving from level 1.

states: $N$ (parameter) $0 < N$
$N$ (parameter) → $0 < N,M$
$M$: (**I1′**:M)(**S1′**:1) $0 < N,M$
$N$: otherwise

**UNACK** (2.1.1 first unacknowledged frame number counter)

Indicates the next expected receive (i.e., acknowledgment) sequence number N(R) in a received (information or "supervisory") frame. Receipt of the number N(R) = $N$ constitutes acknowledgment from the remote end that the (locally generated) information frame with sequence number N(S) = $N$ was received by the remote end.

states: N (parameter) $0 \leq N < 8$
$N$ (parameter) → $0 \leq N,M < 8$
$M$ (parameter): · **(N(R)**:M)**(N(R)A**:OK) $M \neq N$
$N$: otherwise

**RETR** (2.1.2 retransmission counter)

Counts the number of (re-) transmissions of a frame subsequent to the expiration of timer T1. OSN 2.4.5.8 and 2.4.8.2.

states: N (parameter) $0 \leq N \leq N2$
$N$ (parameter) → $0 \leq N < N2$
$N + 1$: · **(T1**:EXP)
0: · [**(CP**:{UA,RNR}) +
**(CP′**:{I,RR,RNR,REJ}).
**(N(S)A**:OK)**(N(R)A**:OK)$\tilde{\alpha}$(nn)]
$N$: otherwise

OSN is vague about what reception is correct enough to warrent resetting RETR to 0 (e.g., RNR is accepted, according to OSN 2.4.5.8, even with **(N(R)A**:FRMR) whereas acceptance of I with **(N(S)A**:REJ) is not indicated positively or negatively).

$N2$ →
0: · **(CP**:SABM)**(SD**:L)
$N2$: otherwise

**PP** (2.1.3 packet pointer)

Points to packet in packet buffer that was last transmitted to level 1. (This process "goes back $N$" for retransmission of lost packets. When acknowledgments arrive, it must decrement accordingly, to keep pace with the packet buffer queue length, process **Q**.)

states: $N$ (parameter) $0 \leq N \leq Q_{max}$
$N$ → $0 \leq N \leq Q_{max}$
$N + 1$: · **(SW**:0)**(TR**:ON)**(CP**:I) if $N < Q_{max}$
$M$: · **(N(R)A**:OK) $\sum_{[j-i]=N-M}$ {**(N(R)**:$i$)**(UNACK**:$j$)
+ **(CP′**:REJ)**(N(R)**:$i$)**(V(S)**:$j$)}
$N$: otherwise

**T1** (2.2.1 timer 1)

Times unacknowledged frame, pending initiation of retransmission

procedure. (Expiration of timer is modelled as a nondeterministic selection change, and thus is more general than reality.)
states: OFF
ON
EXP (expired)
Define
$\alpha$ = (**CP**:{REJ,SABM,DISC,FRMR}) + (**C/R**:C)(**CP**:RR)
+ $\tilde{\alpha}$(aa) + (**C/R**:C)(**P/F**:1)
OFF →
ON: · $\alpha$
OFF: · [$\tilde{\alpha}$ + (**CO**:FRMR)] OSN 2.4.4.1, 2.4.4.3, 2.3.3.3, 2.3.4.3, 2.4.5.4, 2.4.8.1 and 2.4.6, in which setting T1 when sending FRMR is unformal. In part, $\alpha$ is defined in accordance with the comment to state 6 of STS.

Define
$\beta$ = (**STS**:{2,4})(**CP′**:UA) + (**N(R)A**:OK)(**CP′**:{RR,RNR,REJ})
(**C/R′**:R)(**P/F′**:1) + $\tilde{\alpha}$(nn)
ON →
OFF: · $\beta$ OSN 2.4.4.1, 2.4.5.4, 2.4.5.8
EXP: · $\tilde{\beta}$
ON: · $\tilde{\beta}$
EXP →
OFF: · (**TR**:ON)(**SD**:L)[(**STS**:{1,2})(**CP**:SABM) +
(**STS**:3)(**CP**:FRMR) + (**STS**:4)(**CP**:DISC) +
(**STS**:{5,7})(**LS**:NB)(**CP**:RR) + (**STS**:6)(**LS** <
**NB**)(**CP**:REJ) + (**STS**:{5,6,7})(**LS**:B)(**CP**:RNR) +
(**N(R)A**:OK)(**P/F′**:1)(**C/R′**:R)$\tilde{\alpha}$(nn)]
EXP: otherwise

**T2** (2.2.2 timer 2)

Times period of "link idleness" and periods during which consecutive flags are received. (Expiration of timer is modelled as a nondeterministic selection change, and this is more general than reality.)
states: OFF
ON
EXP (expired)
OFF →
ON: ·(**T1**:OFF)
OFF: otherwise
ON → It is assumed that T2 times any combination of link idleness and consecutive flags (OSN is vague about this).

If the local end rejects a frame while the remote end enters a failure mode wherein it transmits continuous flags (only), the local end will eventually wait forever in **STS** state 7, without any further transmissions.

EXP: ·(**T1**:ON)(**TR**:OFF)
OFF: ·(**T1**:ON) + (**TR**:OFF)]
ON: ·(**T1**:ON)(**TR**:OFF)
EXP →
OFF: ·(**TR**:OFF)(**SW**:0)(**SD**:L)
EXP: otherwise

**STS** (2.3.1 connection status)

Indicates status of level 2 link control. Acronyms, names, and definition are from OSN Table 2.1. (Collisions of conflicting commands from remote end, host or internal devices such as timers are rarely resolved in OSN; here, in each case a [hopefully] reasonable resolution is given.)

states: 1 (disconnected)
2 (link setup)
3 (frame rejected)
4 (disconnect request)
5 (information transfer)
6 (REJ control primitive sent)
7 (waiting acknowledgment)

1 (disconnected) →
2:·[(**CP′**:DM) + (**T1**:EXP) + (**H**:GO)](**CP′**:SABM)
5:·(**CP′**:SABM)
1: otherwise
2 (link setup) →
1:·(**CP′**:{DM,DISC})(**LS**:STP)
4:·(**LS**:STP)(**CP′**:UA)
5:·(**CP′**:UA)
2: otherwise
3 (frame rejected) →
1:·(**CP′**:DISC)
2:·[(**CP′**:{DM,FRMR}) + (**RETR**:*N*2) +
(**H**:GO)](**LS**:STP)(**CP′**:{SABM,DISC})
4:·(**LS**:STP)(**CP′**:{SABM,DISC})
5: (**CP′**:SABM)
3: otherwise
4 (disconnect request) →
1:·[(**CP′**:{UA:DM}) + (**RETR**:N2)](**H**:GO)
2:·(**H**:GO)
4: otherwise

Set $\alpha$ = [(**CP′**:{UA,DM,FRMR}) + (**RETR**:$N2$) + (**H**:GO)]
·(**CP′**:DISC)(**N(R)A**:FRMR)[(**LS**:STP) + $\alpha$(fp)].
$\beta$ = (**N(R)A**:FRMR)(**CP′**:DISC),
$\gamma$ = $\tilde{\alpha}$(fp)(**LS**:STP)(**CP′**:DISC)(**N(R)A**:FRMR),
$\delta$ = $\tilde{\alpha}$(fp)[(**T1**:EXP) + (**T2**:EXP)](**CP′**:{DISC,UA,DM,FRMR})
(**RETR**:NZ)(**H**:GO)(**N(R)A**:FRMR)(**LS**:STP),
$\epsilon$ = $\tilde{\alpha}$(fp)(**LS**:NB)(**N(S)A**:REJ)(**N(R)A**:OK)
5 (information transfer) →
1:·(**CP′**:DISC)
2:·$\alpha\tilde{\epsilon}$
3:·$\beta$
4:·$\gamma$
6:·$\epsilon$(**CP′**:DISC)
7:·(**LS**:NB)$\delta\tilde{\epsilon}$
5: otherwise
6: (REJ control primitive sent) →
1:·(**CP′**:DISC)
2:·$\alpha$(**CP′**:{I,SABM})
3:·$\beta$
4:·$\gamma$
5:·(**LS**:NB)[$\tilde{\alpha}$(fp)(**CP′**:I) + (**CP′**:SABM)].
(**N(R)A**:FRMR)(**CP′**:DISC)
7:·$\delta$(**CP′**:{I,SABM})
6: otherwise

Modelled according to OSN Table 2.1.c which permits at most 1 retransmission of REJ, this contradicts OSN 2.3.4.3 which permits up to $N2$ retransmission of REJ. Here, RR rather than REJ is retransmitted upon expiration of T1, up to $N2$ times.

7 (waiting acknowledgment) →
1:·(**CP′**:DISC)
2:·$\alpha$(**CP′**:SABM)[(**CP′**:{RR,RNR,REJ}) + (**C/R′**:C) + (**P/F′**:0)]
3:·$\beta$
4:·$\gamma$
5:·[(**CP′**:SABM) + [$\tilde{\alpha}$(fp)(**CP′**:{RR,REJ}) + (**CP′**:RNR)]
(**C/R′**:R)(**P/F′**:1)]
7: otherwise

**LS** (2.3.2 local host status)

Indicates status of local end as deduced from last received control signal from the local host. OSN Table 2.1.

states: NB (not busy)
B (busy)
STP (stop)

OFF
NB (not busy) →
B: ·(**H**:B)
STP:·(**H**:STP)
NB: otherwise
B (busy) →
NB: ·(**H**:NB)
STP:·(**H**:STP)
B: otherwise
STP (stop) →
OFF:·(**STS**:1)
NB ·(**H**:GO)
STP: otherwise
OFF →
NB: ·(**H**:GO)
OFF: otherwise

**RS** (2.3.2 remote host status)

Indicates busy/not busy status of remote end as deduced from a receipt of RNR control primitive from remote level 2, based upon OSN Table 2.1.

states: NB (not busy)
B (busy)
NB (not busy) →
B: ·(**CP'**:RNR)(**N(R)A**:OK)(**STS**:{5,6,7})
NB: otherwise
B (busy) →
NB:·(**CP'**:{RR,REJ,SABM})(**N(R)A**:OK)
B: otherwise

**N(S)A** (2.3.4 N(S) analyzer)

Indicates whether or not the sequence number N(S) of the last received (information) frame is the next expected sequence number. If not, it causes a retransmission request (REJ primitive) to be sent.
OSN 2.3.4.2

states: OK
TEST
REJ
OK →
TEST:·(**S1'**:1)
OK: otherwise
TEST →
OK:·$[(\mathbf{STS}{:}\{5,6,7\}) + (\mathbf{CP'}{:}1) + \sum_{n} (\mathbf{V(R)}{:}n)(\mathbf{N(S)}{:}n)]$

REJ:otherwise
REJ →

OK: $\cdot\tilde{\alpha}$(fp)
REJ: $\cdot\alpha$(fp)

**N(R)A** (2.3.5 N(R) analyzer)

Indicates whether or not receive sequence number N(R) in the last received (information or "supervisory") frame is "within range," i.e., whether or not [UNACK* ≤ N(R)* ≤ V(S)*], where (*) denotes the state of the associated counter. An N(R) is within range if and only if it acknowledges a sent but yet unacknowledged frame (in which case it also acknowledges all previous unacknowledged frames). If N(R) is not within range, the N(R) analyzer process **N(R)A** causes the link to be reinitialized (the FRMR primitive is sent), and all previously transmitted but unacknowledged I frames are cleared from the packet buffer (they remain unacknowledged and are not retransmitted). (According to OSN 2.3.3.5, detection and recovery from the possible loss of such frames is left to a higher-level protocol. I find no reason not to provide for recovery in level 2. This could be accomplished merely by retransmitting all frames starting with the first unacknowledged one, without reinitializing the link. However, the specification given below is consistent with OSN 2.3.3.5.)

states: OK (within range)
TEST
FRMR (not within range)

OK (within range) →
TEST: $\cdot$(**S1**′:1)
OK: otherwise
TEST →
OK: $\cdot[(\mathbf{STS}{:}\{5,6,7\}) + (\mathbf{CP}'{:}\{\text{I,RR,RNR,REJ}\}) + \sum_{[i\leq n\leq j]} (\mathbf{UNACK}{:}i)(\mathbf{N(R)}{:}n)(\mathbf{V(S)}{:}j)]$ OSN 2.4.7c
FRMR: otherwise
FRMR →
OK: $\cdot$(**STS**:3)
FRMR: otherwise

**SD** (2.3.6 sender)

Classifies current status of header assembler as "to be sent" or "already sent." (This is not an explicit process of OSN; it is used in conjunction with the transmitter process **TR** to coordinate buffering of outgoing frame.)

states: L (last) Header already sent.
L′ Placekeeping state.
C (continuing) Header to be sent.

selections: (**SD**:L)
(**SD**:C)

L (last) →
L′:·[(**N(R)A**:TEST) + $\tilde{\alpha}$(aps)(**CP**:1)]
L: otherwise
L′ →
C: (**SD**:L)(**N(R)A**:TEST)[$\alpha$(aps) + (**CP**:1)]
L′: (**SD**:L)[(**N(R)A**:TEST) + $\tilde{\alpha}$(aps)(**CP**:I)]
C (continuing) →
L:·(**TR**:ON)(**SW**:0)
C: otherwise

**SW** (2.3.7 TR shadow)

Marks the instant that the transmitter process **TR** turns on and off. (Implicit process for control of certain counter.)

states: 0 (transmitter was OFF)
1 (transmitter was not OFF)
0 (transmitter was OFF) →
1: · (**TR**:OFF)
0: otherwise
1 (transmitter was not OFF) →
0: · (**TR**:OFF)
1: · otherwise

**I** (3.1.1 packet counter)

Indicates the smallest in-sequence level 3-assigned number of a packet unacknowledged by the remote end.

states: $N$ (parameter) $N \geq 1$
$N$ →
$N+1$:·(**N(R)A**:OK)$\tilde{\alpha}$(nn)
$N$: otherwise

**Q** (3.1.2 packet buffer queue length)

Indicates the number of packets N in the packet buffer. (By assumption, the packets in the packet buffer bear numbers $M$, $M+1$, ..., $M+N-1$ where $M$ is the state of the packet counter **I**.) The buffer may be loaded by local level 3; this is modelled by a nondeterministic jump in state from $N$ to $N+1$.

states: $N$ (parameter) $0 \leq N \leq Q_{max}$
$N$ →
$N+1$:· $N < Q_{max}$
$M$: · (**N(R)A**:OK) $\sum_{[j-i]=N-M}$ $0 \leq M < N$
(**N(R)**:i)(**UNACK**:j)
$N$: · (**N(R)A**:OK)
$N$ →
$N+1$:· if $0 \leq N < Q_{max}$

$M+1$:· **(N(R)A:OK)** $\sum_{[j-i]=N-M}$ **(N(R):**$i$**)(UNACK:**j**)** if $0 \leqslant M < N-1$

$M$: · **(N(R)A:OK)** $\sum_{[j-i]=N-M}$ **(N(R):**i**)(UNACK:**j**)** if $0 \leqslant M < N$

$N$: otherwise $0 \leqslant N \leqslant Q_{\max}$

The transition from $N$ to $M+1$ models the simultaneous acknowledgment of $N-M$ frames and the reception of a frame from level 3.

**R** (3.2.1 packet receiver)

Indicates the level 3 packet number of last packet passed by level 2 to level 3. (The passing of the actual packets is modelled here by the passing of these packet numbers. The transmission of bits from level 1 to level 3 is assumed to occur in real time. That is, it is assumed that level 2 can transmit to level 3 as fast as level 1 can transmit to level 2. The sequence of events during the course of which a packet passes from level 1 [through level 2] to level 3 is as follows. An incoming information frame is first analyzed by peripheral level 2 equipment (B). If the frame is not immediately discarded on the basis of an invalid opening flag on syntactically bad address/control fields, the control field is analyzed by the N(S)/N(R) analyzers. The control field is followed immediately by the information field [packet]. Two possible ways to treat this packet are (1) to immediately transmit it to level 3 and then abort this transmission if the level 2 analysis indicates the frame must be discarded or (2) to buffer the packet during the course of the level 2 analysis. [In option (2), virtually the entire frame must be buffered, as the CRC check and closing-flag-valid check cannot be made until the entire frame has been received.] In either case, the checks on the frame are not all simultaneous. When the final check test is passed [valid closing flag] the UNACK and V(R) counters are updated.)

states: $N$ (parameter) $N \geqslant 0$

$N \rightarrow$

$M$ ·**(CP′:I)(N(S)A:OK)(N(R)A:OK)(LS:NB)(I′:M)** $M \neq N$

$N$: otherwise

### *7.5 Level 1 processes*

The eight processes comprising incoming level 1 are modelled here. Each process has an acronym which ends with a prime (′). The processes of outgoing level 1 are identical but that their acronyms omit the prime (′).

**C/R′** (4.1 incoming L1 address)

Signals the address field of a frame arriving from remote level 2.

states: C
R
$\phi$ (null state)

v (parameter) → $v \in \{C,R\}$
$\phi$:·(**S1′**:1)
v: otherwise
$\phi$ (null state) $v \in \{C,R\}$
v ·(**R1′**:1)(**TRr**:ON)(**C/Rr**:$v$)
$\phi$: otherwise

**CP1′** (incoming control primitive)

Signals the control primitive in the control field of a frame arriving from remote level 2.

states: I
RR
RNR
REJ
SABM
DISC
DM
UA
FRMR
$\phi$ (null state)

For names of states, see control primitive assembler process 1.1.1.2.

v (parameter) → $v \in \{I,...,FRMR\}$
$\phi$:·(**S1′**:1)
v: otherwise
$\phi$ (null state) →
v:·(**R1′**:1)(**TRr**:ON)(**CPr**:$v$) $v \in \{I,...,FRMR\}$
$\phi$: otherwise

**N(S)1′** (4.3 incoming L1 sequence number)

Signals the sequence number in the control field of a frame arriving from remote level 2. (The state of **N(S)** is equal to the state of **V(S)** when the frame was transmitted).

states: N (parameter) $0 \leq N < 8$
$\phi$ (null state)

N (parameter) →
$\phi$ ·(**S1′**:1)
N: otherwise
$\phi$ (null state) →
N·(**R1′**:1)(**TRr**:ON)(**V(S)**r:N)(**CPr**:I)
$\phi$: otherwise

**P/F1′** (4.4 incoming L1 poll/final bit)

Signals the poll/final bit in the control field of a frame arriving from remote level 2.

states: 0
1
$\phi$ (null state)

v (parameter) → $v \in \{0,1\}$

$\phi \cdot$(**S1′**:1)

v: otherwise

$\phi$ (null state) →

v:·(**R1′**:1)(**TR**r:ON)(**P/F**r:v)

$\phi$: otherwise

**N(R)1′** (4.5 incoming L1 receiver sequence number)

Signals the state of the received sequence number assembler **V(R)** in the control field of a frame arriving from remote level 2.

states: N (parameter) $0 \leq N < 8$
$\phi$ (null state)

N (parameter) →

$\phi$:·(**S1′**:1)

N: other

$\phi$ (null state)

N:·(**R1′**:1)(**TR**r:ON)(**V(R)**r:N)(**CP**r:{I,RR,RNR,REJ})

$\phi$: otherwise

**I1′** (4.6 incoming L1 information field)

Signals the packet number of the packet in the information field of a frame arriving from remote level 2.

states: N (parameter) $0 < N$
$\phi$ (null state)

N (parameter) →

$\phi$:·(**S1′**:1)

N: other

$\phi$ (null state) →

N:·(**R1′**:1)(**TR**r:ON)(**I**r:N)(**CP**r:1)

$\phi$: other

**R1′** (4.7 incoming L1 receive moderator)

Indicates to remote level 2 whether or not incoming level 1 is ready to receive a frame. (This process provides synchronization for transmissions from remote end to local end.)

states: 0 (not ready to receive)
1 (ready to receive)

0 (not ready to receive) →

1:·(**CP1′**:$\phi$)

0:·

1 (ready to receive) →

0: · (**TRr**:OFF)(**SWr**:1)
1: otherwise

**S1′** (4.8 incoming L1 send moderator)

Indicates to local level 2 whether or not level 1 is ready to send a frame.

states: 0 (not ready to send)
1 (ready to receive)

0 (not ready to receive) →
1: · (**CP1**:$\phi$)
0: ·

1 (ready to receive) →
0: ·

## VIII. ACKNOWLEDGMENTS

I wish to thank Fred Berg and Paul Long for their generous contributions to my understanding of BX.25, which understanding, if still imperfect, reflects only upon myself.

