Performance Management of Multiple Access Communication Networks

Suk Lee, *Member, IEEE*, and Asok Ray, *Senior Member, IEEE*

Abstract—**This paper focuses on conceptual design, development, and implementation of a performance management tool for computer communication networks to serve large-scale integrated systems. The objective is to improve the network performance in handling various types of messages by on-line adjustment of protocol parameters. The techniques of perturbation analysis of Discrete Event Dynamic Systems (DEDS), stochastic approximation (SA), and learning automata have been used in formulating the algorithm of performance management. The efficacy of the performance management tool has been demonstrated on a network testbed.**

The conceptual design presented in this paper offers a step forward to bridging the gap between management standards and users' demands for efficient network operations since most standards such as ISO (International Standards Organization) and IEEE address only the architecture, services, and interfaces for network management. The proposed concept for performance management can also be used as a general framework to assist design, operation, and management of various DEDS such as computer integrated manufacturing and battlefield C^3(Command, Control, and Communications).

I. Introduction

MULTIPLE-access computer networks are designed to provide communications between spatially distributed heterogeneous devices via common media and flexibility for changes in the system configuration. They are, therefore, well suited to serve large-scale integrated systems like banking and brokerage, battlefield command and control, autonomous manufacturing and processing plants, and advanced aircraft and spacecraft. Since the system requirements may widely vary according to the specific application, a computer network must be tailored at the design stage by selection of appropriate protocols and assignment of the default parameters [1]–[3]. However, since the condition under which a network actually operates may change from that considered at the design stage, control and management actions are required to adjust the network parameters so that the design and operational objectives are satisfied. For example, under crisis situations, a military communication network may experience a traffic pattern which is entirely different from that under the normal peace-time operations. Similarly, in an aircraft control system network, the AI-based decision support systems for vehicle management are expected to generate a significant amount of additional traffic when dealing with component failures or inflicted damage. Therefore, the network must always adapt to the dynamic environment especially if it is required to serve a large collection of heterogeneous users.

The responsibility of adapting the network to the dynamical environment belongs to network management which aims to maintain reliable, flexible, and efficient operations. The discipline of network management is a relatively young field, and its importance is being realized as the number and diversity of the subscribers increase. Basic rules have been reported on the architecture, service definition, protocol standardization, and distribution of network management functions [4]–[6]. Several standards for network management have been already published or under preparation as an important part of the integrated protocol suite [2], [3], [7], [8]. However, analytical work has been largely limited to the classical areas of routing and flow control [9]. Researchers have not apparently addressed the key issue of how to accomplish the network management tasks in real time as problems or disruptions arise either in the network operations or in the process that is served by the network. This has happened partly because the development of a methodology for network management is beyond the scope of standardization. Nevertheless, from the users' point of view, network management is extremely crucial for maintaining uninterrupted operators as many important functions are dependent on the communication services provided by the network.

The major components of network management are fault management, configuration management, and performance management. As its name implies, *fault management* is responsible for detection, isolation, and recovery from component failures and inflicted damage. *Configuration management* is related to network initialization and accommodation of any network configuration changes including those requested by fault management and performance management. *Performance management* is responsible for improving the network performance by adjusting the protocol parameters, and is critical for efficient operation of large-scale integrated systems that operate in a dynamic environment. Network operations with default settings of protocol parameters may not prove to be efficient under diverse operating conditions for a prolonged period of time. For example, a network for Computer Integrated Manufacturing (CIM) should be able to deliver various messages such as CAD (Computer Aided Design) file transfer, interpersonnel electronic mail, sensor and control signals within their time limits of delivery under

Manuscript received May 29, 1992; revised February 18, 1993. This work was supported in part by the National Science Foundation under Research Grant DDM-90-15173.

S. Lee is with the Department of Mechanical, Industrial, and Nuclear Engineering, University of Cincinnati, Cincinnati, OH 45221.

A. Ray is with the Department of Mechanical Engineering, The Pennsylvania State University, University Park, PA 16802.

IEEE Log Number 9212160.

changing traffic patterns caused by common events like arrival of new production orders and machinery breakdown. Even though individual protocols offer mechanisms to handle various types of messages efficiently, the key parameters of the protocol suite which determine the network performance are at the network operator's disposal. Usually, the operator adjusts these parameters on the basis of certain heuristics and his/her individual experience because there is no established relationship between protocol parameters and network performance. Also, no systematic approach to parameter adjustment exists.