## REFERENCES

1. A. S. Tanenbaum, *Computer Networks*, Englewood Cliffs, N.J.: Prentice-Hall, 1981.
2. K. Bartlett and D. Rayner, "The Certification of Data Communication Protocols," Proc. Trends and Applications Symp., NBS, 1980.
3. G. V. Bochmann and R. J. Chung, "A Formalized Specification of HDLC Classes of Procedures," Proc. NTC 1977, Los Angeles, 3A2-1/11.
4. P. Zafiropulo, et al., "Towards Analyzing and Synthesizing Protocols," IEEE Trans. Commun., *COM-28* (1980), pp. 651–60.
5. T. P. Blumer and R. L. Tenney, "A Formal Specification Technique and Implementation Method for Protocols," Tech. Report No. 81-15, Bolt, Beranek, and Newman, Inc., 1981.
6. R. P. Kurshan, "Coordination of Concurrent Probabalistic Processes," Lect. Notes in Contr. Inf. Sci., *58* (1984), pp. 605–14.
7. S. Aggarwal, R. P. Kurshan, and K. Sabnani, "A Calculus for Protocol Specification and Validation," Proc. IFIP Protocol Spec., Test., Verif., North-Holland (1983), pp. 19–34.

## AUTHOR

**Robert P. Kurshan,** Ph.D. (Mathematics), 1968, University of Washington; Krantzberg Chair for Visiting Scientists, Technion, Haifa, Israel, March–August 1976; AT&T Bell Laboratories, 1968—. Mr. Kurshan is a member of the Mathematics Research Center. His current interests concern formal development, specification, and analysis of distributed systems such as communication protocols.

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Approximate Analysis of a Generalized Clocked Schedule

By A. A. FREDERICKS, B. L. FARRELL, and D. F. DEMAIO*

(Manuscript received December 8, 1983)

Clocked schedules represent an important method of task scheduling for computer systems with real-time applications. In this paper we consider a generalized class of clocked schedule that includes those used in many stored program control switching systems. Key performance measures for this class are discussed, and an analytic approximation method for analyzing certain of these measures is given. This approximation method is most applicable in evaluating long-term delays. (A companion paper by Doshi addresses short-term delays for systems with extremely time-critical tasks.) Comparisons are made with exact numerical results (obtained using the method presented in a companion paper by Ackroyd), detailed simulation models, and field data.

## I. INTRODUCTION

Processor scheduling concerns specifying when each task that must be done in a computer system is to be scheduled for execution, and how conflicts in task execution are to be resolved—e.g., by setting task priorities. In this paper we consider the performance analysis of a class of schedules for computer systems with real-time applications, that is, applications where at least some tasks have time-critical execution requirements. Collection of digits in a call-processing system is one of the more important examples of a time-critical task.

To introduce the concept of a clocked schedule, we consider the simple example system of Fig. 1. The processor must respond in a

* Authors are employees of AT&T Bell Laboratories.

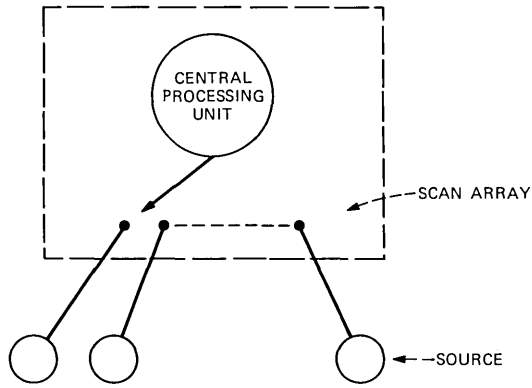



Fig. 1—Example system.

timely manner to requests from sources to send data. This consists of recognizing new requests, doing setup work (e.g., assigning buffers, sending a "ready to receive" signal), and collecting the data sent. For concreteness, consider this to be a microprocessor system handling incoming dial pulse trunks. Collecting the data corresponds to counting the number of pulses sent (number dialed). We can distinguish at least two distinct tasks: task 1, collecting pulses on active trunks; task 2, scanning inactive trunks for new requests and performing needed setup work. Task 1 is clearly the most time critical. Figure 2 shows a clocked schedule for this system. Time is divided into intervals (slots) of length $T$ (dictated by the on-off lengths of the pulses), and in each slot, first task 1 is executed (for all active trunks), then task 2. If, at the beginning of a time slot, task 2 is executing, it is interrupted and task 1 execution is given control.

In applications of this type, the large number of stimuli (e.g., state

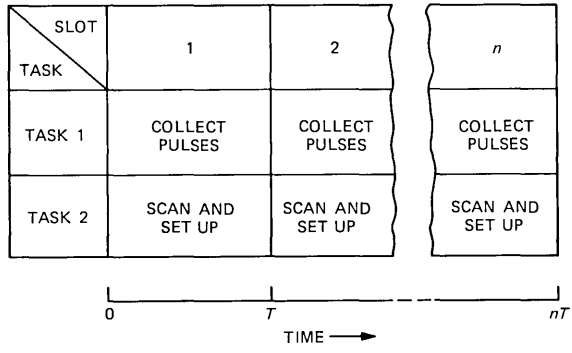



Fig. 2—Clocked schedule for example system.

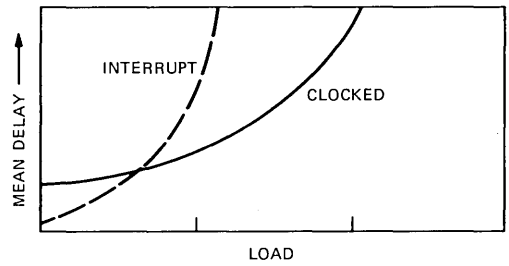



Fig. 3—Comparison of clocked and interrupt-driven schedule.

changes on active trunks) that need processing per unit time usually precludes the use of an interrupt-driven schedule having one interrupt per stimulus. Figure 3 gives a qualitative description of the relative trade-offs between clocked and interrupt-driven schedules. For a more detailed, quantitative treatment of this trade-off, see, for example, Ref. 1. For a discussion of performance trade-offs of other scheduling alternatives, see Ref. 2, where this example system is used to provide a tutorial on the analysis and design of processor schedules for computer systems with real-time applications.

Considerable work on analyzing clocked schedules similar to the basic one described above, including applications to distributed microprocessor-based systems, has been done by P. Kuehn and others (e.g., see Refs. 3, 4, and 5). Here we consider a generalized clocked schedule that forms the basis for the processor scheduling in many real-time systems, including microprocessor-based systems as well as large single- and multiprocessor call-processing systems. In Section II we define the class of schedules to be studied, discuss relevant performance measures and work-load characterizations, and use concrete examples to illustrate some scheduling variants to this class of schedules. In Section III we develop approximations for certain performance measures introduced in Section II. The approximation method is based on that given in Ref. 6. In Section IV we illustrate the accuracy of the approximation by using comparisons with exact calculations (using the methods described in Ref. 7), simulation, and field data. Section V contains some concluding remarks.

## II. A GENERAL CLASS OF CLOCKED SCHEDULES

The simple clocked schedule noted above can be generalized in a variety of ways. We will consider some of the most important generalizations from an applications point of view and also note some additional variants of practical importance.

### 2.1 *Generalized clocked schedule*

Figure 4 shows the general class of clocked schedules that we will consider. Time is divided into intervals (slots) of length $T$. The first (labeling) column lists, in order of descending priority, all potential tasks for scheduling. A 1 in the $i$th row and $j$th column indicates that the $i$th listed task is scheduled for execution in the $j$th slot. We assume that every task is scheduled periodically and that the total period of the schedule is $N_{\text{period}}$. Three classes of tasks are indicated on Figure (4): High Priority (HP), Low Priority (LP), and Fill (F). The distinction between these will become clear in the following discussion.

The execution of the schedule proceeds as follows. We begin at time $t = 0$ at the beginning of the first slot. The tasks scheduled are executed in the order that they appear. Assuming all HP and LP tasks are completed before the end of the time slot ($t < T$), F work (assumed scheduled in every slot) is begun. If an LP task is still executing at the end of the time slot, it is interrupted, and it and all lower-priority LP tasks are added to the work list of the next slot *at their same priorities.* The HP tasks are *not* interrupted at the end of the time slot, that is, all HP tasks scheduled are completed before new work is scheduled. We will thus often refer to HP tasks as noninterruptable tasks and LP tasks as interruptable tasks. (F work is also interruptable.)

Thus, in summary, we have the following execution pattern. When

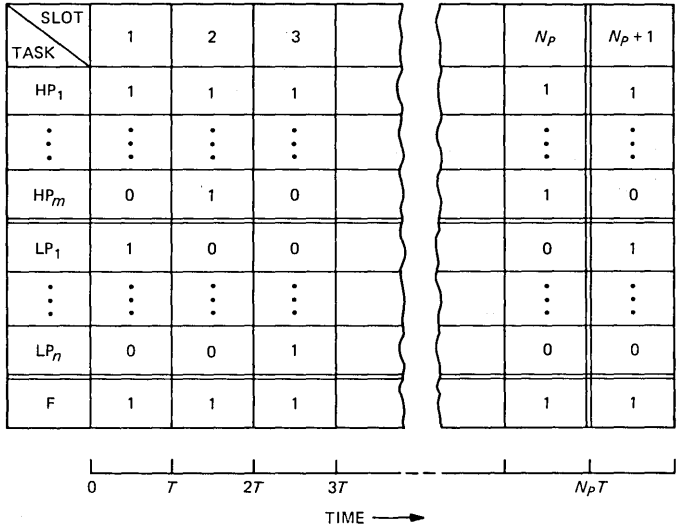



Fig. 4—General clocked schedule.

the HP list of tasks for the $j$th slot is begun (possibly delayed due to past scheduled HP work), the interrupt timer is inhibited. At the completion of all HP tasks, the timer is enabled. If the current time slot has already expired, the interrupt timer will immediately fire (scheduling new HP tasks); otherwise the LP work is begun. If an interrupt occurs prior to completion of all LP tasks, the remaining work is added to the next slot (the remaining rows in the $j$th column are "or'ed" to the $(j + 1)$st column). If all LP work completes prior to the interrupt, F work is executed until the interrupt occurs.

### 2.2 Performance measures

For a given task, scheduled every $\tau$ time units, we distinguish three categories of performance measures: short-term delays—delays on the order $T$ or less; medium-term delays—delays greater than $T$ but less than $\tau$; and long-term delays—delays greater than $\tau$. While this categorization is somewhat arbitrary, it has been useful in practice.

HP tasks are generally reserved for executing the most time-critical functions (e.g., digit reception in our example), and hence short-term delays are the most important performance measures. Indeed, short-term delays generally are needed to determine an appropriate time-slot length, $T$. A particularly important performance measure in this category is the probability that HP work scheduled for a given time slot is still executing at the end of that slot. This measure is generally referred to as the probability of (slot) overrun. Medium- and long-term delays also are often important for HP tasks; the latter is often needed to set appropriate levels for sanity timers. (In most systems, timers are set to monitor the time between successive executions of each task. If this time exceeds a given threshold [set to minimize the probability of false alarms due to work-load variability], certain maintenance routines are invoked.)

Long-term and sometimes medium-term delays are usually the most important performance measures for LP tasks, which generally are less time critical. For example, receiver attachment is typically scheduled as an LP task, possible every 100 ms, but with delays up to 500 ms acceptable.

Typical F tasks are line scanning for new originations or maintenance. For such tasks one defines a quantity $T_f$ (e.g., time to scan all lines once and time to complete a set of maintenance tasks) such that the relevant performance measures are based on delays in completing $T_f$ time units of F work. (Sometimes F work consists of executing a new schedule, in which case distributional information for delays can be an input to another performance analysis.) The delays of interest

for F work are generally long term, much greater than $T$, and often greater than $N_{\text{period}}\ T$, the entire schedule period.

### *2.3 Work-load characterization*

Two main aspects of work-load characterization must be considered: the job-processing times associated with task execution, which generally depends on the number of jobs (stimuli) to be processed; and the characteristics of the arrival of jobs (stimuli). If we denote by $X_i$ the (random) amount of time required to process task $i$, then often $X_i$ is adequately characterized by $a_i + b_i J_i$, where $a_i$ is the overhead associated with entering task $i$, $J_i$ is the (random) number of jobs found (in our example, the number of trunks that had state changes to be processed), and $b_i$ is the time needed to process each job found. To discuss the characterization of the arrival process, we use the queueing model description of a clocked schedule given in Fig. 5. There are two queues for each HP or LP task, an internal queue and an external queue. There is also a "never empty" single queue for F work. The arrival rate of jobs for the $i$th task is denoted by $\lambda_i$—often (we note some important exceptions shortly) the assumption of independent Poisson streams for these external arrivals is adequate. These external arrivals enter the external holding queue and are only allowed to pass to the internal queues, where they can be processed, when the indicated gates are opened. We distinguish four main arrival characterizations based on the gate-closing mechanism.

Gating 1: Here, at the start of each new time-slot execution, all gates corresponding to tasks scheduled for execution in this time slot are instantaneously opened, moving all existing jobs from the holding queues to the internal queues. The gates are then immediately closed, barring further movement of new jobs into the internal queues.

Gating 2: At the start of execution of a new slot, the gate for the highest-priority scheduled task is opened instantaneously and then

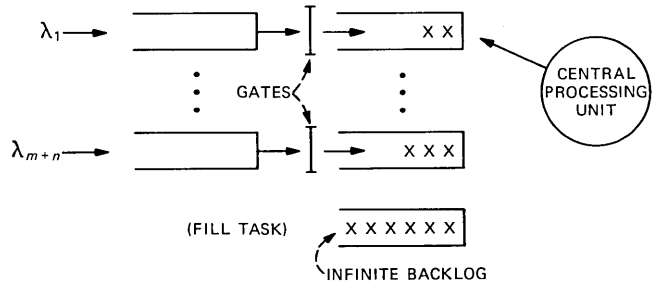



Fig. 5—Queueing model of clocked schedule.

closed. As each new scheduled task is ready for execution, its gate is correspondingly opened instantaneously and then closed.

Gating 3: Same as gating 2, but once opened, each gate is left opened until there are no more jobs for this task; the gates are then closed and the gate for the next task is opened.

Gating 4: All gates are always open.

Although in any given case, one of these gating mechanisms can often be used to characterize the arrivals, in actual systems, quite often a mixture of all are present, and indeed, there are further complications, e.g., the $\lambda_i$ are not independent. (Even in our example, clearly a large number of state changes in a given slot implies a high probability of a large number of state changes in a slot that is displaced in time by a dial pulse length.) We will note some examples as we discuss some variants of the class of clocked schedules we have defined for study.

### *2.4 Some variants and examples*

We briefly note some examples of systems that use clocked schedules as a means of demonstrating some of the complexities of clocked schedules for real systems and how they often deviate from the "ideal" models noted above. However, practical experience has indicated that these somewhat idealized models can often provide valuable insight into system performance.

### *2.5 Microprocessor peripheral interface system*

An example of a rather basic clocked schedule is that used in the Microprocessor Peripheral Interface System (MPIS), designed to provide an intelligent interface between certain switching systems and T-carrier facilities. The main purpose of MPIS is to execute the tasks associated with the setup and tearing down of interoffice calls via a T-carrier facility. This includes digit reception/transmission/timing and interoffice signaling. For this system, the schedule for each slot is identical. After some overhead associated with each new time slot, the first task executed is to poll each of the T-carrier digroups and to process signaling change reports from or orders to them. With this polling mechanism, some work arriving after the task is initiated will be processed, while some will not. Hence the arrivals to this task behave like a combination of gating 2 and gating 3. The next task consists of unloading a First-In First-Out (FIFO) queue with orders from the switching machine and, unless the order is a high-priority maintenance order, sorting and scheduling the orders for processing by other tasks later in the slot. Gating 1 is a reasonable approximation for characterizing the arrival pattern to these latter tasks. However,

there is clearly a dependence between the work associated with these tasks and the work done in the FIFO task.

### 2.6 Support processor

Large 1 *ESS*™ switching equipment offices have a Support Processor (SP), which performs most of the needed input/output and timing functions. Almost all of the tasks are executed as HP, LP, or F tasks consistent with the model of Fig. 4. Typical HP tasks include digit reception and transmission, LP tasks include timing functions, and both trunk and line scanning for new originations are done as F work. However, one low-priority (interruptable) task, Peripheral Order Buffer (POB) execution, is clearly at variance with Fig. 4. This task executes orders that control peripheral equipment, e.g., open (close) relays; thus, after completing execution, it must wait a minimum period of time (20 ms) for the controllers to free and reset. Hence, this task is executed asynchronously with the rest of the schedule being rescheduled precisely 20 ms after completing. Moreover, these orders can be loaded by the main processor at any time; hence gating 4 applies.

### 2.7 Private branch exchange

Clocked schedules are common in many Private Branch Exchanges (PBXs). Again, HP work usually involves digit reception and transmission, and LP work includes scanning for new originations and receiver attachment, while F work is devoted to maintenance tasks.

These variants, and many others in actual systems, cause the models discussed above to be approximate at best. Nonetheless, as noted, they can often supply useful performance information. We will use these examples to look at some concrete quantification in Section IV.

## III. APPROXIMATE ANALYSIS OF LONG-TERM DELAYS

For systems where short-term delays are an important performance measure, it is usually necessary to capture a considerable amount of scheduling detail (e.g., the dependencies and gating noted for MPIS in Section II). This problem has been addressed in Ref. 8, where exact solutions for realistic models are given. Here we concern ourselves with obtaining rather simple analytic approximations to long-term delays. In practice, the results have also been useful for medium- and sometimes short-term delay calculations.

### 3.1 Approximating task waiting time for the simplest system

Consider the simple clocked schedule depicted in Fig. 6. We assume

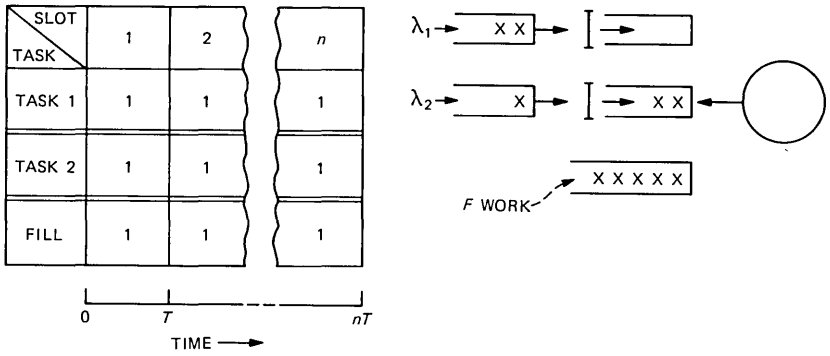



Fig. 6—Simplest clocked schedule.

that the arriving work associated with the $i$th task, $i = 1, 2$, forms sequences $\{X_1^k\}$, $\{X_2^k\}$ of independent, identically distributed random variables and that these sequences are mutually independent. We also assume that the work arrivals follow the gating 1 procedures described above. Denote the distribution functions for these work items by $F_i(x)$, i.e., $F_i(x) = P_r\{X_i \leq x\}$. Under these assumptions, the delays for task 1 can be obtained by studying a D/G/1 queue with (deterministic) interarrival time equal to $T$ and service-time distribution given by $F_1(x)$. Reference 8 gives a variety of approximation methods for estimating the delays in such a system. In particular, if we denote the waiting time by $t_1$, then Ref. 9 shows how to determine suitable parameters $a_1$, $C_1$ such that

$$W_1(x) = 1 - C_1 e^{-a_1 x} \tag{1}$$

is a good approximation to $P_r\{t_1 \leq x\}$. (We discuss some specific choices of $a_1$, $C_1$ in Section IV.) To study the delays for task 2, we first define a random walk, $\{Y^k; y\}$ by

$$\begin{aligned} Y^{k+1} &= Y^k - (T - X_1^{k+1}) = Y^k - Z^{k+1}, \qquad k \geq 0 \\ N(y) &= \min\left\{n: \sum_{k=1}^{n} Y^k < 0, \qquad \text{given } Y^0 = y\right\}, \end{aligned} \tag{2}$$

where $T$ is the time slot length.
Now define:

$\tau_N$ = time from arrival of an arbitrary batch of task 2 work until its processing begins (task 2 waiting time).

$w$ = work backlog (task 1 and task 2) just before the start of an arbitrary time slot.

$W_c(n) = P_r\{\tau_N > nT\}$. (Complementary task 1 waiting-time distribution).

$G_c(n; y) = P_r\{\mathrm{N}(y) > n\}$.

$H(x) = P_r\{w \leq x\}$.

We then have that

$$W_c(n) = \int_0^\infty G_c(n; x)dH(x), \qquad n \geq 1$$
$$W_c(0) = 1 - H(0)F_1(0). \tag{3}$$

That is, if in the random walk (2), $y$ is the backlog seen by an arriving task 2 batch of work, then $Z^k$ is the amount of time available in the $k$th slot to work off this backlog (or add to it if $Z^k < 0$), $Y^k$ is the remaining backlog at the end of the $k$th slot (start of the $(k+1)$st slot), and the new work for task 2 arriving at the beginning of slot 1 begins execution in the first slot that ends with $Y^k < 0$.

The usefulness of (3) depends on our ability to determine reasonable approximations for $G_c(n; x)$ and $H(x)$. First, we note that the work backlog, $w$, is precisely what would be seen by an arrival to a D/G/1 queue with service-time distribution equal to the convolution of $F_1$ and $F_2$, i.e., with service time $X_1 + X_2$. Again the approximation methods of Ref. 8 allow us to reasonably approximate $H(x)$ by using a simple exponential form, $\mathrm{H}'(x)$, i.e., with

$$H'(x) = W_2(x) = 1 - C_2 e^{-a_2 x}. \tag{4}$$

The central limit theorem implies that $G_c(n; x)$ is asymptotically Gaussian, for large $n$. We thus approximate $G_c(n; x)$ by $G_c'(n; x)$ defined by

$$G_c'(n; x) = \frac{1}{\sqrt{2\pi}} \int_{(n-m_N)/\sigma_N}^\infty e^{-y^2/2} dy, \tag{5}$$

where

$$m_N = \frac{x}{\overline{Z}}$$

$$\sigma_N^2 = \frac{x\sigma_z^2}{\overline{Z}^3}$$

and, as above, $Z = T - X_1$.

Equation (5) can be written as

$$G_c'(n; x) = \frac{1}{2}\left[1 - \mathrm{Erf}\left[\frac{A_1 n}{\sqrt{x}} - A_2\sqrt{x}\right]\right], \tag{6}$$

where

$$A_1 = \bar{Z}^{3/2}/\sqrt{2\sigma_Z^2}$$

$$A_2 = \bar{Z}^{1/2}/\sqrt{2\sigma_Z^2}$$

and

$$\text{Erf}(v) = \frac{2}{\sqrt{\pi}} \int_0^V e^{-s^2} ds.$$

Using (4) and (6) in (3) yields the approximation $W_c'(n)$ to $W_c(n)$

$$W_c'(n) = A_3 \exp(-A_4 n), \qquad n \geq 1$$

$$W_c'(0) = 1 - (1 - C_2)F_1(0), \tag{7}$$

where

$$A_3 = \frac{a_2 C_2}{2\sqrt{A_2^2 + a_2}[\sqrt{A_2^2 + a_2} - A_2]}$$

$$A_4 = 2A_1(\sqrt{A_2^2 + a_2} - A_2).$$

Equation (7) thus provides our desired approximation of the delay distribution for task 2. Note that (7) can readily be evaluated for $n$ nonintegral yielding an interpolation formula.

While the use of (5) might seem to imply that $n$ would need to be reasonably large for (7) to be useful, in fact, practical experience has indicated that (7) can provide a useful approximation, even for small $n$ (e.g., $n = 1$). This will be illustrated via some examples in Section IV. We will look at how other performance measures can be approximated, e.g., task sojourn time and delays for F work. First, we look at how we can use this method to obtain approximate delays for the more general clocked schedule of Fig. 4.

### *3.2 Approximating low-priority task waiting time for a general clocked schedule*

To treat the generalized clock schedule introduced in Section II, we need to extend the above approximation method to include an arbitrary number of tasks in a given time slot and, more important, to include more complex schedules. The former can be accounted for readily in the following manner. Assume for now that all slots have an identical schedule and that $\text{LP}_j$ is the task to be analyzed. We then replace all tasks of higher priority than $\text{LP}_j$ by a single task with an equivalent work load, i.e., we make the transformation

$$\sum_{i=1}^{m} \text{HP}_i + \sum_{i=1}^{j-1} \text{LP}_i \rightarrow \text{task 1}$$

$$\text{LP}_j \rightarrow \text{task 2}$$

and use the above analysis. Two methods of obtaining the needed aggregated work-load characterization for task 1 have been found useful. The simplest method is to match the relevant means and variances. For example, if the work loads take the form $a_i + b_i J_i$ for the $i$th task, and $\lambda_i$ is assumed Poisson, then we assume that the work load for the aggregated task, task 1, has the form $a + bJ$, where

$$a = \sum a_i$$

$$b\bar{J} = \sum b_i \bar{J}_i$$

$$b^2\bar{J} = \sum b^2\bar{J}_i.$$

Another alternative that is often preferable (particularly if the dominant work load is the result of a small number of jobs with large processing times) is to match the mean work load and the probability that there is no time available in a slot to execute task 2 work. That is, in addition to $a = \sum a_i$, $b\bar{J} = \sum b_i\bar{J}$, we must require that

$$P_r\{a + bJ > T\} = P_r\{\sum a_i + \sum b_i J_i > T\} = P^*.$$

The methods of Ref. 8 can be used to obtain $P^*$. A simpler method for estimating $P^*$ that has proved useful is to use a "clustering" technique to replace the potentially large number of tasks that must be aggregated by precisely three, i.e.,

$$\sum a_i + \sum b_i J_i \rightarrow a_{,1} + b_{,1}J_{,1} + b_{,2}J_{,2},$$

where the $\sum a_i$ *and* tasks with small values of $b_i$ map into $a_{,1}$, tasks with medium values of $b_i$ map into $b_{,1}J_{,1}$, and tasks with large values of $b_i$ map into $b_{,2}J_{,2}$. The resulting estimate of $P^*$ can then readily be computed. This also provides a simple approximation technique for the probability of overrun, another important performance measure noted earlier.

We now consider a more complex schedule. Let $N_{\text{period},j}$ be the scheduling period for $\text{LP}_j$. We assume that $N_{\text{period},j}$ divides $N_{\text{period}}$, the period of the total schedule. We introduce a new clocked schedule with slot length $T_{,j} = N_{\text{period},j}\, T$, and a task 1 that is a suitable average of all tasks of higher priority than task $j$ in all slots, i.e., let $K_j = N_{\text{period}}/N_{\text{period},j}$, and let $I_{\text{HP},k}$ and $I_{\text{LP},k}$ be the collection of indices for tasks of higher priority than task $j$ in the $k$th scheduling interval, $k = 1, \ldots, K_j$. Then

$$\underset{k=1,\ldots,K_j}{\text{Average}} \left\{ \sum_{i \in I_{\text{HP},k}} \text{HP}_i + \sum_{i \in IP,k} \text{LP}_i \right\} \rightarrow \text{task 1}$$

$$\text{LP}_j \quad \rightarrow \text{task 2},$$

and we again use the above analysis. (Aggregation methods similar to those discussed above can be used here.)

### 3.3 Other performance measures

The method outlined above can be extended to obtain useful approximations to other performance measures. We first look at the conditional sojourn time of a task. For simplicity of notation, we consider the simple system of Fig. 6. The work-load aggregation given above can also be used here for more complex schedules.

We now define:

$\tau_S(y)$ = time from arrival of an arbitrary batch of task 2 work until completion of $y$ units of work on that batch.

$S_c(n; y) = P_r\{\tau_S(y) > nT\}$ (Complementary conditional sojourn time).

We then have that

$$S_c(n; y) = \int_0^\infty G_c(n; x + y)dH(x). \tag{8}$$

Thus if $y = 0$, then $S_c(n; 0) = W_c(n)$.

Using (4) and (6) in (8), we obtain a rather complex integral involving the error function Erf$(\cdot)$ that does not seem to have a closed form representation in terms of Erf$(\cdot)$. Since our objective is to obtain, where possible, simple analytic approximations (exact numerical methods are available in Ref. 7), in addition to the Gaussian assumptions above, we assume the following (approximate) decomposition, $S_c'(n; y)$, of $S_c(n; y)$:

$$S_c'(n; y) = \int W_c'(n - m)dG'(m; y), \tag{9}$$

where

$$G'(m; y) = \frac{1}{\sqrt{2\pi}\sigma_{N_y}} \exp\left[-\frac{1}{2}\left[\frac{x - \bar{N}_y}{\sigma_{N_y}}\right]^2\right]$$

and

$$\bar{N}_y = \frac{y}{\bar{Z}}$$

$$\sigma_{N_y}^2 = \frac{y\sigma_z^2}{\bar{Z}^3},$$

i.e., $G'(m; y)$ is a (continuous in $m$) Gaussian approximation to $P_r\{N(y) = m\}$. This results in the following expression for $S_c'(n, y)$:

$$S_c'(n, y) = \frac{1}{2} W_c(n) \left[1\text{-Erf}\left[\frac{\bar{N}_y}{\sqrt{2}\sigma_{N_y}}\right]\right]$$
$$+ \frac{1}{2} A_3 \exp\left[A_4(\bar{N}_y - n) + \frac{A_4^2\sigma_{N_y}^2}{2}\right]$$
$$\cdot\left[\text{Erf}\left[\frac{\sqrt{2}A_4\sigma_{N_y}}{2} + \frac{\bar{N}_y}{\sqrt{2}\sigma_{N_y}}\right]\right.$$
$$\left. - \text{Erf}\left[\frac{\sqrt{2}A_4\sigma_{N_y}}{2} + \frac{\bar{N}_y}{\sqrt{2}\sigma_{N_y}} - \frac{n}{\sqrt{2}\sigma_{N_y}}\right]\right]$$
$$+ \frac{1}{2}\left[1 - \text{Erf}\left[\frac{n - \bar{N}_y}{\sqrt{2}\sigma_N}\right]\right] \tag{10}$$

and the $A_i$'s are as before.

While (10) may seem complicated, it involves nothing more complex than an error function, Erf($\cdot$), whose efficient computation is available in most computer system libraries.

Equation (10) can be used to obtain approximations for a variety of important performance measures. For example, the time from arrival to completion of a batch of task 2 work is given by

$$S_c'(n) = \int_0^\infty S_c'(n; y) dF_2(y). \tag{11}$$

For the common case where $X_2$ is composed of individual jobs, we can use (10) to compute the average sojourn time seen by a job. Also, if, as a fill task, we schedule, say, $T_f$ of maintenance as the next fill job starting at some time point, then $S_c'(n; T_f)$ provides the delay in finishing this task (with task 1 including *all* scheduled [nonfill] tasks).

Often, F work is repetitive, e.g., continuously scanning a set of lines. In such a case, there is no "waiting time" for fill work, i.e., by its definition, it starts immediately after completing. In this case, the elapsed time until all fill work is done can be approximated simply by $G_c'(n; T_S)$, where $T_S$ is the time to scan all lines.

Finally, we note that in the gating model "gating 1," jobs are additionally delayed as they wait in the outer queue. Since this delay is independent of the delays internal to this system, performance measures including this delay can be computed by simply convolving this external delay with the internal job delays.

## IV. EXAMPLES AND VALIDATION

The approximation method we have introduced is relatively simple, considering the complexity of the problem. Its usefulness, of course,

depends on how well it can capture the essence of the delay characteristics. We discuss here some applications and comparisons with: (1) exact results for the general class of clocked schedules introduced in Section II (using gating 1), (2) detailed simulation results that include all of the inherent work-load dependencies in an actual system, and (3) a comparison with some actual field data.

The approximation results were computed using a prototype software package, PACS (Performance Analysis of Clocked Schedules). This explicit implementation of our approximations determines $C_i$ and $a_i$ in the following manner. First, the service-time distribution in a D/G/1 queue is replaced by an appropriate hyperexponential, exponential, Erlangian, deterministic distribution that matches the first two moments of the service time. Then the approximation referred to as the $A^*$ approximation in Ref. 9 is used to determine $C_i$ and $a_i$. (For extreme light or heavy loads the approximations $A_{LT}, A_{H,0}$ of Ref. 9 are used.)

The first two examples considered use a mean and variance match of the aggregated work loads to a Gaussian distribution (see below). The third example used the clustering method noted above, while the fourth example again used a mean and variance match.

Example 1: As a first example we consider the simple schedule of Fig. 6 with the following parameter values:

$T$ (Time slot length) = 10 ms
$\lambda_i$ (Poisson arrival rate of jobs for task $i$) = 0.8/10 ms = 0.08/ms
$b_1$ (Time to process each job for task 1) = 5.0 ms
$b_2$ (Time to process each job for task 2) = 4.9 ms.

The resulting approximation for the backlog (eq. [4]) is

$W_2(x) = 1 - C_2 e^{-a_2 x} = 1 - 0.554 e^{0.0760x}$ the variable $Z = T - X_1 =$ 10 ms − 5.0 ms ($J_1$), where $J_1$ = number of jobs found for task 1. For eqs. (5) and (10) we thus have

$\bar{Z} = 6.0$ ms
$\sigma_Z^2 = 20.0$ ms$^2$.

The resulting waiting time (eq. [5]) and sojourn time (eq. [10]) for task 2 is shown in Fig. 7, where it is compared with the exact results obtained by the methods described in Ref. 7. Even for this simple example, the exact equilibrium solution is quite complex, being quasi-periodic in nature. Our simple method is seen to provide reasonable approximations to the waiting and sojourn time.

Example 2: Here we consider the more complex schedule shown in Fig. 8 with the parameters indicated. We look at the sojourn-time distribution for task 7. The work-load aggregation discussed above thus leads to $K = 2$, $T_{sj} = 40$ ms. Task 1 work is characterized by the

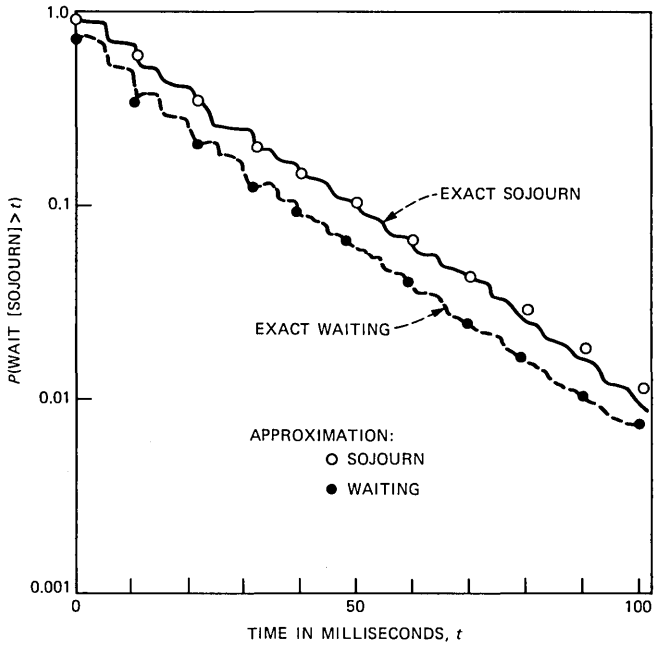



Fig. 7—Waiting and sojourn times for Example 1.

| SLOT / TASK | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 |
| 3 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 0 | 1 |
| 5 | 1 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 0 |
| 7 | 0 | 1 | 0 | 1 |
| F | 1 | 1 | 1 | 1 |

0 20 40 60 80
TIME IN MILLISECONDS

| $\lambda_i$ | $b_i$ |
|---|---|
| 0.050 | 2.0 |
| 0.050 | 2.0 |
| 0.075 | 2.0 |
| 0.050 | 3.0 |
| 0.050 | 3.0 |
| 0.063 | 2.0 |
| 0.050 | 3.0 |

Fig. 8—Schedule for Example 2.

mean and variance of the task $i$ ($i = 1, \ldots, 6$) work falling in each of two slots (of length 20 ms.) Figure 9 shows the resulting approximation for the sojourn-time distribution again compared with the exact results obtained from the methods of Ref. 7.

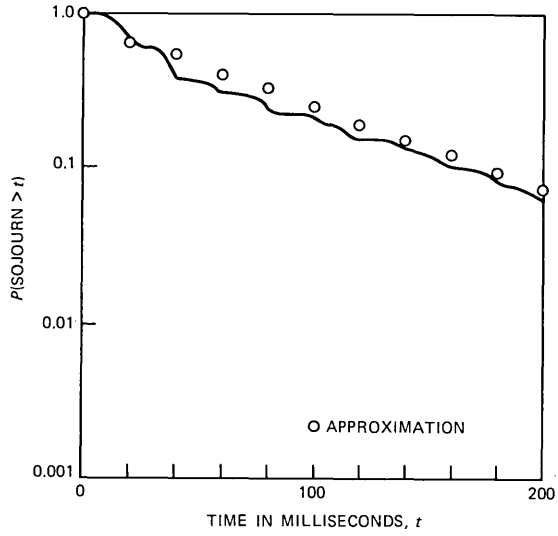



Fig. 9—Sojourn time for Example 2.

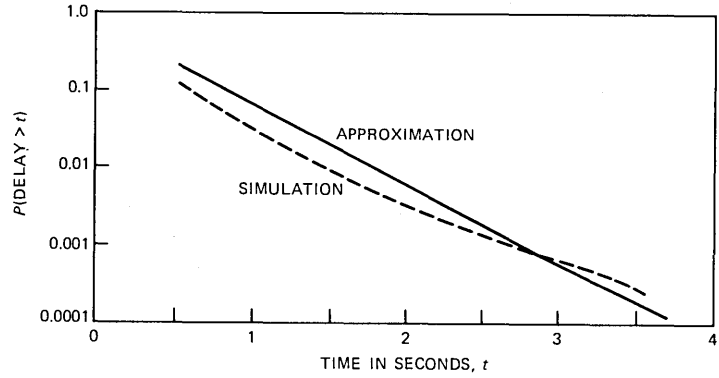



Fig. 10—Receiver attachment delay for Example 3.

Example 3: As noted in Section II, one of the LP tasks for the PBX discussed is the attachment of a digit receiver for a new line origination. Figure 10 shows the delays in receiver attachment as computed from our approximation method compared with the results of a detailed simulation model. This is a call-by-call simulation that includes most of the work-load dependencies associated with the actual system. We again see quite reasonable agreement.

Example 4: In the SP noted earlier, line scanning is a fill task. Figure 11 shows the line-scan delays predicted by our approximation as compared with data collected at a field site. We see that the approximation results fall well within the observed band of field data.

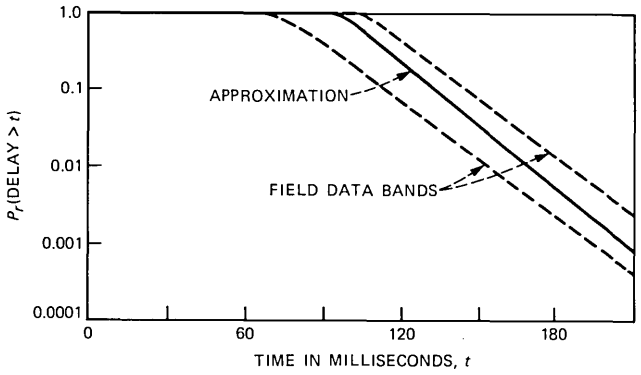



Fig. 11—Line-scan delay for Example 4.

## V. CONCLUDING REMARKS

The approximation methods presented here, while quite simple in nature, have been used in the performance optimization of many systems. Often, these relatively easy-to-compute approximations are used in the preliminary design phase to determine several feasible candidate schedules, as well as to identify performance issues needing closer study. The more precise methods of Refs. 7 and 8 then have been used to address these issues. The final one or two potential schedules generally are then evaluated using detailed call-by-call simulation. In the case of modifications to existing systems, field trials are conducted for validation. For new systems, predicted performance is verified in laboratory models and early field introduction as needed. This has helped bring high-performance, high-capacity systems to fruition.

## REFERENCES

1. D. R. Manfield and P. Tran-Gia, "Queuing Analysis of Scheduled Communications Phases in Distributed Processing Systems," *Performance 81, Proc. 8th Int. Symp. Comp. Perf. Mod., Meas., and Eval.*, Amsterdam, November 4–6, 1981, F. J. Kylstra, ed., New York: North-Holland, pp. 233–50.
2. A. A. Fredericks, "Analysis and Design of Processor Schedules for Real Time Applications," in *Applied Probability-Computer Science: The Interface*, Vol. 1, R. L. Disney and T. J. Ott, eds., New York: Birkhauser, 1982, pp. 433–50.
3. P. J. Kuehn, "Analysis of Switching System Control Structures by Decomposition," Proc. 9th ITC, Spain, 1979.
4. P. Tran-Gia and H. Jans, "Delay Analysis of Clocked Event Transfer in Distributed Processing Systems," ORSA/TIMS, Detroit, April 19–21, 1982.
5. H. Weisschuh, "Development of the Control Software for a Stored Program Controlled PCM Switching System," 24th Report on Studies in Congestion Theory, Univ. of Stuttgart, 1977.
6. A. A. Fredericks, "Analysis of a Class of Schedules for Computer Systems With Real Time Applications," *Performance of Computer Systems*, M. Arato, A. Butrimenko, E. Gelenbe, eds., New York: North-Holland, 1979, pp. 201–16.
7. M. H. Ackroyd, "Numerical Computation of Delays in Clocked Schedules," AT&T Technical Journal, this issue.