This paper presents conceptual design, development, and implementation of a performance management tool for multiple-access computer communication networks. The objective is to improve the network performance by on-line adjustment of protocol parameters. To the best of the authors' knowledge, it is the first application of Perturbation Analysis (PA) of Discrete Event Dynamic Systems (DEDS) to multiple-access network protocols. Moroever, the technique of PA has been combined with Stochastic Approximation (SA) and Learning Automata (LA) to formulate the performance management algorithm. This paper is organized into six sections, including the Introduction and two Appendices. Section II presents the main theme of the proposed performance management tool, including the concepts of PA, SA, and LA. Analytical formulation of the tool for the specific case of a token bus protocol is presented in Section III. Details of implementation of the performance management algorithm on a network testbed are described in Section IV. The test results are presented and discussed in Section V. Finally, the paper is summarized and concluded in Section VI along with recommendations for future research. The priority mechanism of the token bus protocol considered here is briefly in Appendix A, and an example of perturbation analysis of timer setting changes for the protocol is presented in Appendix B.

II. Performance Management

The role of performance management is to manipulate the adjustable protocol parameters in real time so that the network can adapt itself to a dynamic environment. Performance management is divided into two tasks: i) *performance evaluation* to find how changes in protocol parameters affect the network performance measure; and ii) *decision making* on how to adjust the protocol parameters. The first task is essentially equivalent to finding a relationship between the network performance and the protocol parameters, and may be required to estimate the network performance at some points in the neighborhood of the current parameter settings. The second task is to decide the direction and magnitude of the parameter adjustment vector, i.e., what are the next settings for improved network performance, utilizing pieces of information provided by the first task, and the performance history.

Performance Evaluation: The analytical techniques, such as queueing theory [10], often require unrealistic assumptions like Poisson arrival, and tend to be mathematically untractable as the structure of the performance measure becomes complex. Furthermore, network traffic statistics such as message arrival process and distribution of message length, which are required as inputs to the analytical model, are very difficult to estimate on-line. On the other hand, discrete-event simulation [11] is a viable alternative to analytical techniques. A major advantage of simulation over any analytical technique is that a DEDS can be modeled with much less stringent assumptions, and more complex performance measures can be handled with relative ease. However, discrete-event simulation usually suffers from signficant computational burden because a single simulation run represents only one realization of a stochastic process. In order to obtain an accurate performance estimate under a given set of parameters, several independent runs (or a lengthy run if the process is ergodic) are needed, and these runs should be repeated for individual sets of pertinent parameters. In order to avoid the estimation of network traffic statistics which are still required, one can record time of arrival and length for each message and feed this information into a simulation model. However, this requires a large amount of information transfer from each individual station to the performance manager, which may degrade the overall network performance.

Over the last decade, Ho and his colleagues ([12], [13], and references therein) have developed the technique of Perturbation Analysis (PA) to circumvent the difficulties of conventional analysis and simulation in DEDS. PA estimates the DEDS performance under perturbed conditions (with different parameter values) by observing the sequence of events occurring over a period of time in the nominal (i.e., unperturbed) system. In fact, PA constructs parts of event sequence for the perturbed system based on the nominal one. This approach has a computational advantage over repetitive simulation runs, especially when no analytic technique is available. When the effects of n parameters on a performance measure are to be evaluated, the conventional discrete-event simulation needs $n + 1$ runs (one with the nominal parameters and n runs, each with one perturbed parameter and the remaining nominal values). On the other hand, PA needs only one run because it calculates the performance measure of the perturbed system based on the inherent information from the simulation with the nominal parameter. Therefore, the ratio of computation time can be approximately 1 to $n+1$ if the processing time for PA algorithms is negligible compared to that for discrete-event simulation. For performance management, PA is very suitable because this technique can directly utilize on-line observation of events. This does not require any identification of statistical parameters of the network traffic and is computationally more efficient than discrete-event simulation. Furthermore, PA still retains the inherent advantages of simulation over analytical techniques.

Decision Making: This task requires parameter optimization, and can be accomplished numerically by Stochastic Approximation (SA) which utilizes random measurements over a finite period of time to estimate the finite difference quotient of the performance measure with respect to decision variables [14]. However, since the performance measure itself is a random variable (with an unknown distribution) in this situation, the estimated quotients have a nonzero variance at every point. Consequently, the SA technique has to reduce

its step size as the extremal point is approached. In many situations, however, more than one optimization algorithm may be required because any individual algorithm is likely to be efficient only in some region of the protocol parameter settings under given statistics of network traffic. An additional level of decision making is needed to select the most appropriate optimization algorithm according to the current parameter settings and traffic statistics. This approach is likely to enhance the efficiency and credibility of performance management in a dynamic operating environment whose characteristics are unknown or partially known. Techniques like Learning Automata (LA) can be used for decision making at the upper level for selecting the most efficient and credible optimization algorithm based on past performance [15].