8. B. T. Doshi, "Analysis of Clocked Schedules—High-Priority Tasks," AT&T Technical Journal, this issue.
9. A. A. Fredericks, "A Class of Approximations for the Waiting Time Distribution in a G/G/1 Queueing System," B.S.T.J., *61,* No. 3 (March 1982), pp. 295–325.

## AUTHORS

**Darlene F. DeMaio,** B.A. (Mathematics), 1976, Lycoming College; M.S. (Industrial and Applied Mathematics), 1981, Polytechnic Institute of New York; AT&T Bell Laboratories, 1976—. Ms. DeMaio has been involved in studying the performance of computer operating systems using both analytic and simulation models. Her current interest is in the development of user friendly performance analysis software.

**Brian L. Farrell,** B.S. (Mathematics, Computer Science), 1978, St. Peter's College; M.S. (Operations Research), 1982, Columbia University; AT&T Bell Laboratories, 1978—. Mr. Farrell has worked on projects involving the construction of simulation models for real-time computer systems. He has also worked on a software package for the analysis of clocked schedules and more recently has been involved in performance analysis and capacity estimation for operations systems.

**Albert A. Fredericks,** B.S. (Mathematics), 1962, Fairleigh Dickinson University; M.S., Ph.D. (Mathematics), 1965 and 1970, respectively, Courant Institute, New York University; AT&T Bell Laboratories, 1961—. Mr. Fredericks is Head of the Performance Analysis Department. His responsibilities include the development of methods for analyzing the performance of computer/communications systems and manufacturing systems.

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Numerical Computation of Delays in Clocked Schedules

By M. H. ACKROYD*

(Manuscript received December 8, 1983)

A discrete-time model for clocked schedules is described. The distributions of the waiting and sojourn times for tasks in this model can be computed by a two-stage process. The first stage involves computing the distribution of the work awaiting execution at each time slot in the schedule. The second stage yields the required delay distributions. The computations in both stages involve discrete convolutions, which can be computed via the fast Fourier transform. The availability of an exact method of analysis permits the detailed study of schedules. It also enables the accuracy of approximate methods of analysis, such as those in a companion paper by Fredericks et al., to be determined. The present paper considers just one form of scheduling mechanism; a companion paper by Doshi considers several other mechanisms.

## I. INTRODUCTION

Clocked schedules provide a means for organizing and controlling the execution of tasks in software for switching systems or other systems having strict timing requirements.[1,2] This paper describes a method for computing schedules. The method is based on the use of a discrete-time model, which makes it possible to compute the required distributions exactly, except for the effect of computational error. The availability of an exact method of analysis has some advantages, even though it can be somewhat expensive in terms of the amount of computation needed. An exact method permits the detailed study of

* AT&T Bell Laboratories.

the behavior of clocked schedules and the comparison of schedules that differ only in small details. In addition, it can be used to check the accuracy of more economical, approximate methods of analysis.

The next section of the paper describes the discrete-time model for clocked schedules that allow the methods presented here. Section III explains the method for the numerical computation of the required delay distributions. Section IV describes some aspects of an implementation of the method, and Section V gives some illustrative results computed by its use.

## II. CLOCKED SCHEDULE MODEL

There are two aspects of the model that need to be described. First, we need to describe the discipline under which the execution of the different software tasks, each having its own priority, is organized and controlled. Second, we need to specify the statistical characteristics of the execution times of the tasks.

### *2.1 Organization and control of task execution*

Figure 1 illustrates how task execution is organized. There are $N_{\text{task}}$ tasks, each with a unique priority. The priorities are numbered from

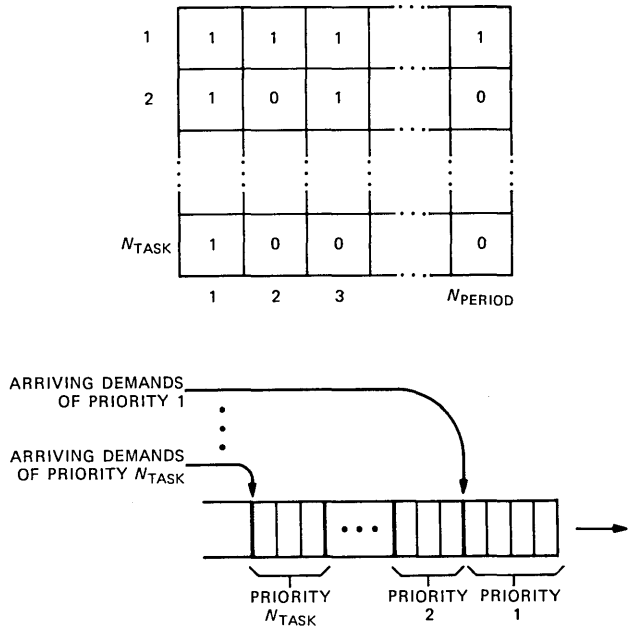



Fig. 1—Schedule table and demands for the execution of tasks waiting in a preempt resume head-of-line priority queue.

one, top priority, to $N_{\text{task}}$, lowest priority. The schedule is specified by a table whose elements are ones and zeros. The table has $N_{\text{period}}$ columns and is considered to be repeated periodically. Each tick of the system clock, at $t = 0, T, 2T, \cdots$, corresponds to one column of the schedule table, and each task corresponds to one row of the table. The periods between consecutive ticks are referred to as *slots*.

When a clock tick occurs, the system consults the schedule table to determine which tasks have become due for execution. The tasks due for execution are indicated by the presence of ones in the positions corresponding to these tasks and the current clock tick. The system instantly places demands for the execution of the newly scheduled tasks in a queue. This queue, which is illustrated in Fig. 1, is a head-of-line priority queue.[3] In the queue, demands for task execution are grouped in order of task priority, the group of demands for the highest-priority task being at the front of the queue.

All the tasks are considered, in this paper, to be of interruptable type,[1] i.e., if a task of priority $i$ is still being executed when a clock tick occurs, its execution is instantly interrupted, rather than running on to completion, through the occurrence of the clock tick. The demand for the execution of the task is put back in the queue at the head of the group of demands of priority $i$ until no demands of higher priority remain to be handled. The execution of the task of priority $i$ is then resumed where it was left off, with no overhead (extra execution time) involved in the resumption. The queue is of the head-of-line, preempt resume type.

### 2.2 Task execution times

We assume that the time required to execute a task, given that it has exclusive use of the processor, is an independent, discrete random variable. This key assumption makes possible the numerical method of solution described in the next section. Task execution times are of the form $k\Delta$, where $k$ is a random integer and where the interval $\Delta$ is an integer submultiple of the clock period $T/\Delta = N_{\text{subdv}}$. The execution time of each task is characterized by the distribution of the random integer $k$. The method of this paper applies to arbitrary distributions for $k$, which need to be available in numerical form.

If experimental information about the task execution times is available, e.g., as histograms, these data can be used directly to provide the numerical input required by the analysis. Alternatively, the numerical input required can be provided by evaluating a formula that provides a model for the execution times of the tasks. One such model, used in Ref. 2, is based on the assumption that the task execution time consists of an initial overhead $a$, followed by $n$ executions of a job, where $n$ is Poisson distributed and where each execution of the job requires time

$b$. Provided that $a$ and $b$ are integer multiples of $\Delta$, this model fits the form assumed in this paper. Examples of the use of this model are given in Section V.

### *2.3 Performance measures*

Various measures of the performance of a system controlled by a clocked schedule may be of interest. Here we deal with two: the distribution of the waiting time of a task and the distribution of its sojourn time. The waiting time for a task is the time that elapses from the instant it is scheduled until its execution starts. The sojourn time is the time from the instant a task is scheduled until its execution is completed.

As mentioned before, the execution of a task may be considered to consist of a number of repeated executions of a single job. We do not, in this paper, deal with the waiting and sojourn times for the individual jobs that, in a batch, constitute the task. Of course, the waiting time for a task provides a lower bound on the waiting time for a job within the task. Similarly, the sojourn time for a task provides an upper bound on the sojourn time for a job.

In this paper we are concerned with the behavior of systems in equilibrium. Nonetheless, the delay distributions will generally be time dependent. Consider, for example, a task of low priority that is scheduled at several slots. The delays for the low-priority task will generally be larger at slots where much work of higher priority is also scheduled than at slots where little work of higher priority is scheduled.

In assessing the performance of a schedule to determine whether it meets a specification, the average of the delay distributions of a task will usually be of prime interest—the average being computed over all slots where the task is scheduled. Sometimes, though, the individual delay distributions will be of interest, such as when a schedule is being analyzed in detail in order to improve it. In either case, the approach described in the next section involves computing the delay distributions for a task at each slot where it is scheduled. The average distribution is obtained by averaging the individual distributions.

## III. COMPUTATION OF DELAY DISTRIBUTIONS

As shown in the previous section, a system controlled by a clocked schedule can be considered as a nonstationary multiclass priority queueing system. The analytical solution of such a system would be difficult. However, because of the form assumed for the distributions of task execution time, numerical results can readily be obtained. In this section we detail the computation of the waiting-time distribution. The computation of the sojourn distribution is very similar, so it is mentioned only briefly.

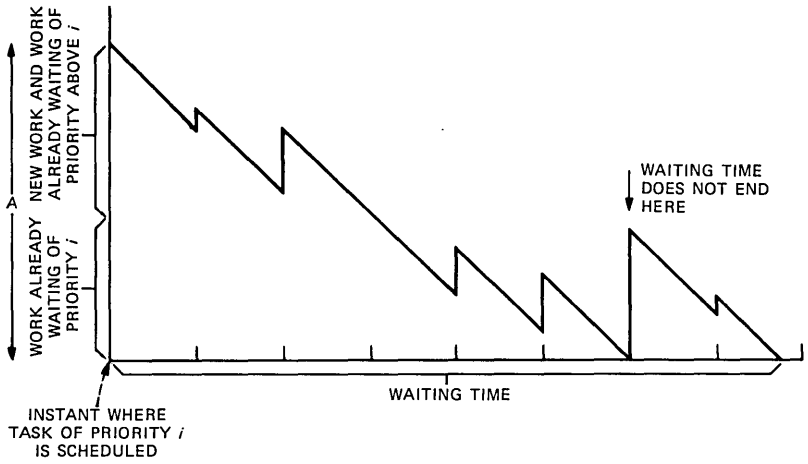



Fig. 2—Relationship between task waiting time and the work waiting in the queue.

We consider a task of priority $i$. At the instant the task is scheduled to be executed, the demand for its execution joins the queue, behind any previously scheduled demands for execution of the task still waiting to be handled. Its waiting time consists of the sum of two components, as illustrated in Fig. 2. The two parts of the waiting time are as follows:

1. The time required to clear any backlog of work, also of priority $i$, which was already waiting at the instant the task was scheduled. The wait of the task is unaffected by any arrivals of priority $i$ after its own, so these do not need to be considered.
2. The time required to clear the backlog of work, if any, of priority higher than $i$ that was already waiting at the instant it has scheduled plus the time required to deal with any work, of priority higher than $i$, that is scheduled during its wait.

We now elaborate on our definition of waiting time. The waiting time of a task is considered to have ended when the waiting work taking precedence over it becomes zero and then remains zero for a finite period. The waiting time is not considered to have ended if the waiting higher-priority work merely becomes zero for an instant—an event that can have finite probability, with the discrete distributions considered here. An example of such an event is illustrated in Fig. 2.

The situation is quite similar for sojourn times. The sojourn time of a task ends when the sum of the work in the queue taking precedence over the task plus the work remaining to execute the task itself becomes zero. In contrast to our definition of the waiting time, the sojourn is considered to have ended if this sum becomes zero for even just an instant.

The distribution of the waiting time for a task of priority $i$ can be computed in two stages. The first stage involves computing the distribution of the work, of priority $i$ and higher, awaiting execution immediately after the task is scheduled. This work is indicated by the time 'A' in Fig. 2. The second stage involves computing the distribution of the waiting time itself from the results of the first stage. This is done by a recursive process in which the $l$th stage of the recursion yields the probability that the waiting time has duration $l\Delta$. The two stages are now described in more detail.

### *3.1 Computing the distribution of the work awaiting execution*

We need to compute the distribution of the work, of priority $i$ and higher, awaiting execution just after each clock tick. $y_n$, the work of priority $i$ and higher awaiting execution at $t = nT_+$, can be expressed in terms of $y_{n-1}$ and $x_n$, the new work joining the queue at $t = nT$. The equation is

$$y_n = (y_{n-1} - T)^+ + x_n , \tag{1}$$

where $(\cdot)^+ = \max(0, \cdot)$. This expresses the fact that the work awaiting execution is diminished by $T$ during a slot, but the waiting work cannot become negative.

The right side of (1) involves the addition of a pair of independent random variables. The distribution of the sum of independent variables is obtained by convolving their distributions, which leads to a formula for the distribution of $y_n$ in terms of the distribution of $y_{n-1}$:

$$f(n, i, k) = [\Psi_{k,N_{\text{subdv}}} f(n-1, i, k)] *_k q(n, i, k). \tag{2}$$

Terms used in (2) are defined below:

$f(n, i, k)$ = the probability that $y_n$, the work in the queue, of priority $i$ and higher, at $t = nT_+$, would require a total time $k\Delta$ for its execution. $i$, though indicated explicitly, is held constant throughout all computations.

$q(n, i, k)$ = the probability that $x_n$, the new work of priority $i$ and higher, scheduled for execution at $t = nT$, would require a total time $k\Delta$ for its execution.

$\Psi_{k,N_{\text{subdv}}}$ = the operation of shifting a distribution $N_{\text{subdv}}$ places left on the $k$ axis and then sweeping the probability on the negative $k$ axis up to the origin.[3] The shift corresponds to the subtraction of $T = \Delta N_{\text{subdv}}$ in (1), and the sweep corresponds to the fact that the waiting work can be reduced only to zero.

$*_k$ = the operation of convolution with respect to the discrete variable $k$, e.g., $u(k) *_k v(k) = \sum_{l=-\infty}^{\infty} u(l)v(k-l)$.

By starting with a known distribution $f(0, i, k)$, such as the distribution corresponding to an empty queue, (2) can be evaluated repeat-

edly to give the distributions $f(n, i, k)$ for successive values of $n$. The volume of computation needed would normally be quite unreasonable if (2) were evaluated directly. However, by using a fast Fourier transform algorithm,[3] we can reduce the amount of computation to reasonable proportions.

The distribution of the work arriving at the queue, $q(n, i, k)$ is, due to the periodicity of the schedule table, periodic in $n$. That is, $q(n, i, k) = q(n + N_{\text{period}}, i, k)$, for all $n$. Provided that the average work scheduled per clock tick is less than $T$, the queue will be stable, and $f(n, j, k)$ will converge to a periodic steady state. (Repeated evaluation of eq. [2] is analogous to a stable time-invariant linear system driven by a periodic input. After initial transients have decayed, only the periodic response remains.) On convergence, we have obtained the periodic steady-state distribution, $\hat{f}(m, i, k)$, for $m = 1, 2, \cdots, N_{\text{period}}$, i.e., for each slot in the schedule. Except in special cases, convergence requires an infinite number of iterations, so an appropriate criterion must be used, in practice, to determine when the iterations may be stopped.

### 3.2 Computation of the waiting-time distribution

The waiting-time distribution can be computed from the work distribution, $\hat{f}(m, i, k)$, by a recursive process. We require the computation of a sequence of probabilities $w(m, i, k)$, for $k = 0, 1, \cdots$, where $w(m, i, k)$ denotes the probability that a task of priority $i$, scheduled at slot $m$, waits a time period of $k\Delta$ until its execution starts. Equivalently, we require the computation of the probability that the work in the queue, having precedence over the task, first becomes zero at time $k\Delta$ after the task is scheduled (and remains zero for at least $\Delta$).

We define a sequence of conditional distributions $r(m, i, k, l)$, $l = 0, 1, \cdots$, where $r(m, i, k, l)$ is the probability that the work in the queue having precedence over the task of priority $i$ scheduled at slot $m$ is $k\Delta$, given that the waiting time is $l\Delta$ or more (for $l > 0$). For $l = 0$, $r(m, i, k, l)$ is the unconditional distribution of the waiting work that has precedence over the task of interest, at slot $m$. This work is the sum of the work, of priority $i$ and higher, left over from the previous slot, plus any new work, of priority higher than $i$, joining the queue at slot $m$. By convolving the distributions of the two parts of the sum, we obtain the required distribution:

$$r(m, i, k, 0) = [\Psi_{k, N_{\text{subdv}}} \hat{f}(m - 1, i, k)] *_k q(m, i - 1, k). \quad (3)$$

The convolution in (3) can be computed, as previously, by the use of a fast Fourier transform algorithm.

The probability that the waiting time is zero is given by the origin

point of the distribution computed from (3), i.e., $w(m, i, 0) = r(m, i, 0, 0)$.

We now consider the computation of the probability that the waiting time is $l\Delta$, for $l > 0$. $r(m, i, 0, l)$ is the probability that the work in the queue having precedence over the task of interest becomes zero at $l\Delta_+$ after the task is scheduled, given that the task's wait has not yet ended. The unconditional probability that this work becomes zero at $l\Delta_+$ after the task is scheduled at slot $m$ is given by

$$w(m, i, l) = \left\{ 1 - \sum_{j=0}^{l-1} w(m, i, j) \right\} r(m, i, 0, l), \tag{4}$$

i.e., the required probability value is obtained from the previously computed points in the waiting distribution and the $l = 0$ point on the conditional distribution $r(m, j, k, l)$.

To complete the $l$th stage of the recursion, the conditional distribution $r(m, j, k, l + 1)$ must be computed. It is the distribution of waiting work having precedence over the task of interest at $(l + 1)\Delta_+$ after the clock tick where the task was scheduled, given that the wait did not end at $l\Delta_+$ after the task was scheduled. (We consider $l\Delta_+$ because the wait is not considered to have ended if the waiting work becomes zero at $l\Delta_-$ and then immediately becomes nonzero again due to the arrival of new high-priority work.) $r(m, j, k, l + 1)$ is obtained by modifying $r(m, j, k, l)$ in three ways, as follows:

1. The $k = 0$ point is set to zero and the remaining points are normalized. This discounts the case where the waiting time is $l\Delta$.

2. The conditional distribution is shifted one place to the left on the $l$ axis to account for the reduction of the waiting work during an elapse of time $\Delta$.

3. The conditional distribution is convolved with the distribution of the new work, of priority higher than $i$, joining the queue at $(l + 1)\Delta$ after the task of interest was scheduled. If none is scheduled to join the queue or if $(l + 1)\Delta$ is not a clock instant, the conditional distribution is left unchanged.

The new conditional distribution is thus given by

$$r(m, i, k, l) = \left[ \Psi_{k,1} \left( \frac{r(m, i, k, l) - r(m, i, 0, l)\delta(k)}{1 - w(m, j, l)} \right) \right] *_k g(k), \tag{5}$$

where $\delta(k) = 1$, $k = 0$; $\delta(k) = 0$, otherwise, and where $g(k)$ is the distribution of arriving work having priority higher than $i$, i.e.,

$$g(k) = \begin{array}{l} q(m + l/N_{\text{period}}, i - 1, k), \text{ for } l \bmod N_{\text{period}} = 0 \\ \delta(k), \text{ otherwise.} \end{array}$$

The recursive scheme defined by (4) and (5) is started with $r(m, i, k, 0)$, given by (3).

The normalizing division by $1 - w(m, i, l)$ in (5) is followed, at the next stage of the recursion, by a corresponding multiplication by $1 - w(m, i, l)$ in (4). In the actual computations, the divisions in (5) and the corresponding multiplications in (4) can be omitted. The computed intermediate quantities then lose their interpretation as probabilities, of course.

Evaluation of (6) gives the waiting-time distribution for a task scheduled at the $m$th slot in the schedule. As mentioned before, for a task scheduled more than once per period of the schedule, the waiting-time distribution will generally vary with $m$. The computations involving (4) and (5) can be repeated, for each slot where the task is scheduled, to obtain the individual waiting distributions. When the average of the distributions is required, this can be obtained by summing the individual distributions and normalizing the sum.

### *3.3 Sojourn distribution*

The computation of the sojourn distribution is similar to the computation of the waiting distribution, but it differs in two ways. We are concerned with when the waiting work of priority $i$ and higher becomes zero, rather than the work whose priority is higher than $i$. Also, the sojourn ends even if this work becomes zero for just an instant.