A learning automaton consists of a set of actions, a corresponding set of action probabilities, and a reinforcement scheme. The action probabilities are updated by the reinforcement scheme according to the response from the environment which reacts to the action of the learning automaton. A Performance Evaluator (PE), as a part of the environment, is required to interpret the response. If the interpretation is favorable, then the probability of the chosen action is increased and those for other actions are decreased. By repeating this process, the learning automaton can select the best action under the current environment. In a dynamic environment, the performance management algorithm must know whether the network traffic statistics have changed because: i) the optimization algorithm is likely to have its own ability of adaptation such as reduction of the step size in SA; and ii) a change in network traffic may mislead the learning automaton in evaluating the performance of the optimization algorithms. Therefore, whenever any change in network traffic statistics takes place, the step size of the optimization algorithm and the PE may have to be reset.

III. Formulation of a Performance Management Procedure

The proposed performance management tool is based on the principles of Perturbation Analysis (PA), Stochastic Approximation (SA), and Learning Automata (LA), and its overall structure is shown in Fig. 1. The Linear Token Passing Bus (LTPB) protocol [8] has been selected to demonstrate the efficacy of this performance management tool for adjustment of a Token Holding Timer (THT) and three Token Rotation Timers (TRT1, TRT2, and TRT3) in real time on the basis of measured data. The operating principle of LTPB and its priority mechanism are described in Appendix A.

A. Performance Evaluation Via Perturbation Analysis

The knowledge of the relationship between a selected performance measure and the pertinent protocol parameters is a critical requirement for performance management. In this case, the behavior of the LTPB protocol with perturbed parameter settings under the same stochastic realization is constructed from a single observation with nominal parameters. The settings of the four timers of LTPB directly influence data latency (i.e., the time interval between the instant a message enters the source station's queue to the instant the last bit of the message is received at the destination station), whose statistics are often one of the most dominant factors in network performance. How to assess the effects of change in timer settings on network operations is illustrated by a simple example of an LTPB network in Appendix B.

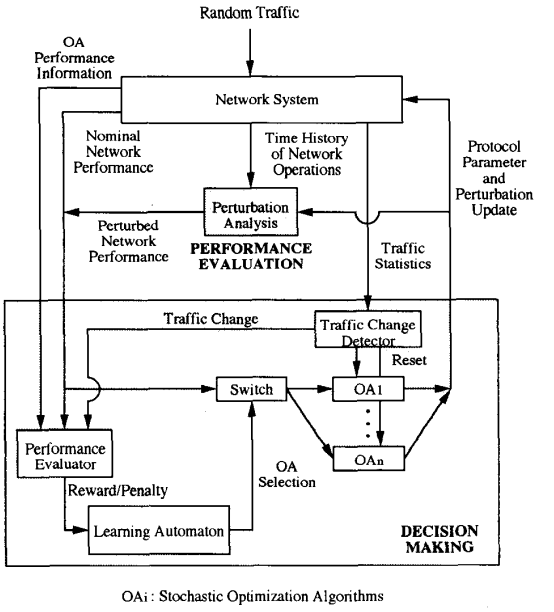

Fig. 1. Overall structure of performance management.

The PA Algorithm for the LTPB Protocol: To reduce the complexity of the problem, we assume that there is no queue saturation on both nominal and perturbed paths. This assumption holds good for the majority of network applications because the network is usually designed to operate sufficiently below its full load and a safe margin is allowed in the allocation of queue capacity. The PA algorithm consists of two parts: i) detection of any possible difference between the nominal path and a perturbed path; and ii) construction of the portions of the perturbed path which differ from the nominal one. No additional event scheduling in discrete-event simulation is required for either of these two parts. This is critical because a major share of computations in discrete-event simulation results from event scheduling and its execution.

Detection of Difference between Nominal and Perturbed Paths: A portion of the perturbed path begins to differ from the nominal one when the number of message transmissions from a queue on a particular token reception are different from each other. (Transfer of the right to transmit between the queues of a station is also considered as a token reception.) In order to establish the test conditions for a different number of message transmissions on the perturbed path, several notations are introduced.

m: Number of messages just transmitted from the current queue on the nominal path, $m = 0, 1, 2, \cdots$.

q: Number of messages waiting in the current queue after m transmissions on the nominal path, $q = 0, 1, 2, \cdots$.

L_k: Transmission time of the kth message, $k = 1, 2, \cdots, m$.

T: Nominal token circulation time of the current queue, i.e., the time interval between the previous and current token reception instants.

THT: Nominal length of THT.

ΔTHT: Perturbation in THT, $THT + \Delta THT > 0$.

TRT_i: Nominal length of TRT$_i$.

ΔTRT_i: Perturbation in TRT$_i$, $TRT_i + \Delta TRT_i > 0$.

The following tests must be executed before passing the token to the next queue after m message transmissions.

TESTO [test for priority class 0]

Case a) $\Delta THT > 0$ and $m > 0$:

$$\sum_{k=1}^{m} L_k < (THT + \Delta THT) \quad \text{and} \quad q > 0$$

Case b) $\Delta THT < 0$ and $m > 1$:

$$\sum_{k=1}^{m-1} L_k \geq (THT + \Delta THT)$$

TESTi [test for priority class, $i, i = 1, 2, 3$]

Case a) $(TRT_i - T) > 0, \Delta TRT_i > 0$ and $m > 0$:

$$\sum_{k=1}^{m} L_k < (TRT_i + \Delta TRT_i - T) \quad \text{and} \quad q > 0$$

Case b) $(TRT_i - T) > 0, \Delta TRT_i < 0$ and $m > 1$:

$$(TRT_i + \Delta TRT_i - T) \leq \sum_{k=1}^{m-1} L_k$$

Case c) $(TRT_i - T) \leq 0$ and $\Delta TRT_i > 0$:

$$(TRT_i + \Delta TRT_i - T) > 0 \quad \text{and} \quad q > 0$$

Case d) $(TRT_i - T) > 0, \Delta TRT_i < 0$ and $m = 1$:

$$(TRT_i + \Delta TRT_i - T) \leq 0.$$

TEST0(a) is applicable if a priority class 0 queue can transmit more message(s) on the perturbed path in addition to m transmitted messages on the nominal path. Additional transmission is possible only when THT is increased and there has been at least one message transmission on the nominal path. (No transmission on the nominal path implies that the queue is empty upon token reception.) Further, the increased THT should be long enough so that the perturbed THT is not expired even after m trnasmissions and the queue should not be empty for additional transmission(s). TEST0(b) is opposite to TESTO(a). If TESTO(b) is satisfied, then the number of transmissions on the perturbed path is less than the number m of transmissions on the nominal path. In this case, ΔTHT should be negative and the perturbed THT should expire during the $(m-1)$st transmission at the latest. TESTi(a) and TESTi(b) are largely equivalent to TEST0(a) and TEST0(b), respectively. TESTi(b) represents a slightly different situation compared to TEST0(b) in the following sense. TEST$_i$(b) could imply that no transmission is allowed on the perturbed path while there would be at least one transmission on the perturbed path for TEST0(b). TESTi(c) implies that some transmissions are possible on the perturbed path while there has been no transmission on the nominal path due to expiration of TRTi. If TESTi(d), which can be considered as a special case of TESTi(b), is satisfied, then no transmission is allowed on the perturbed path while one transmission was possible on the perturbed path.

Construction of the Perturbed Path: The construction of a perturbed path consists of three parts: maintenance of the perturbed queue status; calculation of the perturbed timer status; and propagation of the effects of perturbation on the instant of token reception. Records associated with a message such as generation time and message length are kept even after the message is transmitted on the nominal path. These records are discarded only after the message is transmitted on both nominal and perturbed paths. In this way, queue contents on the perturbed path are available for construction of the perturbed path.

For the priority level 0, THT status is independent of the token circulation time T since THT is always reset to its full value at each token reception. Therefore, upon a token reception, the remaining interval of THT on the nominal path $R(0)$ is always equal to THT during which the priority 0 queue can start to transmit its messages. On the perturbed path, the interval $R'(0)$ is

$$R'(0) = THT + \Delta THT.$$

For the lower-priority levels, the remaining interval of TRTi on the nominal path is equal to $R(i) = \max(TRT_i - T, 0)$ upon a token reception, where $\max(\cdot, \cdot)$ denotes the larger of the two arguments. Since the interval is dependent on T, the perturbation on the instants on the current and previous token receptions at a queue should be considered, which are denoted by ΔC and ΔP, respectively. The perturbation on the remaining time when the token is received, $\Delta R(i)$, is

$$\Delta R(i) = \begin{cases} \Delta P - \Delta C + \Delta TRT_i \\ \quad \text{if } (TRT_i - T) > 0 \\ \Delta P - \Delta C + \Delta TRT_i + TRT_i - T \\ \quad \text{if } (TRT_i - T) \leq 0. \end{cases}$$