To consider work of priority $i$ and higher, (3) is replaced by

$$r(m, i, k, 0) = [\Psi_{k,N_{\text{subdv}}} \hat{f}(m - 1, i, k)] *_k q(m, i, k)$$

or, equivalently from this equation and (2),

$$r(m, i, k, 0) = \hat{f}(m, i, k). \tag{6}$$

Equation (5) is used unchanged. So that we do not disregard cases where the waiting work becomes zero for just an instant, we use the conditional distribution of the waiting work immediately prior to clock instants, rather than immediately after. $r(m, i, k, l-1)$ is the conditional distribution of work at $t = (l - 1)\Delta_+$ after the task is scheduled, so $r(m, i, k + 1, l - 1)$ is the conditional distribution at $l\Delta_-$. Equation (6) is therefore replaced by

$$s(m, i, l) = \left\{\left[1 - \prod_{j=0}^{l-1} s(m, i, j)\right]\right\} r(m, i, 1, l - 1),$$

where $s(m, i, l)$ denotes the sojourn distribution for the task or priority $i$ scheduled at slot $m$.

## IV. IMPLEMENTATION

The method given in the previous section has been implemented as a computer program in which all the convolutions are computed via Fast Fourier Transform (FFT) routines.[4] The delay distributions for a task of priority $i$ are computed in four principal steps:

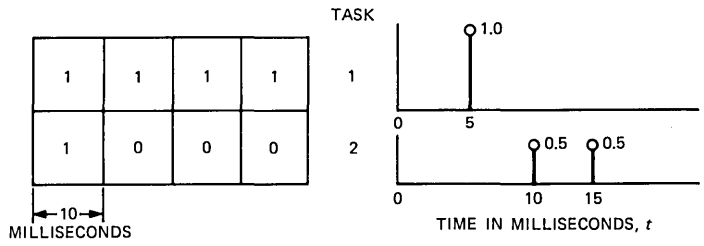
1. Initialization
2. Distribution of waiting work, via (2)
3. Waiting-time distributions, via (3, 4)
4. Sojourn-time distributions, via (6, 7).

The major part of the initialization step is the computation of the distributions $q(m, i, k)$ for each slot $m$ in the period of the schedule. This is done by convolving the individual task execution time distributions, as specified by the presence of ones in each column of the schedule table.

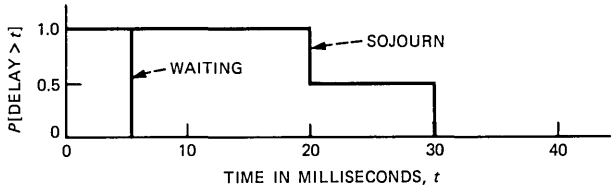
Convolutions performed via the FFT are cyclic convolutions of finite sequences, rather than the aperiodic convolutions of infinite sequences required here.[4] In principle, this introduces error, due to wraparound of the tails of the computed distributions. However, when a suitably large FFT block size is used, the magnitude of the tail extending beyond the end of the block is small by comparison with the error due to arithmetic rounding (or truncation). The effect of wraparound is then negligible, with no appreciable effect on the computed results.

The iterations in computing the distributions of the waiting work are computationally expensive. Of course, if the FFT were not used, the computation would be more expensive still—by factors of over one hundred, for large block sizes. To minimize the computational expense, only as many iterations should be used as necessary for a required level of accuracy. In using the method, the following criterion has been found useful. The computed cumulative distribution of the waiting work at the first slot in the schedule is compared with the corresponding distribution obtained $N_{\text{period}}$ iterations previously. In equilibrium, these cumulative distributions would be identical. The iterations are stopped when the computed cumulative distributions agree completely, in element-by-element comparison, to $N_{\text{place}}$ decimal places, $N_{\text{place}}$ being specified by the user. Experience shows that this is a reasonably satisfactory rule for halting the iterations. Experience with examples such as the one mentioned below suggests that the use of this criterion usually results in the computed cumulative distributions of waiting and sojourn times also having an accuracy of about $N_{\text{place}}$ decimal places in each point. If a large value of $N_{\text{place}}$ is specified, the desired accuracy may be unachievable, due to the effects of arithmetic rounding.

Testing an implementation of the method presents a difficulty because the delay distributions can be calculated manually only for




(a)




(b)

Fig. 3—Example of a simple schedule whose delay distributions can readily be calculated.

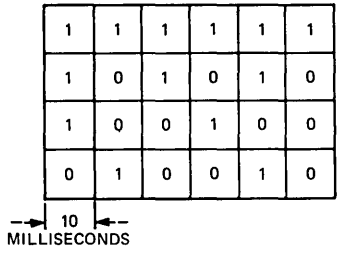
simple, somewhat atypical examples. Statistical simulations can be used to check for gross errors in the implementation, but it would be impractical to use simulation results to determine the accuracy of the extremes of the distribution tails. One example whose solution can be calculated manually is a schedule with a single task, scheduled every clock instant. If the execution time of the task is zero with probability $\alpha$, and $2T$ with probability $1 - \alpha$, it is straightforward to show that the waiting-time distribution is geometric, with parameter $(1 - \alpha)/\alpha$.

In the present form of the program, the FFTs are computed in double precision floating-point arithmetic, with the remainder of the computation being done in single precision. The accuracy of the computed delay distributions depends on the particular problem, but results are typically accurate to four decimal places when an IBM 3081 computer is used. When a *VAX** machine is used, which has the same word length but has different floating-point arithmetic characteristics, results are typically accurate to five decimal places.

## V. EXAMPLES

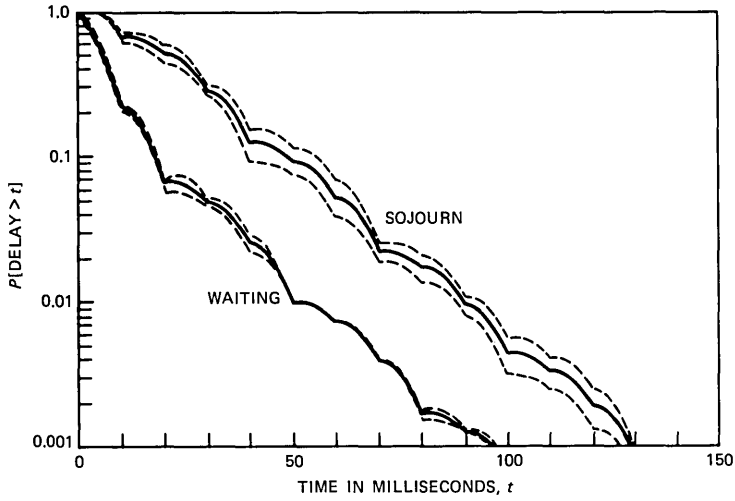
Figure 3a illustrates a simple example of a clocked schedule; the schedule table and the distributions of the execution times for the

* Trademark of Digital Equipment Corporation.




| TASK | $a$ | $b$ | $\bar{n}$ |
|---|---|---|---|
| 1 | 0.1 | 1.5 | 0.600 |
| 2 | 0.1 | 2.1 | 3.762 |
| 3 | 0.1 | 3.2 | 0.906 |
| 4 | 0.1 | 0.4 | 14.75 |

(a)

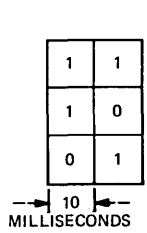


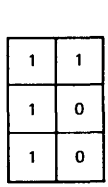

(b)

Fig. 4—Example of a schedule where the waiting and sojourn distributions depend on the slot at which the task is scheduled.

individual tasks are shown. The task of higher priority always takes 5 ms to execute. The task of lower priority, if it had sole use of the processor, would take 10 or 15 ms, with equal probability. This example is unusual in that the waiting and sojourn distributions can readily be computed manually. For the task of lower priority, the waiting time is always 5 ms—the time it waits while the higher-priority task is executed. In each slot, there are 5 ms available for the task of lower priority. Its sojourn time is therefore, with equal probability, 20 or 30 ms. The complementary cumulative distributions of waiting and sojourn time are shown in Fig. 3b.

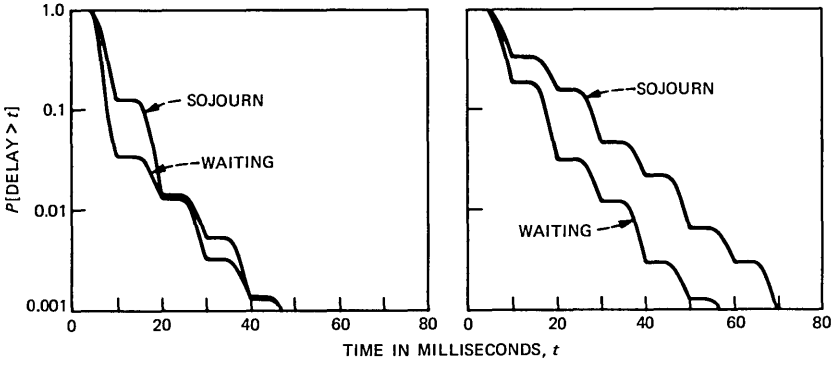
Figure 4a illustrates another simple clocked schedule. For each task, the execution time has the form $a + nb$, where $a$ denotes the overhead




| TASK | $a$ | $b$ | $\bar{n}$ |
|---|---|---|---|
| 1 | 0 | 0.1333 | 1.5 |
| 2 | 0 | 3.0 | 0.6 |
| 3 | 0 | 2.0 | 0.6 |

(a)




(b)

Fig. 5—Two schedules, identical except for the slot at which task 3 is scheduled, illustrating how delays can depend on details of the schedule.

incurred in starting the execution of the task, $b$ denotes the time to execute a single job, and $n$ is the number of jobs in the task, $n$ being Poisson distributed. This is a form for the task execution time assumed in Ref. 1. Figure 4a shows the schedule table and, for each task, the values of $a$ and $b$, and of $\bar{n}$, the expected number of jobs in the task. Figure 4b shows the complementary cumulative distributions of the waiting and sojourn times for task 4. In this example, the delay distributions differ according to the time slot at which the task is scheduled. The individual distributions are shown hatched; their average is shown as a continuous line. The presence of cusps is quite typical of delay distributions for low-priority tasks. They exist because, if a task of low priority is still being executed when a clock tick occurs, then it will probably have to wait considerably longer, while the newly scheduled high-priority work is executed.

Figure 5a illustrates two simple schedules; they differ only in the slot at which the task of lowest priority is scheduled. Figure 5b shows the resulting delay distributions for the task of lowest priority. An apparently minor change in the schedule results in significant increases in delay.

## VI. CONCLUSION

A method has been presented, based on the use of a discrete-time model, for the numerical computation of delay distributions for interruptable tasks in clocked schedules. The method is particularly useful when a detailed examination of the characteristics of a schedule is needed, such as how the delays depend on the slot where a task is scheduled. It does not depend on assumptions that delays are long, so it gives information about delays that are smaller than a clock period, as well as for longer delays. An important use of the method is checking the accuracy of computationally more economical, though approximate, methods of analysis.

Only systems in equilibrium have been discussed here. However, the method is capable of being extended to the analysis of transient conditions such as those that occur when a system controlled by a clocked schedule is subjected to a sudden increase of traffic.

The method presented here uses an iterative method to compute the equilibrium distribution of the work awaiting execution at each slot in the schedule. An alternative approach involves the direct solution of the equilibrium equations. Levinson's method for the solution of block Toeplitz systems[5] has been used, experimentally, for the analysis of clocked schedules by the direct solution of the equilibrium equations. However, this approach must be further developed in order to become a practical alternative to the iterative method described in this paper.

## VII. ACKNOWLEDGMENTS

I am grateful to A. A. Fredericks and B. T. Doshi for introducing me to clocked schedules and for many discussions on the topic. I am indebted to B. S. Gotz for her careful reading of an earlier version of this paper.

## REFERENCES

1. A. A. Fredericks, B. L. Farrell, and D. F. DeMaio, "Approximate Analysis of a Generalized Clocked Schedule," AT&T Tech. J., this issue.
2. A. A. Fredericks, "Analysis of a Class of Schedules for Computer Systems With Real Time Applications," *Performance of Computer Systems*, M. Arata, A. Butrimenko, E. Gelenbe, eds., New York: North-Holland, 1979, pp. 201–16.
3. L. Kleinrock, *Queueing Systems*, New York: Wiley, 1975, Vol. I, Ch. 8; Vol. II, Ch. 3.
4. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.
5. M. H. Ackroyd, "Stationary and Cyclostationary Finite Buffer Behaviour Computation via Levinson's Method," AT&T Bell Lab. Tech. J., *63*, No. 10 (December 1984), pp. 2159–70.

## AUTHOR

**Martin H. Ackroyd,** B.Sc. (Electronic and Electrical Engineering), 1966, Birmingham University, U.K.; Ph.D. (Electronic and Electrical Engineering),

1970, Loughborough University of Technology, U.K. Loughborough University, U.K.; Central Research Laboratories of EMI Limited; Aston University, U.K.; AT&T Bell Laboratories, 1982–1984; Essex University, 1984—. Mr. Ackroyd is now the Professor of Telecommunication and Information Systems in the Department of Electrical Engineering Science at Essex University, U.K. At AT&T Bell Laboratories, he was in the Performance Analysis Department of the Network Planning Division. His research interests include the numerical solution of problems in information system performance engineering. Previously he has done research in other areas, including digital signal and image processing. Senior member, IEEE.

AT&T Technical Journal
Vol. 64, No. 2, February 1985
Printed in U.S.A.

# Analysis of Clocked Schedules—High-Priority Tasks

By B. T. DOSHI*

(Manuscript received December 8, 1983)

Clocked schedules are used in a variety of real-time systems to perform tasks in accordance with their delay requirements. A comprehensive overview of various clocked schedules can be found in the accompanying paper by Fredericks et al. For one model of a clocked schedule we can see approximate and exact analysis of the relevant performance measures in the papers by Fredericks et al. and by Ackroyd in this issue of the *Journal.* These results can also be used to evaluate the long-term delays in the other models. For extremely time-critical (high-priority) tasks, the probability of tasks not getting served in the scheduled interval and the short-term delay distribution are important performance measures that are sensitive to the detail structure of the clocked schedule. In this paper, we show that these performance measures can be obtained in terms of steady-state distribution of an embedded Markov chain. This steady-state distribution is calculated, exactly or approximately, for a number of models, and the results are used to compare, numerically, various scheduling mechanisms.

## I. INTRODUCTION

Clocked schedules are used in a variety of real-time systems to perform tasks in accordance with their timing requirements. These real-time systems (switching, monitoring, etc.) are characterized by having to perform some tasks with extremely stringent timing requirements. We call these high-priority tasks. Besides these, real-time systems also perform tasks that can tolerate somewhat longer delays

* AT&T Bell Laboratories.

without affecting performance. These will be called low-priority tasks. Finally, these systems perform a variety of background tasks (audits, maintenance, etc.), which have very liberal timing requirements but which require a specified minimum fraction of the processor time over a reasonably long period of time. These tasks are usually performed when no high- or low-priority work is present in the system. We call these the fill tasks. We emphasize that the boundaries among these categories of tasks are not clear-cut and that within each category the tasks may have significantly differing time scales. There are various mechanisms for allocating the processor time to meet the varying requirements of the individual tasks. These mechanisms and their relative merits are discussed in Ref. 1. Clocked schedules provide a very effective and reliable mechanism to achieve the desired timing objectives. Of course, to use them effectively, it is necessary to understand their performance as it applies to the various categories of tasks. Considerable progress has been made in this direction (see Refs. 1, 2, and 5 through 8). For low-priority and fill tasks approximate analysis of queueing models is described in Ref. 1. Exact numerical procedures for these models are developed in Ref. 2. For many systems the analyses in Refs. 1 and 2 can also be used to study high-priority tasks. However, for other systems the modeling assumptions may not adequately capture the working of the schedule to accurately analyze the short-term delay of high-priority tasks. For such systems it is necessary to incorporate additional features into the clocked schedule model and understand the implications of these features. Some of these features are the gating mechanism, the strategies used under overrun, and the dependence introduced by autonomous work generation. In this paper we consider increasingly complex models of clocked schedule by introducing these features one at a time, and we develop methodology to study the performance measures for the high-priority tasks in these situations. To analyze these models in a relatively simple way, we make assumptions that restrict our analysis only to high-priority tasks.

This paper is organized as follows: In Section II we briefly define the clocked schedule and the performance measures that we will use. We also introduce various gating mechanisms and overrun strategies in that section. In Section III we analyze a simple clocked schedule with respect to the performance measures for the high-priority tasks. We also use this analysis to numerically compare various scheduling and priority mechanisms. Section IV deals with various generalizations and complex clocked schedules.

## II. MODEL OF A CLOCKED SCHEDULE

The mechanism underlying a clocked schedule has been discussed in detail in Ref. 1, so we will be very brief here.

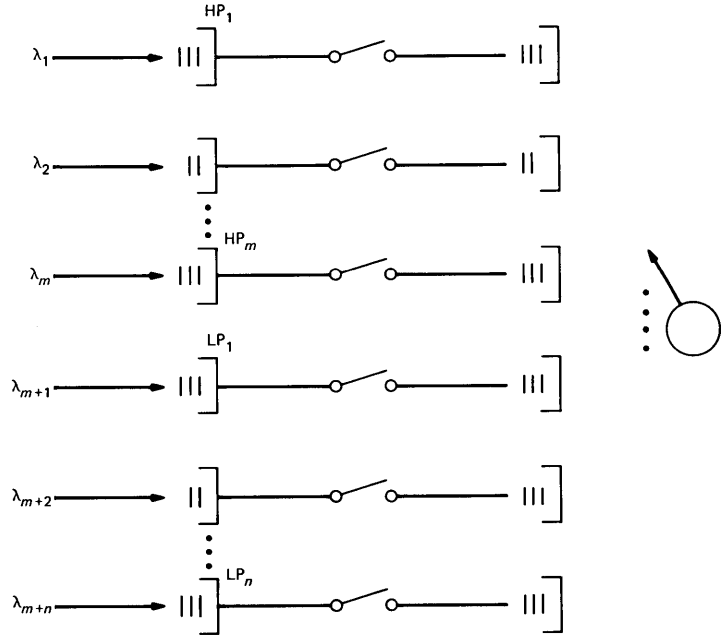



Fig. 1—Multiclass queueing structure in clocked schedules.

Assume that there are $m$ high-priority tasks and $n$ low-priority tasks numbered $\{1, 2, \ldots, m, m+1, \ldots, m+n\}$. Each task has two queues, the external queue and the internal queue (see Fig. 1). Jobs arrive at the external queues and, at specified times, are transferred to the corresponding internal queues. The internal queues are served according to a priority scheme, with lower-numbered queues having higher priority. The priority of the high-priority queue over the low-priority queue and the fill work is preemptive. The priority between two high-priority queues is determined by the overrun strategy discussed below.

The time is divided into intervals of length $T$ ms. During each of these time slots some high-priority tasks and some low-priority tasks are scheduled (for an example see Fig. 2). The jobs in the queues for the tasks scheduled in a given slot are typically moved from the external queues to the internal queues in that time interval. The gating mechanism determines when this transfer happens. Three gating mechanisms are used in practice, the actual choice depending on the application and hardware limitations. The first gating mechanism (G1) transfers the jobs in the external queues to the internal queues for all tasks scheduled in a given interval at the beginning of that interval. No more transfer takes place during that interval. In the second gating mechanism (G2) the processor starts with task 1 at the

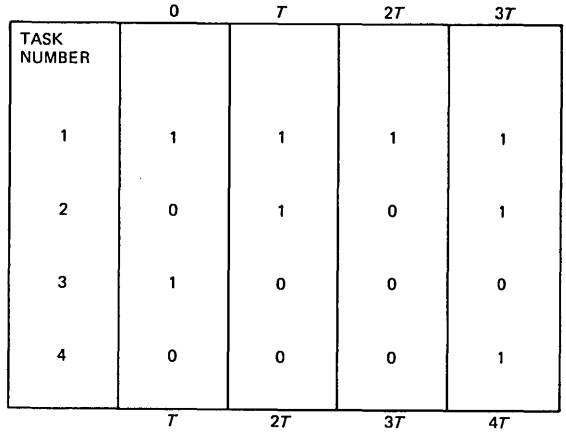



Fig. 2—An example of a clocked schedule for $m = 2$, $n = 2$.

beginning of each interval. If it is scheduled for that interval the transfer takes place and the processor serves the internal queue for task 1 to completion. During this period no additional transfer takes place for queue 1. After exhausting the first queue the processor moves to queue 2. If the corresponding task is scheduled, the transfer takes place at this time, and so on. Note that the order of service for the internal queues is always (1, 2, . . . , $m + n$, fill). The schedule table only indicates when the transfer takes place for each task. The third gating mechanism (G3) is essentially similar to G2, the difference being that when the processor is serving an internal queue, the new arrivals to the corresponding external queue move immediately to the internal queue.