The first case applies when the timer is not expired on the nominal path as shown in Fig. 2. The second case applies when the timer is already expired prior to the token reception as shown in Fig. 3. Then, at the token reception, the remaining interval on the perturbed path $R'(i)$, during which priority i transmissions can start, is:

$$R'(i) = \max(R(i) + \Delta R(i), 0).$$

ΔC for the next queue is yet to be obtained for construction of the perturbed path. The time $X(i)$, spent in transmitting messages from the current queue, is known from the nominal path. The time, $X'(i)$, spent for message transmissions on the perturbed path can be obtained on the basis of the perturbed queue status and the perturbed remaining time $R'(i)$. $X'(i)$ is calculated by summing up the transmission time, L_j, of the jth message from the perturbed queue either until the perturbed queue becomes empty or until the perturbed TRTi is expired

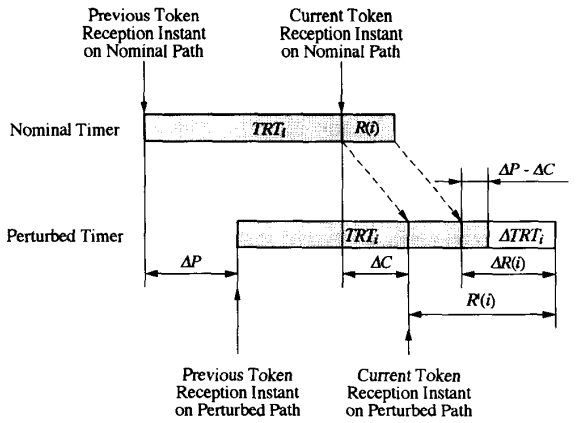

Fig. 2. Perturbed remaining interval of TRT_i (case 1).

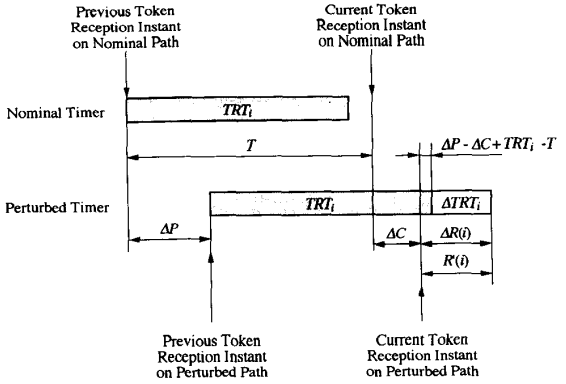

Fig. 3. Perturbed remaining interval of TRT_i (case 2).

i.e., $R'(i) - X'(i) \leq 0$. While calculating $X'(i)$, observations like data latency on the perturbed path can be recorded for estimating the perturbed performance.

Defining $\Delta X(i) = X'(i) - X(i)$, perturbation at the instant of token reception for the next queue is obtained as:

$$\Delta C(i+1) = \Delta C(i) + \Delta X(i)$$

where the indices i and $i+1$ indicate that the current and next queues are based on modulo 4, i.e., they range from 0 to 3.

Summary of the PA Algorithm: The algorithm for perturbation analysis of the LTPB procotol is summarized in four steps as delineated.

Step 1: Perform appropriate test (TEST0 and $\mathrm{TEST}i, i = 1, 2, 3$) just before passing the token to the next queue or station. Set the flag if any of the tests are satisfied.

Step 2: If the flag is not set, repeat Step 1 for the next queue or station. Otherwise, proceed to Step 3.

Step 3: Execute the construction procedure for the current queue before passing the token.

Step 4: Repeat Step 3 until all queues have $\Delta C = 0$ and $\Delta P = 0$ during a token circulation. If so, clear the flag and go to Step 1.

B. Stochastic Approximation

Although the Stochastic Approximation (SA) technique described in [14] has been proven to converge in a stochastic sense, it is known to have a slow convergence in many practical situations due to the fact that its step size is uniformly reduced regardless of the current value of decision variable $\boldsymbol{x}[k]$ (a 4×1 vector of timer settings in this paper) at the kth iteration. To circumvent this difficulty, the conventional algorithm is modified following Ho *et al.* [16]. This modified algorithm, hereafter referred to as the Modified Stochastic Approximation (MSA), can be written as follows.

$$\boldsymbol{x}[k+1] = \boldsymbol{x}[k] - \Gamma[k]\boldsymbol{f}(\boldsymbol{q}^{\pm}[k], \cdots, \boldsymbol{q}^{\pm}[k - m_w + 1]).$$