If all the internal queues become empty before any interval ends, then the processor does fill work until the end of that interval. If the interval ends while the processor is working on a low-priority job, then that work is preempted and the processor moves to queue 1. If the processor is working on a high-priority queue when the interval ends, then we say an overrun has occurred. We consider three different overrun strategies, S1, S2, and S3. In strategy S1 (Fig. 3) the remaining high-priority work in the internal queues is expelled. This is done in some systems either because the necessary bookkeeping ability is not available or because serving a high-priority task after a reasonably long delay is most likely to result in unsuccessful service and waste of the processor time. More importantly, as we will see in Section IV, this strategy is the starting point of our analysis for more common strategies S2 and S3, which are discussed below. The strategy S2 (Fig.

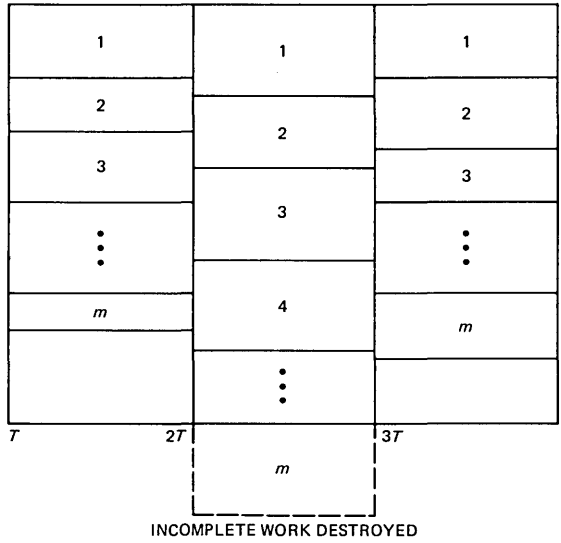



Fig. 3—Strategy S1, in which incomplete work is destroyed.

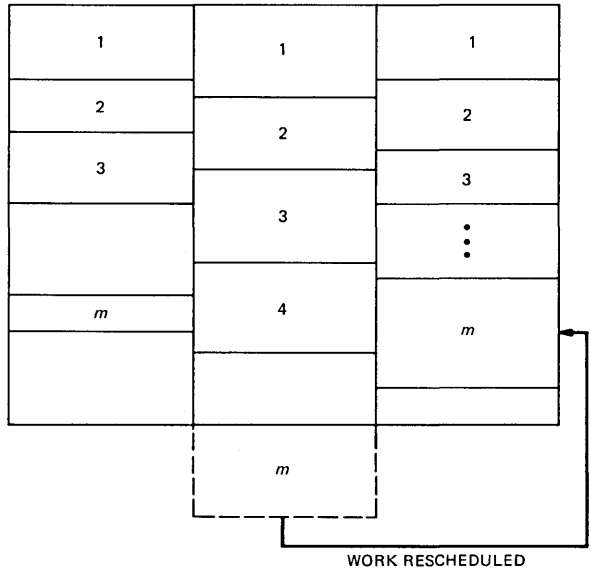



Fig. 4—Strategy S2, in which incomplete work is rescheduled.

4) under overrun is the same as that used for interrupting a low-priority task, namely, the incomplete work is rescheduled and the processor moves back to queue 1. In strategy S3 (Fig. 5) the end of interval is ignored until all the high-priority internal queues are empty.

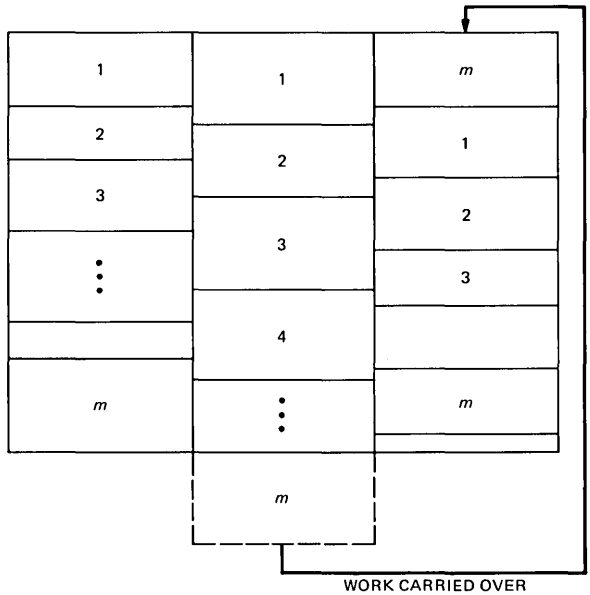



Fig. 5—Strategy S3, in which incomplete work is carried over.

Only then does the processor move to queue 1. Note, however, that the end of interval is usually marked by an interrupt from an asynchronous clock. Thus in S3 an overrun reduces the time available in the next interval.

As mentioned earlier, the high-priority tasks are typically very time critical. In this paper we consider high-priority tasks as those which we require to be completed in the scheduled interval with very high probability. This dictates which performance measures capture our interest.

We are interested in the probability of an overrun. Assuming that a high-priority task is scheduled once every $p$ time slots, we are interested in the delay distribution in the range $[pT, (p + 1)T]$. Once we address these two questions successfully, we can answer the other important questions: What is the largest $T$ for which the delay criteria for all high-priority tasks are satisfied? This frequently determines the best value of $T$. Which is the best priority order for the high-priority tasks?

Since we are interested in the delay distribution over a short time range, the effects of gating mechanism and overrun may be quite significant here. In this paper we develop a methodology to address the issues discussed above in a manner suitable for relatively easy computations. We begin with a rather simple special case of the general model and then introduce complexity in the schedule gradually.

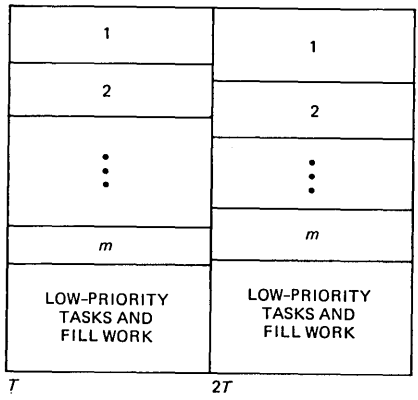



Fig. 6—A simple clocked schedule.

## III. ANALYSIS OF A SIMPLE CLOCKED SCHEDULE

Recall that high-priority tasks have preemptive priority over low-priority and fill tasks and so are not affected by them. Thus we can simplify the description of the schedule by replacing all the low-priority and the fill tasks with a single task, $m + 1$.

The special case we study here (see Fig. 6) is a clocked schedule in which each of the $m$ high-priority tasks is scheduled in each interval. Thus the entire schedule has a period of one time interval ($T$). In addition we assume that the jobs in the external queue $i$ arrive according to a Poisson process at rate $\lambda_i$, and that the arrival processes in different queues are independent. When the processor starts serving queue $i$, it incurs an overhead $OH_i$ with distribution function $H_i$. Each job in queue $i$ has service time $X_i$ with distribution function $F_i$, $i = 1, \ldots, m$. For $i = 1, 2, \ldots, m$, let

$$a_i = \int_0^\infty x dH_i(x),$$

$$\alpha_i = \int_0^\infty (x - a_i)^2 dH_i(x),$$

$$b_i = \int_0^\infty x dF_i(x),$$

$$\beta_i = \int_0^\infty x^2 dF_i(x),$$

$$\rho_i = \lambda_i b_i,$$

and

$$\gamma_i = \lambda_i \beta_i.$$

We analyze this clocked schedule with overrun strategy S1 and gating mechanisms G1 and G2. Note that we require the high-priority queues to exhaust in every interval with high probability, so the difference between overrun strategies S1, S2, and S3 should be small.

### *3.1 Analysis for gating mechanism G1*

The analysis for the gating mechanism G1 is simple. First consider an arbitrary clock interval of length $T$. Suppose we let this interval continue until all the internal high-priority queues are empty. Let $t_i$ denote the time from the beginning of this interval to the moment the processor exhausts internal queue $i$. Then

$$t_i = \sum_{j=1}^{i} OH_j + \sum_{j=1}^{i} \sum_{k=1}^{N_j} X_{jk}, \tag{1}$$

where $X_{jk}$ is the service time of the $k$th job in the $j$th queue and $N_j$ is the number of jobs served by the processor from the $j$th queue. $N_j$ has Poisson distribution with mean $\lambda_j T$. Thus, if $\tilde{G}_i$, $\tilde{H}_i$ and $\tilde{F}_i$ denote the Laplace transforms of $t_i$, $OH_i$ and $X_i$, respectively, then

$$\tilde{G}_i(s) = \prod_{j=1}^{i} [\tilde{H}_j(s) e^{-\lambda_j T(1-\tilde{F}_j(s))}]. \tag{2}$$

The above equation can be inverted to obtain the distribution of $t_i$. This can be done, for example, by using the transform inversion technique discussed in Ref. 3. Alternatively, the convolutions involved in eq. (1) can be carried out directly using fast Fourier transforms after suitably discretizing the random variables.[2] Obviously, an overrun occurs when $t_m > T$. Thus,

$$P\{\text{overrun}\} = P\{t_m > T\} = 1 - G_m(T).$$

Also,

$$\int_{t=0}^{\infty} e^{-st}(1 - G_m(t))dt = \frac{1 - \tilde{G}_m(s)}{s}. \tag{3}$$

Thus, we can get $P\{\text{overrun}\}$ by inverting the right-hand side of eq. (3) at $T$.

Next, we obtain the sojourn time distribution for a job arriving at queue $i$ assuming that it does get served in the following time slot. Let $S_i$ denote the sojourn time for an arbitrary job arriving at queue $i$, that is, $S_i$ is the time between the arrival of a job at queue $i$ and the completion of its service. Suppose that a job arrives at queue $i$ when $R$ time units are left to the end of a time slot and that there are $N_i$ jobs waiting in that external queue at that time. Then $R$ is uniformly distributed on $(0, T)$ and, conditioned on $R$, $N_i$ has Poisson distribution with mean $\lambda_i(T - R)$. The sojourn time, $S_i$, is then given by

$$S_i = R + t_{i-1} + \sum_{j=1}^{N_i+1} X_{ij} + OH_i.$$

Thus, if $\tilde{S}_i$ denotes the Laplace transform of $S_i$, then

$$\tilde{S}_i(s) = \frac{\tilde{H}_i(s)\tilde{G}_{i-1}(s)\tilde{F}_i(s)[e^{-\lambda_i T(1-\tilde{F}_i(s))} - e^{-sT}]}{T(s - \lambda_i(1 - \tilde{F}_i(s))}. \tag{4}$$

Once again, eq. (4) can be inverted to obtain the distribution of $S_i$.

We next obtain the mean and variance of $t_i$ and the mean of $S_i$. From eq. (2) we easily get

$$E[t_i] = \sum_{j=1}^{i} (a_j + \rho_j T), \tag{5}$$

and

$$\text{Var}[t_i] = \sum_{j=1}^{i} (\alpha_j + \gamma_j T). \tag{6}$$

Also, by differentiating eq. (4) and setting $s = 0$, we get

$$\bar{S}_i = E[S_i] = \sum_{j=1}^{i-1} (a_j + \rho_j T) + b_i + \frac{T}{2}(1 + \rho_i) + a_i. \tag{7}$$

### *3.2 Analysis for gating mechanism G2*

We next analyze the simple clocked schedule under gating mechanism G2. The overrun strategy is still assumed to be S1. Our general approach is to define an embedded Markov chain and show that the relevant performance measures can be obtained easily in terms of the steady-state distribution of this Markov chain. We then find expression for this steady-state distribution.

Let $N_{n,i}$ denote the number of jobs in the external queue $i$ when the $n$th time slot of length $T$ begins. Then $\{(N_{n1}, N_{n2}, \cdots N_{nm}): n \geq 1\}$ is an ergodic Markov chain. Let $(N_1, N_2, \cdots N_m)$ denote the steady-state random vector corresponding to this Markov chain and let $\tilde{P}(z_1, z_2, \cdots, z_m)$ denote the generating function of the random vector $(N_1 \cdots N_m)$, that is,

$$\tilde{P}(z_1, z_2, \cdots, z_m) = E[z_1^{N_1} z_2^{N_2} \cdots z_m^{N_m}].$$

We first show that the probability of overrun and the sojourn time distribution for any queue can be obtained easily in terms of $\tilde{P}$. Consider an arbitrary time slot and, as before, allow all high-priority internal queues to be completely served in this slot by letting the available interval overrun, if necessary. Let $t_i$ denote the time from the beginning of this time slot until the completion of service to the internal queue $i$. Let $(N_1, N_2, \cdots N_m)$ denote the number in the external queues at the beginning of this slot, and for $i = 1, 2, \cdots m$,

let $N_i'$ denote the number transferred from the $i$th external queue to the internal queue when the service to that queue begins. Let $t_o = 0$. Then for $i = 1, 2, \cdots, m$,

$$t_i = t_{i-1} + OH_i + \sum_{j=1}^{N_i'} X_{ij}, \tag{8}$$

$$N_j' = N_i + M_i(t_{i-1}), \tag{9}$$

where

$$M_i(x) \sim \text{Poisson } (\lambda_i x). \tag{10}$$

Equations (8) through (10) allow us to calculate the distributions of $t_i$ and $N_i'$, $i = 1, 2, \cdots m$, recursively. In terms of transforms, let

$$\tilde{G}(s_1, s_2, \cdots s_m) = E[e^{-(s_1t_1+s_2t_2+\cdots+s_mt_m)}].$$

Then, $\tilde{G}$ can be obtained recursively from eqs. (8) through (10) as follows: Let $\underline{s} = (s_1, s_2, \cdots, s_m)$. Define $\sigma_i(\underline{s})$ and $\omega_i(\underline{s})$, $i = 1, \cdots, m$, recursively by

$$\sigma_m(\underline{s}) = s_m,$$

$$\omega_m(\underline{s}) = \tilde{F}_m(\sigma_m(\underline{s})),$$

$$\sigma_i(\underline{s}) = \lambda_{i+1}(1 - \omega_{i+1}(\underline{s})) + \sigma_{i+1}(\underline{s}) + s_i, \qquad 1 \leqslant i \leqslant m-1, \tag{11}$$

and

$$\omega_i(\underline{s}) = \tilde{F}_i(\sigma_i(\underline{s})), \qquad 1 \leqslant i \leqslant m-1.$$

Then

$$\tilde{G}(\underline{s}) = \tilde{P}[\omega_1(\underline{s}), \omega_2(\underline{s}), \omega_3(\underline{s}), \cdots \omega_m(\underline{s})] \prod_{i=1}^{m} \tilde{H}_i[\sigma_i(\underline{s})]. \tag{12}$$

The marginal Laplace transform of $t_m$ now follows by putting $\underline{s} = \underline{s}^{(m)} = (0, 0, \cdots, \underline{s}_m)$. Once again, this can be inverted to get the distribution function of $t_m$. In particular,

$$P\{\text{overrun}\} = 1 - P\{t_m < T\},$$

and

$$\int_0^\infty e^{-st} P\{t_m > t\} dt = \frac{1 - \tilde{G}(\underline{s}^{(m)})}{s}. \tag{13}$$

Thus $P\{\text{overrun}\}$ can be obtained by inverting the right-hand side in eq. (13) at $T$.

We next derive the distribution function of the sojourn time in a given queue in terms of $\tilde{P}$. Suppose we are considering queue $i$. We assume that the probability that the task $i$ overruns the interval is essentially zero. Let $S_i$ denote sojourn time of an arbitrary job in queue $i$ and $\tilde{S}_i$ its Laplace transform. Let $N_{i1}''$ and $N_{i2}''$ denote, respectively, the numbers in the internal queue $i$ and the external queue $i$ when an

arbitrary job in queue $i$ completes service. Let $N_i'' = N_{i1}'' + N_{i2}''$. Finally, let $N_i'$ be the number of jobs in the external queue $i$ just before they are transferred to the internal queue and the processor starts working on them. Let $\tilde{r}_{i1}$, $\tilde{r}_{i2}$, $\tilde{r}_i$, and $\tilde{q}_i$ denote the generating functions of $N_{i1}''$, $N_{i2}''$, $N_i''$, and $N_i'$, respectively. We express $\tilde{S}_i$ in terms of $\tilde{r}_i$, $\tilde{r}_i$ in terms of $\tilde{q}_i$, and $\tilde{q}_i$ in terms of $\tilde{P}$. Combining these relations we will get $\tilde{S}_i$ in terms of $\tilde{P}$.

First note that, since the service within a given queue is First-In First-Out (FIFO), $N_i''$ denotes the number of arrivals to queue $i$ during a time interval of length $S_i$. Thus, from Ref. 4 we get

$$\tilde{S}_i(s) = \tilde{r}_i(1 - s/\lambda_i). \tag{14}$$

Next,

$$\begin{aligned}\tilde{r}_i(z) &= E[z^{N_i''}] \\ &= E[E[z^{N_i''} | N_i']] \\ &= E[E[E[z^{N_{i1}''+N_{i2}''} | K_i] | N_i']],\end{aligned} \tag{15}$$

where $K_i$ is the position of the job that completed service in the batch of size $N_i'$ transferred to the $i$th internal queue when the processor last arrived there. From eq. (15) we get

$$\begin{aligned}\tilde{r}_i(z) &= E[E[z^{N_i'-K_i}\tilde{F}_i(\lambda_i(1-z))^{K_i}\tilde{H}_i(\lambda_i(1-z)) | N_i']] \\ &= \frac{\tilde{H}_i(\lambda_i(1-z))(\tilde{F}_i(1-z))[\tilde{q}_i(z) - \tilde{q}_i(\tilde{F}_i(\lambda_i(1-z)))]}{\bar{n}_i(z - \tilde{F}_i(\lambda_i(1-z)))},\end{aligned} \tag{16}$$

where $\bar{n}_i = E[N_i']$. Finally, by the arguments used to derive eq. (12) we obtain the following relation between $\tilde{q}_i$ and $\tilde{P}$: Let

$$\begin{aligned}\sigma_{i-1}'(z) &= \lambda_i(1-z), \\ \omega_{i-1}'(z) &= \tilde{F}_{i-1}[\sigma_{i-1}(z)], \\ \sigma_j'(z) &= \lambda_j(1 - \omega_{j+1}'(z)) + \sigma_{j+1}'(x), \qquad 1 \leqslant j \leqslant i-1,\end{aligned}$$

and

$$\omega_j'(z) = \tilde{F}_j(\sigma_j'(z)) \quad 1 \leqslant j \leqslant i-2.$$

Then

$$\begin{aligned}\tilde{q}_i(z) &= E[z^{N_i'}] \\ &= \prod_{j=1}^{i-1} \tilde{H}_j(\sigma_j'(z))\tilde{P}(\omega_1'(z), \omega_2'(z), \cdots, \omega_{i-1}'(z), z).\end{aligned} \tag{17}$$

Combining eqs. (14), (16), and (17) we obtain $\tilde{S}_i$ in terms of $\tilde{P}$.

Thus, all performance measures of interest can be expressed in terms of the steady-state distribution of the vector $(N_1, N_2, \cdots, N_m)$. We next evaluate this distribution $P(l_1, l_2, \cdots, l_m)$ or, equivalently, its generating function $\tilde{P}(z_1, z_2, \cdots, z_m)$. Let

$$Q(K_1, K_2, \cdots, K_m | l_1, l_2, \cdots, l_m)$$
$$= P\{N_{n+1,1} = k_1, N_{n+1,2} = k_2, \cdots, N_{n+1,m}$$
$$= k_m | N_{n,1} = l_1, \cdots, N_{n,m} = 1_m\}$$

be the transition probability function for the Markov chain $\{(N_{n,1}, N_{n,2}, \cdots, N_{n,m}): n \geq 1\}$. Then $P$ is the unique nonnegative solution of

$$P(k_1, k_2, \cdots, k_m)$$
$$= \sum_{l_i=0}^{\infty} \sum_{l_2=0}^{\infty} \sum_{l_m=0}^{\infty} P(l_1, l_2, \cdots, l_m) Q(k_1, k_2, \cdots, k_m | l_1, \cdots, l_m), \quad (18)$$

for $k_i = 0, 1 \cdots$, $i = 1, 2 \cdots m$, and

$$\sum_{k_1=0}^{\infty} \sum_{k_2=0}^{\infty} \cdots \sum_{k_m=0}^{\infty} P(k_1 \cdots k_m) = 1. \quad (19)$$

$Q$ can be readily expressed in terms of $l_1, l_2, \cdots, l_m, k_1, k_2, \cdots, k_m$ and the arrival and service time parameters. In particular, it is clear that $Q(k_1, \cdots k_m | l_1, l_2, \cdots l_m)$ does not depend on $l_m$. Thus, if we define

$$P_i(k_1, k_2, \cdots, k_i) = P\{N_1 = k_1, \cdots, N_i = k_i\} \quad (20)$$

and

$$Q_i(k_1 \cdots k_i | l_1, l_2, \cdots, l_{i-1})$$
$$= P\{N_{n+1,1} = k_1, \cdots, N_{n+1,i} = k_i | N_{n,1}$$
$$= l_1, \cdots, N_{n,i-1} = l_{i-1}\}, \quad (21)$$

for $1 \leq i \leq m$, then

$$P_m(k_1, \cdots, k_m) = \sum_{l_1=0}^{\infty} \sum_{l_2=0}^{\infty} \cdots \sum_{l_m=0}^{\infty} P_m(l_1 \cdots l_m)$$
$$\cdot Q_m(k_1, \cdots k_m | l_1 \cdots l_{m-1})$$
$$= \sum_{l_1=0}^{\infty} \cdots \sum_{l_{m-1}=0}^{\infty} P_{m-1}(l_1 \cdots l_{m-1})$$
$$\cdot Q_m(k_1 \cdots k_m | l_1 \cdots l_{m-1}), \quad (22)$$

and, in general, for $i = 2, \cdots m$,

$$P_i(k_1, \cdots k_i) = \sum_{l_1=0}^{\infty} \cdots \sum_{l_{i-1}=0}^{\infty} P_{i-1}(l_1, \cdots, l_{i-1})$$
$$\cdot Q_i(k_1 \cdots, k_i | l_1 \cdots, l_{i-1}). \quad (23)$$

Thus $P_1, P_2, \cdots, P_m$ can be calculated recursively using eq. (23). We start these recursions with

$$P_1(k_1) = \frac{e^{-\lambda_1 T}(\lambda_1 T)^{K_1}}{k_1!} k_1 = 0, \cdots .$$

These recursions use the functions $Q_i$ for $i = 1, 2, \cdots m$. Of course, some form of discretization is necessary to carry out these recursions. For large $m$ it is convenient to have a closed form approximation instead. Such an approximation is possible if we assume that the probability of an overrun occurring by queue $i$, $i = 1, \cdots m - 1$ is negligible. We illustrate this approximation below for the case $m = 2$ and then write down the approximation for general $m$. For $m = 2$, the exact generating function is given by

$$\begin{aligned}\tilde{P}(z_1, z_2) &= E[z_1^{N_1} z_2^{N_2}] \\ &= E[E[z_1^{N_1} z_2^{N_2} | t_1']] \\ &= E[e^{-\lambda_1 T(1-z_1)-\lambda_2 T(1-z_2)+\lambda_2 t_1'(1-z_2)}],\end{aligned}$$

where $t_1'$ represents the time from the beginning of the previous slot until the completion of all work in the internal queue 1 in that slot. Thus,

$$\tilde{P}(z_1, z_2) = e^{-\lambda_1 T(1-z_1)-\lambda_2 T(1-z_2)} \cdot \sum_{l_1=0}^{\infty} \frac{e^{-\lambda_1 T}(\lambda_1 T)^{l_1}}{l_1!} \cdot \left[ \begin{aligned} & e^{\lambda_2 T(1-z_2)} \int_{s_1=T}^{\infty} \int_{x_1=0^-}^{s_1} dH_1(x_1) dF_1^{(l_1)}(s_1 - z_1) \\ & + \int_{s_1=0^-}^{T} \int_{x_1=0}^{s_1} dH_1(x_1) dF_1^{(l_1)}(s_1 - x_1) e^{\lambda_2 s_1(1-z_2)} \end{aligned} \right]. \quad (24)$$

Our approximation is equivalent to replacing the first term in the squared bracket in eq. (24) by

$$\int_{s_1=T}^{\infty} \int_{x_1=0^-}^{s_1^+} dH_1(x_1) dF_1^{(l_1)}(s_1 - z_1) e^{\lambda_2 s_1(1-z_2)}.$$

If we denote the resulting approximation for $\tilde{P}$ by $\bar{P}$, then

$$\bar{P}(z_1, z_2) = \tilde{H}_i(-\lambda_2(1 - z_2)) e^{-\lambda_1 T(1-z_1)-\lambda_2 T(1-z_2)-\lambda_1 T(1-\tilde{F}_1(-\lambda_2(1-z_2)))}. \quad (25)$$

How good is $\bar{P}$ an approximation for $\tilde{P}$? If we put $z_1 = 1$ in eqs. (24) and (25) and then invert the resulting generating functions for $N_2$, we can compare the exact and approximate distributions for $N_2$. Tables I through III give such distributions for three sets of parameters. The approximation seems remarkably good in these cases. These tables

Table I—Distribution of the number in queue 2 at the beginning of an interval with $\lambda_1 = 0.05, \lambda_2 = 0.10, \underline{b_1 = 1.0}, b_2 = 1.0, T = 20.0$

| | $P(N_2 = k)$ | |
|---|---|---|
| $k$ | Exact | Approximation Using $\bar{P}$ |
| 0 | 0.055308 | 0.055308 |
| 1 | 0.104504 | 0.104504 |
| 2 | 0.099035 | 0.099035 |
| 3 | 0.062750 | 0.062750 |
| 4 | 0.029901 | 0.029901 |
| 5 | 0.003648 | 0.003648 |
| 6 | 0.001001 | 0.001001 |
| 7 | 0.000241 | 0.000241 |
| 8 | 0.000052 | 0.000052 |

Table II—Distribution of the number in queue 2 at the beginning of an interval with $\lambda_1 = 0.05, \lambda_2 = 0.10, \underline{b_1 = 2.0}, b_2 = 1.0, T = 20.0$

| | $P(N_2 = k)$ | |
|---|---|---|
| $k$ | Exact | Approximation Using $\bar{P}$ |
| 0 | 0.062126 | 0.062126 |
| 1 | 0.109075 | 0.109075 |
| 2 | 0.097270 | 0.097270 |
| 3 | 0.058602 | 0.058602 |
| 4 | 0.026782 | 0.026782 |
| 5 | 0.009889 | 0.009889 |
| 6 | 0.003069 | 0.003069 |
| 7 | 0.000823 | 0.000823 |
| 8 | 0.000194 | 0.000194 |

Table III—Distribution of the number in queue 2 at the beginning of an interval with $\lambda_1 = 0.05, \lambda_2 = 0.10, \underline{b_1 = 3.0}, b_2 = 1.0, T = 20.0$

| | $P(N_2 = k)$ | |
|---|---|---|
| $k$ | Exact | Approximation Using $\bar{P}$ |
| 0 | 0.070641 | 0.070636 |
| 1 | 0.112676 | 0.112681 |
| 2 | 0.094152 | 0.094152 |
| 3 | 0.054193 | 0.054193 |
| 4 | 0.023988 | 0.023988 |
| 5 | 0.008665 | 0.008665 |
| 6 | 0.002651 | 0.002651 |
| 7 | 0.000704 | 0.000704 |
| 8 | 0.000166 | 0.000166 |

also suggest the region in which the approximation is likely to work well.

In general let $\bar{P}_m$ represent the approximation for $\hat{P}_m$ and for $1 \leq i \leq m-1$, let

$$\bar{P}_i(z_1, z_2, \cdots z_i) = \bar{P}_m(z_1, z_2, \cdots, z_i, 1, 1, \cdots 1). \tag{26}$$

Then we can express $\bar{P}_i$ in terms of $\bar{P}_{i-1}$ as follows, thus enabling a simple recursive calculation of $\bar{P}_m$: For $\underline{z} = (z_1, z_2, \cdots, z_m)$, let

$$\phi_i(\underline{z}) = \lambda_i(1 - z_i),$$

$$\psi_m(\underline{z}) = \phi_m(\underline{z}),$$

and for $1 \leq i \leq m-1$,

$$\psi_i(\underline{z}) = \phi_i(\underline{z}) + \psi_{i+1}(\underline{z}) - \lambda_i(1 - \tilde{F}_i(-\psi_{i+1}(\underline{z}))). \tag{27}$$

Then

$$\begin{aligned}\bar{P}_m(z_1, \cdots, z_m) &= e^{-T\sum_{i=1}^{m}\phi_i(\underline{z})} \prod_{i=1}^{m-1} \tilde{H}_i[-\psi_{i+1}(\underline{z})] \\ &\cdot \bar{P}_m(\tilde{F}_1(-\psi_2(\underline{z})), \tilde{F}_2(-\psi_3(\underline{z})), \cdots, \tilde{F}_{m-1}(-\psi_m(\underline{z})), 1).\end{aligned} \tag{28}$$

This represents $P_m$ in terms of $P_{m-1}$. Applying this equation successively $m$ times we get $P_m$ in terms of $P_{m-1}, P_{m-2}, \cdots, P_1$ and $P_0 = P_m(1, 1, \cdots, 1) = 1$. Thus $P_m$ can be obtained recursively from eq. (28).

### *3.3 Moments for gating mechanism G2*

We again look at the case $m = 2$ and obtain approximate expressions for the first two moments of $t_1$ and $t_2$, where $t_1$ and $t_2$ are as defined in Section 3.1. We also obtain the mean values of the sojourn times, $S_i$, $i = 1, 2$, assuming that the probability of an overrun is negligible. We compare these expressions with those for gating mechanism $G_1$ and also with the gating mechanism G2 but with indices 1 and 2 reversed, that is, with queue 2 served at higher priority than queue 1. We will use our approximation throughout this analysis. First, from eqs. (12) and (28) we get

$$\begin{aligned}\tilde{G}(s_1, s_2) &= E[e^{-t_1 s_1 - t_2 s_2}] \\ &= \tilde{H}_2(s_2)\tilde{H}_1(s_1 + s_2 + \lambda_2(1 - \tilde{F}_2(s_2))\tilde{H}_1(-\lambda_2(1 - \tilde{F}_2(s_2)))) \\ &\quad \cdot e^{-T[\lambda_2(1 - \tilde{F}_2(s_2)) + \lambda_1[2 - \tilde{F}_1(s_1 + s_2} \\ &\quad + \lambda_2(1 - \tilde{F}_2(s_2)) - \tilde{F}_1(-\lambda_2(1 - \tilde{F}_2(s_2))))]].\end{aligned} \tag{29}$$

Putting $s_2 = 0 (s_1 = 0)$ in eq. (29) we get the approximate Laplace transform of $t_1 (t_2)$. Let

$$\tilde{g}_1(s) = \text{Log } \tilde{G}(s, 0) \tag{30}$$

and

$$\tilde{g}_2(s) = \text{Log } \tilde{G}(0, s). \tag{31}$$

Then

$$E[t_i] = -\tilde{g}_i'(0^+) \tag{32}$$

and

$$\text{var}[t_i] = \tilde{g}_i''(0^+) \tag{33}$$

for $i = 1, 2$. Substituting from eq. (29) into eqs. (30) through (33), we get

$$E[t_1] = \alpha_1 + \rho_1 T, \tag{34a}$$

$$E[t_2] = \alpha_1 + \alpha_2 + (\rho_1 + \rho_2)T, \tag{34b}$$

$$\text{Var}[t_1] = \alpha_1 + \gamma_1 T, \tag{34c}$$

and

$$\text{Var}[t_2] = \alpha_1 + \alpha_2 + T(\gamma_1 + \gamma_2) + 2\rho_2(1 + \rho_2)(\alpha_1 + \gamma_1 T). \tag{35}$$

Comparing these with the expressions derived in Section 3.1 for the gating mechanism G1, we conclude that the mean values of $t_1$ and $t_2$ do not depend on the gating mechanism. The variance of $t_1$ also remains the same. However, the variance of $t_2$ for the gating $G_2$ is larger than that for gating $G_1$. The difference is

$$2\rho_2(1 + \rho_2)(\alpha_1 + \gamma_1 T) > 0. \tag{36}$$

This implies that the probability of an overrun is likely to be larger for gating G2 than for gating G1. The difference depends on the average load in queue 2 ($\rho_2$) and the variability of the load in queue 1 ($\alpha_1 + \gamma_1 T$). Since, in general, it is much easier to analyze G1 than to analyze G2, we frequently use the gating G1 as an approximation for gating G2. The left-hand side in (36) then gives a measure of this approximation. If this quantity is small the approximation is likely to be good. We will come back to this point in Section 3.4.

Next, we see what happens if we interchange queues 1 and 2. In other words, the parameters of the arrival and service time for these two queues are interchanged and we compare the moments of $t_1$ and $t_2$ for these two orders. We will use suffix 0 for the original order and $R$ for the reversed order. Then,

$$E_R[t_1] - E_0[t_1] = 0,$$

$$E_R[t_2] - E_0[t_2] = 0,$$

$$\mathrm{Var}_R[t_1] - \mathrm{Var}_0[t_1] = 0,$$

and

$$\mathrm{Var}_R[t_2] - \mathrm{Var}_0[t_2] = 2\rho_1(1 + \rho_1)(\alpha_2 + \gamma_2 T) - 2\rho_2(1 + \rho_2)(\alpha_1 + \gamma_1 T).$$

If we associated smaller variance with better performance, then the order (1, 2) is better than the order (2, 1) if

$$\frac{\rho_1(1 + \rho_1)}{\alpha_1 + \gamma_1 T} > \frac{\rho_2(1 + \rho_2)}{\alpha_2 + \gamma_2 T}. \tag{37}$$

This suggests that if other things are equal the queue with smaller variability should be served first if we are concerned about the overruns.

We next evaluate the mean sojourn time, $\bar{S}_i$, in queue $i$, $i = 1, 2$, assuming that it gets served completely in the scheduled slot. We use eqs. (14), (16), (17), and (28) and some straightforward but tedious algebra to obtain

$$\bar{S}_1 = a_1 + b_1 + \frac{T}{2}(1 + \rho_1), \tag{38}$$

and

$$\bar{S}_2 = a_2 + b_2 + \frac{T}{2}(1 + \rho_2) + \frac{\alpha_1 + \gamma_1 T}{T}(1 + \rho_2). \tag{39}$$

Comparing eqs. (38) and (39) with eq. (7), we see that the mean value, $\bar{S}_1$, of $S_1$ is not affected by the gating mechanism. However, $\bar{S}_2$ does depend on the gating. If we use the suffixes 1 and 2 to denote gatings G1 and G2, respectively, then

$$\bar{S}_{22} - \bar{S}_{21} = \frac{\alpha_1 + \gamma_1 T}{T}(1 + \rho_2) - a_1 - \rho_1 T. \tag{40}$$

Under the assumption that the probability of overrun is very small, the above difference is negative. Thus the mean sojourn time under gating G2 is smaller than under G1.

### *3.4 Numerical results*

In this section we use the analysis of Sections 3.1 through 3.3 to compute the probability of overrun and the sojourn time distributions for some special cases of the simple clocked schedule described earlier. For these numerical results we used the algorithm in Ref. 3 for inverting Laplace transforms. When the underlying function is not

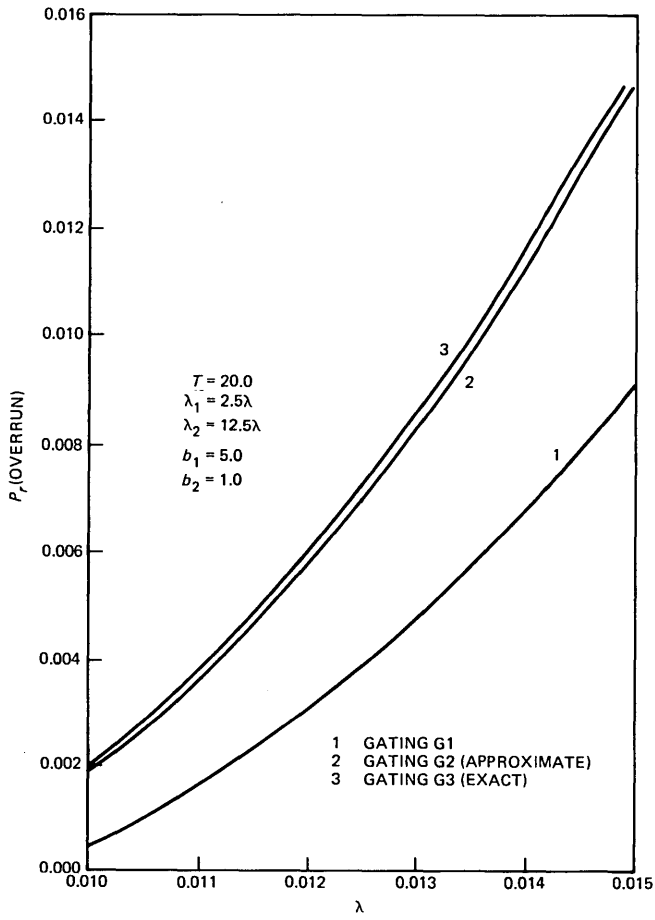



Fig. 7—$P_r${OVERRUN} as a function of load.

smooth enough, it may be more appropriate to discretize the problem and then use one of the routines for inverting the generating functions.

Suppose $m = 2$ and the overhead is zero. The service times are deterministic, 5 ms for queue 1 and 1 ms for queue 2. The length of a time slot is $T = 20$ ms. Finally, $\lambda_1 = 2.5\lambda$ and $\lambda_2 = 12.5\lambda$. Figure 7 shows the probability of overrun as a function of $\lambda$ for gating $G_1$, gating $G_2$ (exact), and gating $G_2$ (approximate). For gating G2 the exact and approximate results agree very well. For the selected values of the parameters the results for gating G1 and G2 differ significantly. The probability of overrun using gating G2 is higher than that using G1. This can be explained by the higher variance, as shown in Section 3.3. Alternatively, consider the work in queue 1 and queue 2 to be done in a time slot (see Fig. 8). For queue 1 the number of jobs to be served is

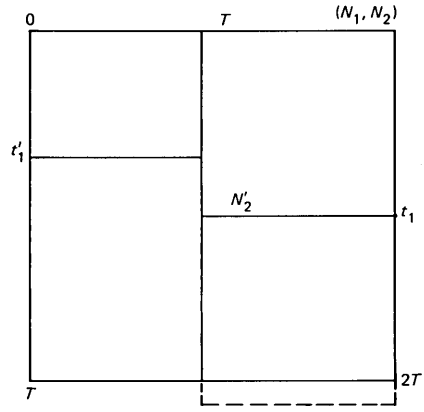



Fig. 8—Work content in queue 2.

the number that arrived in time $T$ for both gating mechanisms. The number of jobs to be served in queue 2 is the number that arrived in time $T$ for gating G1 and in time $T - t_1' + t_1$, where $t_1'$ and $t_1$ are, respectively, times to complete all work in queue 1 in the preceding and current intervals. The mean value of $T - t_1' + t_1$ is $T$, but because of the randomness in $T - t_1' + t_1$, the amount of work arriving at queue 2 in that time interval has a larger tail than that of the work arriving in time $T$. This explains the larger probability of overrun. This also indicates that the difference in the two gating mechanisms is related to the variability in $t_1'$ and $t_1$, which corresponds to the variability in the work arriving at queue 1.

Next we reverse the order of service within a time slot. For gating G1 the probability of an overrun is not affected by this. However, for gating G2 this does change the probability of an overrun. Figure 9 shows the probability of an overrun as a function of $\lambda$ for gating G1, gating G2 with order (1, 2), and gating G2 with order (2, 1). We see considerable reduction in the probability of an overrun with order (2, 1). This is expected because queue 2 has less variable work load than queue 1 has. Also, as expected, the difference between gating G1 and G2 is much smaller with order (2, 1) than with order (1, 2).

We further investigate the effects of the order of service within a time slot by considering an example with three queues ($m = 3$). The service times and overheads are deterministic. The parameters are as shown in Table IV. Queues 2 and 3 have the same parameters, while queue 1 has more variable work load than either queue 2 or 3. The probability of an overrun with gatings G1 and G2 and orders (1, 2, 3), (2, 1, 3), and (2, 3, 1) are shown in Table IV. Once again, we observe that the probability of an overrun decreases as we push the more variable queue down in the priority order.

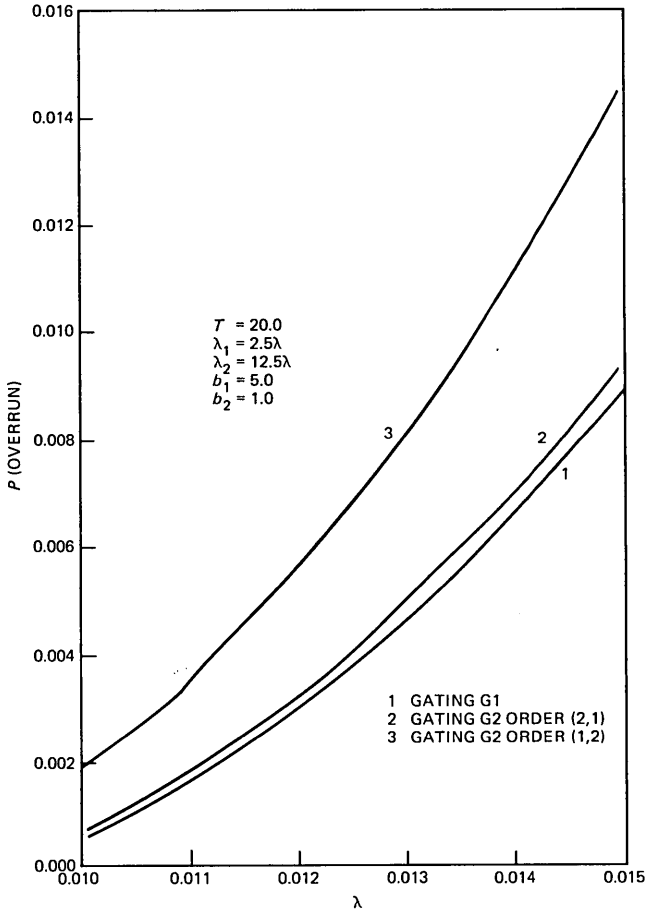



Fig. 9—$P$[OVERRUN] as a function of load.