$\boldsymbol{\Gamma}[k]$ is a diagonal matrix whose nonzero elements are not restricted to be identically equal and are dependent on the number of sign reversals in the corresponding forward or backward finite-difference quotient $q_i^{\pm}[k]$. The ith diagonal element of $\boldsymbol{\Gamma}[k]$ is:

$$\gamma_i[k] = \max\left(c_i r^{e_i[k]}, l\right)$$

where c_i is a positive constant for the initial step size, $r \in (0, 1)$ is a reduction factor, $e_i[k]$ is an integer counter variable indicating the number of sign reveals in the ith quotient $q_i^{\pm}[k]$, and l is the lower bound common to all diagonal element. The forward or backward quotient of the observed performance $g(\boldsymbol{x}[k], \omega[k])$ at a sample point $\omega[k]$ is defined as:

$$q_i^{\pm}[k] = \frac{g(\boldsymbol{x}[k] + \Delta x_i[k]\boldsymbol{u}_i, \omega[k]) - g(\boldsymbol{x}[k], \omega[k])}{\Delta x_i[k]}$$

where $\boldsymbol{u}_i$ is the ith unit vector, a positive perturbation $\Delta x_i[k]$ for ith decision variable is used for forward difference, and a negative perturbation $\Delta x_i[k]$ for backward difference. The MSA algorithm also utilizes past quotients to smooth the adjustments in $\boldsymbol{x}[k]$. Function $\boldsymbol{f}$ takes a weighted average of m_w recent quotient vectors $\boldsymbol{q}^{\pm}[k] = \Sigma_{i=1}^{n} q_i^{\pm}[k]\boldsymbol{u}_i$. That is, the ith element of $\boldsymbol{f}$ is written as:

$$f_i(\boldsymbol{q}^{\pm}[k], \cdots, \boldsymbol{q}^{\pm}[k - m_w + 1]) = \frac{g(\boldsymbol{x}[k], \omega[k])}{x_i[k]} \sum_{j=1}^{m_w} \frac{w_j x_i[k-j+1] q_i^{\pm}[k-j+1]}{g(\boldsymbol{x}[k-j+1], \omega[k-j+1])}$$

where $\Sigma_{j=1}^{m_w} w_j = 1$ and $w_j \geq 0 \forall j$.

Remark: The basic idea behind the MSA algorithm is that the sign of the quotient changes more frequently as $\boldsymbol{x}[k]$ approaches its optimal point since the noise contained in the quotient is likely to determine the sign. The MSA algorithm adapts its step size based on the number of sign reversals in the quotient in contrast with uniform reduction in stochastic approximation. Weighted averaging function $\boldsymbol{f}$ serves to reduce the length of a period to measure the performace and avoid alternating directions of the parameter adjustments. With an appropriate choice of the window size m_w and weighting factors w_j, inherent noise in measurements may be reduced without sacrificing the speed of convergence. However, a rigorous proof of the convergence for MSA algorithms has not been established. This is apparently untractable because of the dependence of the step size on past history.

C. Learning Automata

For autonomous selection of the optimization algorithm, the Variable-Structure Stochastic Automaton (VSSA) [15] has been adopted because greater flexibility can be exercised within a smaller structure in comparison to those in fixed or deterministic settings. The reinforcement scheme in VSSA updates the action probabilities in discrete time based on the responses from the Performance Evaluator (PE). The next action of the automaton is selected on the basis of the updated action probabilities whose sum is equal to 1. A simplified VSSA is represented by the triple $\{\overline{\alpha}, \overline{\beta}, A\}$ where $\overline{\alpha} = \{\alpha_1, \alpha_2, \cdots, \alpha_a\}, \beta = \{\beta_1, \beta_2, \cdots, \beta_s\}$, and $A: \overline{\alpha} \times \overline{\beta} \rightarrow \overline{\alpha}$ is the reinforcement scheme. $\overline{\alpha}$ is the set of available actions (i.e., optimization algorithms); $\alpha[k]$ denotes the action at instant k. β is the set of responses (i.e., possible levels of performance of the current optimization algorithm) that are inputs to the automaton, and the response at instant k is denoted by $\beta[k]$.

The discrete reinforcement scheme for performance management has been formulated following the concept of discrete reward/penalty (DRP) automaton [17]. In this case, the automaton has two available actions (i.e., there is a choice between two alternative optimization algorithms) in response to five possible inputs, namely 0, 0.25, 0.5, 0.75, and 1, from the environment. Among these inputs $\beta_1 = 0$ indicates the most favorable response while $\beta_5 = 1$ is the most unfavorable one. The reinforcement scheme has $M + 1$ states (i.e., the action probability is allowed to assume one of the $M + 1$ values) where M is an even number greater than 2. The reinforcement scheme is presented:

$$\Delta p_1[k] = \begin{cases} \dfrac{2}{M} & \text{if } \alpha[k] = 1 \text{ and } \beta[k] = 0, \\ & \text{or if } \alpha[k] = 2 \text{ and } \beta[k] = 1 \\ \dfrac{1}{M} & \text{if } \alpha[k] = 1 \text{ and } \beta[k] = 0.25, \\ & \text{or if } \alpha[k] = 2 \text{ and } \beta[k] = 0.75 \\ 0 & \text{if } \beta[k] = 0.5 \\ \dfrac{-1}{M} & \text{if } \alpha[k] = 1 \text{ and } \beta[k] = 0.75, \\ & \text{or if } \alpha[k] = 2 \text{ and } \beta[k] = 0.25 \\ \dfrac{-2}{M} & \text{if } \alpha[k] = 1 \text{ and } \beta[k] = 1, \\ & \text{or if } \alpha[k] = 2 \text{ and } \beta[k] = 0 \end{cases}$$

and

$$p_1[k+1] = \begin{cases} \min\left(p_1[k] + \Delta p_1[k], 1\right) & \text{if } \Delta p_1[k] > 0 \\ p_i[k] & \text{if } \Delta p_1[k] = 0 \\ \max\left(p_1[k] + \Delta p_1[k], 0\right) & \text{if } \Delta p_1[k] < 0 \end{cases}$$

where $p_1[k]$ is the probability of selecting α_1. The probability of selecting α_2 is given as $p_2[k] = 1 - p_1[k] \forall k$.

IV. Implementation of the Performance Management Tool

The performance management algorithm has been implemented on a network testbed where the LTPB protocol [8] under consideration was emulated by two interacting application processes. The testbed is operated on the IEEE 802.4 token bus protocol in the environment of the (10 Mb/s broadband) Manufacturing Automation Protocol (MAP) [2]. The physical configuration of the testbed consists of a length of coaxial cable, a head-end remodualtor, and three hosts. Each host computer is equipped with two network cards from Industrial Networking Incorporated (INI) [18]. One of the hosts operates as a network management console for initial downloading of the protocols to the remaining two hosts. At each of these two hosts, a software package emulates a number of LTPB stations by generating, transmitting, and receiving messages which are essentially packets of the Association Control Service Element (ACSE) of MAP [2].

A. Implementation Strategy

Since the PA algorithm involves only logic and addition operations, it has been implemented in a distributed manner to be executed at each station to estimate the network performance under perturbations in each one of the four timer settings of the LTPB priority mechanism. Perturbation in the token reception instant is included in the token to be passed to the next station in the logical ring. In this distributed implementation, additional traffic due to management operations is expected to be significantly smaller than that for a centralized PA algorithm which requires information on contents of queues and timer status from all stations. On the other hand, the decision-making module that includes stochastic approximation and learning algorithms is centralized at a designated station, hereafter referred to as performance manager. Even though the network may have more than one station with partial or complete capability to execute decision-making functions, the centralized strategy allows only one active copy of each decision-making function. The rationale for selecting centralized decision making is that the distributed strategy, where every station could make decisions autonomously based only on its local performance, would result in inconsistency and conflict by having different timer settings over the network.

Network operations under the proposed performance management tool involve a series of iterations, which consist of an observation period and subsequent management actions. At the beginning, the performance manager broadcasts the initial timer settings and timer perturbation vectors to all stations. During an observation period, each station executes its own PA algorithm, and the performance manager monitors the messages flowing over the network. When the performance manager depicts that enough data have been collected, it waits for the token and then broadcasts the request for a performance report to all stations. Then, the performance manager passes the token without any further message transmission. Once this request from the performance manager is received, other stations interrupt their normal operations, prepare the performance reports, and transmit these reports as soon as the token is received. After one complete token circulation, the manager receives the token against and, by this time, reports from all other stations have been received. Then, the manager processes the reports, computes new timer settings, and broadcasts them with new timer perturbation vectors for the next iteration. Upon reception of the new settings and perturbation vectors, all stations set their corresponding variables and wait for the token to resume normal operations.

B. Implementation Details

In this implementation, a measure of the network performance has been formulated on the basis of observed data latency. Throughput of the network is excluded from the performance measure since network traffic is assumed to be below network capacity, which implies that all messages entering the network are eventually transmitted to their destination. In other words, throughput is equal to offered traffic. Therefore, the main focus of the performance measure is on data latency, i.e., how long it takes for a message to reach its destination relative to the instant of its generation. The network performance $g(\cdot,\cdot)$ is expressed as a function of data latency:

$$g(\boldsymbol{x}[k],\omega[k]) = \frac{\rho}{m_a[k]} \cdot \sum_{i=0}^{3}\sum_{j=1}^{m_i[k]} F_i(\delta_j^i(\boldsymbol{x}[k],\omega[k])) + (1-\rho)\left(\frac{1}{m_a[k]}\sum_{i=0}^{3}\sum_{j=1}^{m_i[k]}\delta_j^i(\boldsymbol{x}[k],\omega[k])\right)^2$$