Thus, from the moments considerations and from numerical results we have the following guidelines when the probability of an overrun is of concern:

1. The difference between the two gating mechanisms is greater when the queues with more variable work load are served earlier in a time slot.

2. The probability of an overrun is smaller if queues with more variable work load are pushed down in the priority order.

Of course, in most clocked schedules the differences in the time criticality of various tasks determine the order of service. The above guidelines, however, are useful when deciding order of service among tasks of comparable time criticality.

We next evaluate the sojourn time distributions numerically. The

Table IV—Probability of overrun

| Number of Queues = 3, $T$ = 20.0 Parameters | | | |
|---|---|---|---|
| Queue | Arrival Rate | Service Time | Overhead |
| 1 | 1.0 | 0.2 | 1.0 |
| 2 | 2.0 | 0.1 | 1.0 |
| 3 | 2.0 | 0.1 | 1.0 |
| | $P${OVERRUN} | | |
| Schedule | Gating G2 | Gating G1 | Gating G2 Approximation (First Order) |
| (1, 2, 3) | 0.027 | 0.015 | 0.025 |
| (2, 1, 3) | 0.023 | 0.015 | 0.022 |
| (2, 3, 1) | 0.021 | 0.015 | 0.020 |

case considered here is $m = 2$, no overhead, and deterministic service times. The parameters are $\lambda_2 = 0.05$, $b_2 = 2.0$, $\lambda_1 = 0.05$, and $b_1 = 1.0$. In Fig. 10 we have complementary sojourn time distribution, $1 - S_2(t)$, for queue 2 under gatings G1 and G2. As indicated earlier the delay for queue 2 with gating G1 is larger than with gating G2.

## IV. GENERALIZATIONS AND COMPLEX SCHEDULES

This section contains various generalizations of the basic methodology described in Section III. Some of these generalizations are aimed at relaxing underlying assumptions, while the others look at more complex schedules. To avoid presenting a lot of tedious and repetitive algebra we will only outline the derivations in this section. The details are similar to those in Section III.

### *4.1 The effects of queues feeding downward*

In Section III we assumed that the arrivals to different queues form independent Poisson processes. In many applications an arrival (job) may complete one task and immediately request another. Thus some of the task queues may feed the others. In typical situations the high-priority queues feed downward within an interval. That is, when a job is served in a high-priority queue, it may create an arrival directly to the internal queue of another task still to be served in that interval. Here, we indicate how to extend the results of Section III to include such feed-downs.

The basic clocked schedule is still the same as in Section III, but the arrival processes are now more general. Queue $i$ still has an *exogenous* Poisson arrival process at rate $\lambda_i$, $i = 1 \cdots m$. However, when a job is served in queue $i$, it creates an arrival to the internal

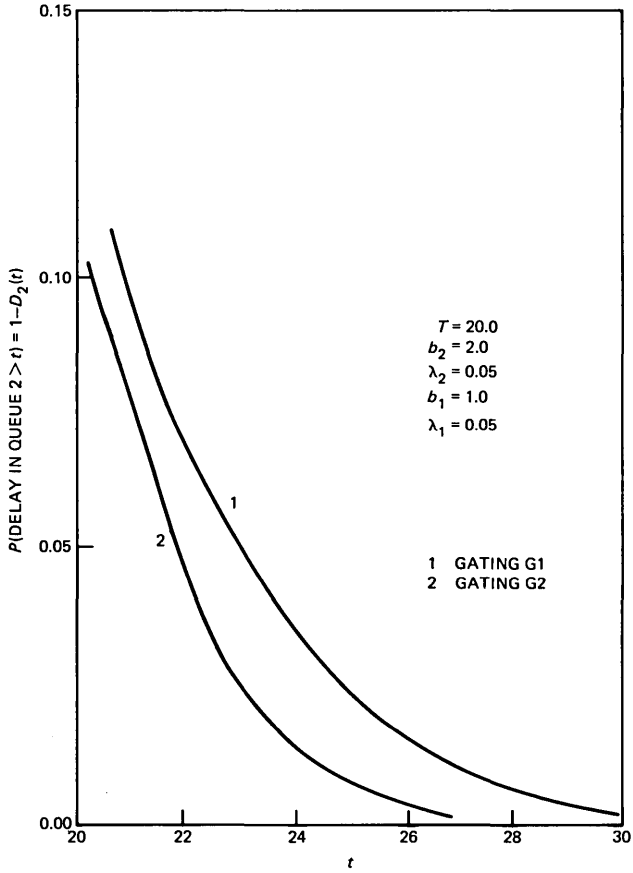



Fig. 10—Delay distribution in queue 2.

queue $j$ with probability $p_{ij}$. We assume

$$p_{ij} = 0 \quad j \leq i, \tag{41}$$

and

$$\sum_{j>i} p_{ij} \leq 1. \tag{42}$$

The Poisson arrivals may represent either genuinely exogenous arrivals or those generated by service completions in other queues in previous intervals. The Poisson approximation seems reasonable when the number of intervals between the completion of a task and the generation of a new one is large.

We consider the overrun strategy S1 and gating G2 (gating G1 is easier to analyze) and obtain the joint transform $\tilde{G}(s_1, s_2, \cdots s_m)$ of

$t_1, \cdots, t_m$. This can be done recursively as follows: We first obtain $\tilde{G}$ in terms of $\tilde{P}$ where

$$\tilde{P}(z_1, \cdots, z_m) = E[z_1^{N_1} z_2^{N_2} \cdots z_m^{N_m}].$$

We then express $\tilde{P}$ in terms of $\tilde{G}$ with the last argument being zero. Since $\tilde{G}(0, 0, \cdots 0) = 1$, we get both $\tilde{G}$ and $\tilde{P}$ by iterating on these two relations. We implicitly make the assumptions of the type made in Section III. Our expressions below are approximations that work when the probability that queue $i$ overruns is very small, $i = 1, \cdots, m - 1$.

For $\underline{z} = (z_1, z_2, \cdots, z_m)$, let $\phi_i(\underline{z}) = -\lambda_i(1 - z_i)$, $i = 1, \cdots m$. For $\underline{s} = (s_1, s_2, \cdots s_m)$, let

$$A_{i,m}(\underline{s}) = 1, \qquad i = 1, \cdots m,$$

$$B_m(\underline{s}) = s_m,$$

$$C_m(\underline{s}) = \lambda_m(1 - \tilde{F}_m(B_m(\underline{s}))A_{m,m}(\underline{s})),$$

and, for $i \leq j \leq m - 1$,

$$A_{i,j}(\underline{s}) = A_{i,j+1}(\underline{s})[1 - p_{i,j+1} + p_{i,j+1}\tilde{F}_{j+1}(B_{j+1}(\underline{s}))],$$

$$B_j(\underline{s}) = s_j + C_{j+1}(\underline{s}) + B_{j+1}(\underline{s}),$$

and

$$C_j(\underline{s}) = \lambda_j[1 - \tilde{F}_j(B_j(\underline{s}))A_{j,j+1}(\underline{s})].$$

Then

$$\tilde{G}(\underline{s}) = \prod_{i=1}^{m} \tilde{H}_i(B_i(\underline{s})) \cdot \tilde{P}(\tilde{F}_1(B_1(\underline{s}))A_{1,1}(\underline{s}), \cdots, \tilde{F}_m(B_m(\underline{s}))A_{m,m}(\underline{s})), \quad (43)$$

and

$$\tilde{P}(\underline{z}) = e^{T\sum_{i=1}^{m}\phi_i(\underline{z})}\tilde{G}(\phi_2(\underline{z}), \cdots, \phi_m(\underline{z}), 0). \quad (44)$$

We next study the effect of these feed-downs numerically. We consider the case $m = 2$, no overhead, and deterministic service times with $b_1 = b_2 = 1.0$. Let $p_{12} = p$. We consider six cases with $p$ varying from 0 to 1 in steps of 0.2. The exogenous arrival rate $\lambda_2$ in queue 2 is adjusted so that

$$\lambda_2' = \lambda_2 + p\lambda_1 = \lambda_1,$$

that is,

$$\lambda_2 = \lambda_1(1 - p)$$

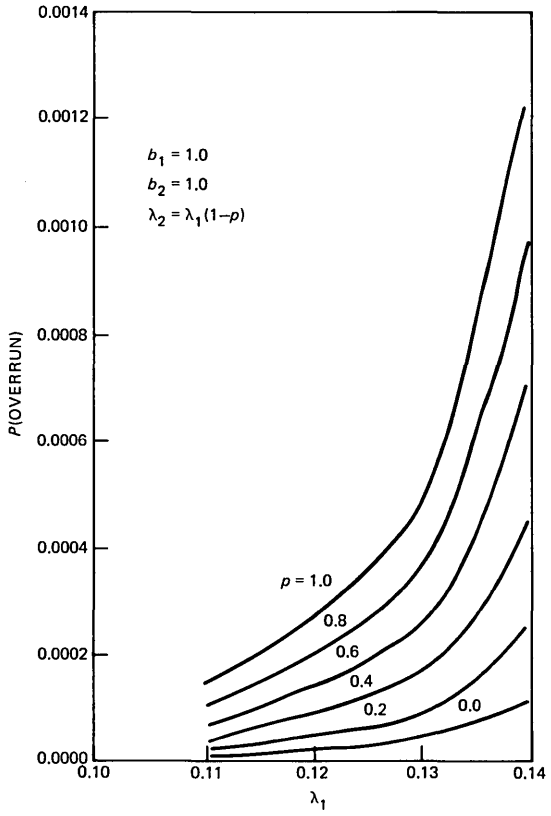



Fig. 11—$P$[OVERRUN] with feed-down.

for all six cases. $\lambda_1$ is varied from 0.1 to 0.15. The case $p = 0$ corresponds to no feed-down. Increasing $p$ corresponds to increasing feed-down and hence increasing correlation between the queues in an interval. The probability of overrun for each of these six cases is presented in Fig. 11. At large $p$, the effect of feed-down is clearly very significant.

### *4.2 Complex schedules*

In Section III we analyzed a simple clocked schedule in which each high-priority queue is scheduled in each interval. Although such schedules occur in practice, in most applications a subset of the set of high-priority tasks is scheduled in each interval. This subset changes from one time interval to the next. The entire schedule may repeat after $N_{\text{period}}$ time slots. Such a schedule is a clocked schedule with time slot length $T$ and schedule period $N_{\text{period}}$. Figure 12 illustrates a schedule with period 2.

In theory we can extend the results of Section III to analyze a

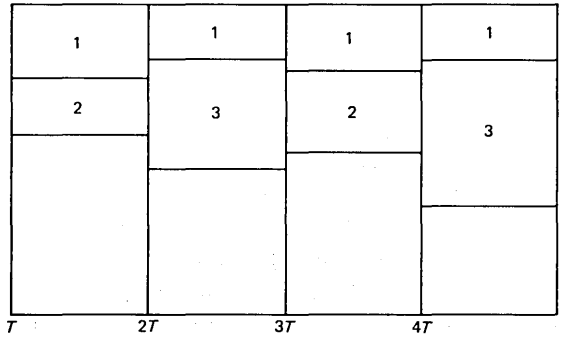



Fig. 12—A clocked schedule with period $2T$.

schedule with period $N_{\text{period}}$. For overrun strategy S1 and gating mechanism G1, the analysis is straightforward and can be carried out individually for each time slot within a period. For gating G2 with overrun strategy S1, note that

$$\{(N_{nN_{\text{period}}+1,1}, N_{nN_{\text{period}}+1,2}, \cdots, N_{nN_{\text{period}}+1}, m{:}n \geq 1)\}$$

is a time-homogeneous Markov chain and its balance equations can be written down as in Section III. We can also use the techniques of Section III to obtain transforms $\tilde{G}$ for each time slot in a period in terms of $\tilde{P}$, the joint generating function of the steady-state numbers in $m$ queues at the beginning of a period.

However, for large $N_{\text{period}}$ both the analysis and numerical evaluations may be difficult because of the dimensionality. Moreover, we need individual analysis for each schedule. When such difficulties arise, the approximation discussed below may be an appropriate alternative. This approximation applies uniformly to a wide variety of complex schedules and, for each slot, requires the same computational effort as for the simple schedule of Section III.

In order to understand this approximation consider an interval in which $m_1$ of the $m$ high-priority queues are scheduled. For simplicity we number them from 1 through $m_1$, the order in which they are scheduled for service. Let $T$ denote, as before, the length of the interval and, for $i = 1, 2 \cdots m_1$, let $T_i$ denote the nominal time between service initiations for queue $i$ (that is, if all queues complete service at their expected times, then the time between two successive starts of service to queue $i$ is $T_i$). For the simple schedule of Section III, $T_i = T$ *for* $i = 1, \cdots m$. For the schedule in Fig. 12, $T_1 = T$, $T_2 = T_3 = 2T$. Let

$$T_i' = T_i - \sum_{j=1}^{i-1} (a_j + \rho_j T_j). \tag{45}$$

Then the approximation for $\tilde{P}$ is given by

$$\tilde{P}(z_1, z_2, \cdots z_{m_1}) = e^{-\sum_{i=1}^{m_1} \lambda_i T_i (1 - z_i)}. \tag{46}$$

The relations expressing $\tilde{G}(s_1, s_2, \cdots s_{m_1})$ and $\tilde{S}_i(s)$ in terms of $\tilde{P}$ remain the same as in Section III.

Recall that in Section III we calculated the probability of overrun for $m = 3$. The parameter values and the probability of overrun are given in Table IV. For the same parameter values we calculated the probability of overrun for gating G2 using the approximation described above. The results given in the last column indicate that the approximation works well in this case. A little reflection suggests that for schedules with longer periods, the approximation should be even better.

### *4.3 Other overrun strategies*

In Section III we analyzed clocked schedules with overrun strategy S1. In theory, it is possible to extend these results to the schedules using overrun strategies S2 and S3. For strategy S2, $\{(N_{n1}, N_{n2}, \cdots N_{nm}) : n \geq 1\}$ is still a time-homogeneous Markov chain. Its balance equations can be written down easily, but they do not have the triangular structure of those for strategy S1. Thus, these equations cannot be solved recursively. They can, however, be solved as a system of linear equations after suitably discretizing the underlying distributions. For strategy S3 an overrun in one time slot will reduce the time available in the following slot. In this case $\{N_{n1}, N_{n2}, \cdots N_{nm}, U_n) : n \geq 1\}$ is a Markov chain, where $N_{ni}$ is the number in the external queue $i$ when the server visits queue 1 for the $n$th time and $U_n$ is the time remaining in that interval. Again, the steady-state distribution of this Markov chain can be obtained by solving a system of linear equations obtained by suitably discretizing the underlying distributions.

The numerical complexity involved in the above procedures may preclude their use for all but small values of $m$. For large $m$, some approximation or relatively easy to obtain bounds are needed. Note that for strategies S2 and S3 the probability of an overrun is bounded below by the probability of an overrun for strategy S1, because strategy S1 ignores the work carried over from an overrun interval into the next. If the amount of overrun is probabilistically small this bound must be quite close.

For strategy S3 we can also get an approximation if we assume that overrun does not extend beyond one clock interval. This approximation is obtained by adding a dummy queue, say queue 0, at the beginning of each schedule interval. The overhead in this queue represents the work carried over from the preceding interval. The

Table V—$P\{\text{OVERRUN}\}$ for strategy S3 with $T = 20.0$, $\lambda_1 = 0.1$, $b_1 = 1.0$, and $b_2 = 5.0$

| | $P\{\text{OVERRUN}\}$ | |
|---|---|---|
| $\lambda_2$ | Approximation $S_1$ | Approximation Using Dummy Queue |
| 0.025 | 0.003278 | 0.003427 |
| 0.05 | 0.023526 | 0.026999 |

arrival rate to this queue is assumed to be zero. We then use the following iterative procedure:

Step 1—Assume that the overhead in queue 0 is 0.

Step 2—Using the results of Section III for strategy S1, obtain the distribution of the carried over work, $(t_m - T)^+$.

Step 3—Assume that the overhead in queue 0 has the distribution of $(t_m - T)^+$ obtained in Step 2.

Repeat Steps 2 and 3 until two successive iterations give "close" values for the distribution of $(t_m - T)^+$.

Numerical values of $P\{\text{OVERRUN}\}$ calculated using strategy S1 and the approximation for strategy S3 are given in Table V. The parameter values are $m = 2$, no overhead, deterministic service times, $b_1 = 1.0$, $b_2 = 5.0$, $\lambda_1 = 0.1$, and $\lambda_2 = 0.025$ and 0.05.

## V. CONCLUSIONS

We presented a methodology to analyze the performance measures for the high-priority tasks in clocked schedules. The analysis was exact for a simple clocked schedule. For more complex clocked schedules this analysis formed the basis for easy to use approximations.

## REFERENCES


1. A. A. Fredericks, B. L. Farrell, and D. F. DeMaio, "Approximate Analysis of a Generalized Clocked Schedule," AT&T Tech. J., this issue.
2. M. H. Ackroyd, "Numerical Computation of Delays in Clocked Schedules," AT&T Tech. J., this issue.
3. D. L. Jagerman, "An Inversion Technique for the Laplace Transform With Application to Approximation," B.S.T.J., 57 (March 1978), pp. 669–710.
4. L. Kleinrock, *Queueing Systems*, New York: Wiley, 1975.
5. A. Fredericks, "Analysis and Design of Processor Schedules for Real Time Applications," *Applied Probability-Computer Science, The Interface I*, edited by R. Disney and T. Ott, Boston, MA: Birkhäuser, 1982, pp. 433–50.
6. D. Manfield and P. Trans-Gia, "Queueing Analysis of Scheduled Communications Phases in Distributed Processing Systems," Performance 81, Proc. of 8th Int. Symp. on Comp. Perf. Mod., Meas. and Eval., Amsterdam, November 4–6, 1981, edited by F. J. Kylstra, Amsterdam: North-Holland, pp. 233–50.
7. A. Fredericks, "Analysis of a Class of Schedules for Computer Systems With Real Time Applications," *Performance of Computer Systems*, edited by M. Aroto, A. Butrimenko, and E. Gelenbe, Amsterdam: North-Holland, 1979, pp. 201–16.

8. H. Jans, "On Queueing Systems With Clocked Operation and Priorities," Proc. 10th ITC, Montreal, 1983, Section 4.4a, Paper No. 4.

## AUTHOR

**Bharat T. Doshi,** B. Tech. (Mechanical Engineering), 1970, I.T.T. Bombay; Ph.D. (Operations Research), 1974, Cornell University; AT&T Bell Laboratories, 1979—. Before joining AT&T Bell Laboratories Mr. Doshi was an Assistant Professor at Rutgers University. At AT&T Bell Laboratories his technical work includes modeling and analysis of processor schedules, overload controls, and capacity estimation. His research interests are queueing theory and scheduling theory applied to the performance analysis of computer, communication, and production systems. Member, IEEE, ORSA; Associate Editor, OR Letters.

**AT&T TECHNICAL JOURNAL** is abstracted or indexed by *Abstract Journal in Earthquake Engineering, Applied Mechanics Review, Applied Science & Technology Index, Chemical Abstracts, Computer Abstracts, Current Contents/Engineering, Technology & Applied Sciences, Current Index to Statistics, Current Papers in Electrical & Electronic Engineering, Current Papers on Computers & Control, Electronics & Communications Abstracts Journal, The Engineering Index, International Aerospace Abstracts, Journal of Current Laser Abstracts, Language and Language Behavior Abstracts, Mathematical Reviews, Science Abstracts (Series A, Physics Abstracts; Series B, Electrical and Electronic Abstracts; and Series C, Computer & Control Abstracts), Science Citation Index, Sociological Abstracts, Social Welfare, Social Planning and Social Development*, and *Solid State Abstracts Journal*. Reproductions of the Journal by years are available in microform from University Microfilms, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.


AT&T