where $\rho \in [0,1]$ is a weighting factor, $m_a[k]$ is the number of messages observed during the kth iteration, i denotes priority level, $m_i[k]$ is the number of the priority level i messages during the kth iteration related to $m_a[k]$ by $m_a[k] = \Sigma_{i=0}^{3} m_i[k]$, and $\delta_j^i(\boldsymbol{x}[k],\omega[k])$ is the data latency of the jth priority level i message during the kth iteration. $F_i(\cdot)$ is a penalty function for priority level i messages and is defined as:

$$F_i(\delta_j^i(\boldsymbol{x}[k],\omega[k])) = \begin{cases} 0 & \text{if } \delta_j^i(\boldsymbol{x}[k],\omega[k])) \le \theta_i \\ (\delta_j^i(\boldsymbol{x}[k],\omega[k]) - \theta_i)^2 & \\ \quad \text{if } \theta_i < \delta_j^i(\boldsymbol{x}[k],\omega[k]) \le \theta_i + b_i & \\ b_i^2 & \text{if } \delta_j^i(\boldsymbol{x}[k],\omega[k]) > \theta_i + b_i, \end{cases}$$

with penalty threshold θ_i and penaly band b_i for priority level i messages.

The first term of the performance measure represents the average penalty over all messages such that a message of which data latency exceeds the corresponding threshold is penalized according to the penalty function $F_i(\cdot)$. This is analogous to variance of data latency. This form of penalty is especially useful for messages carrying time-critical information such as control signals and sensor data, interrupt signals, and video and voice data. The second part of the performance measure takes into account the square of average data latency over all messages. For network emulation on the testbed, ρ is set to 1 because the two terms of the performance measure were found to have a very close correlation with each other from simulation experiments.

The Performance Evaluator (PE) assesses the performance of an optimization algorithm by observing various items after timer settings have been changed. To this end, the performance of an optimization algorithm has been formulated focusing on its stepwise behavior rather than the asymptotic one. In fact, the asymptotic performance of an algorithm may not be measureable because the performance manager usually switches from one algorithm to another. The PE maintains history of the network performance and adjustment of the timer settings in order to provide criteria for evaluation of the recent action by the learning automaton. In this implementation, records for the past five iterations are maintained. The PE compares the current network performance with the average network performance of the past iterations. Also, the magnitude and signs of current timer changes are compared to those of the averaged timer changes for past iterations. Since perturbation analysis is executed all the time, the signs of the next changes are available for the PE to compare the signs of the current changes to those of the next changes. According to these comparisons, the PE selects one value for $\beta[k]$ out of five possible values.

V. Results and Discussion

The first part of this section presents the results from simulation experiments which were conducted to investigate the accuracy of the PA algorithm, which is an important ingredient of the performance management tool. The second part focuses on the results from emulation experiments the network testbed to demonstrate the efficacy of the proposed tool.

A. Results for Simulation Experiments

An LTPB network with 10 stations has been simulated with the transmission rate of 50 Mb/s and queue capacity of 10 messages for every queue. The network traffic is composed of messages at four priority levels. The total network takes 70% of the network capacity on the average: 10% for priority level 0 and 20% each for priority levels 1, 2, and 3. The PA algorithm formulated in Section III-A has been implemented in the simulation. This implementation maintains four separate perturbed paths and only one timer setting can be perturbed in each path. The simulation result under a typical scenario is discussed below.

Having set the timers THT, TRT1, TRT2, and TRT3 at the nominal values of 150, 1000, 800, and 600 μs, respectively, the PA algorithm was used to estimate the perturbed performance under four different perturbations on TRT3: $-100, -50, 50,$ and 100 μs. Four additional simulation experiments have also been performed to obtain network performance with four sets of perturbed timer settings without using the PA algorithm.

Fig. 4 shows the absolute values of percent error in estimating the perturbed network performance with TRT3 perturbations. In Fig. 4, S denotes the number of future message generations considered by the PA algorithm. For $S = 0$ where no future message is taken into consideration (shown by solid black bars), the error for positive perturbation is much larger than that for a negative one. The rationale is that a positive increment in a timer setting is likely to make the perturbation in token reception instant ΔC positive during a major part of the simulation experiment because an increased timer setting enables the corresponding queues to transmit more messages compared to the nominal path. If ΔC is positive, the PA algorithm has to construct the perturbed path with no knowledge on any potential change in the queue contents from the present time t. In order to reduce errors in estimating the

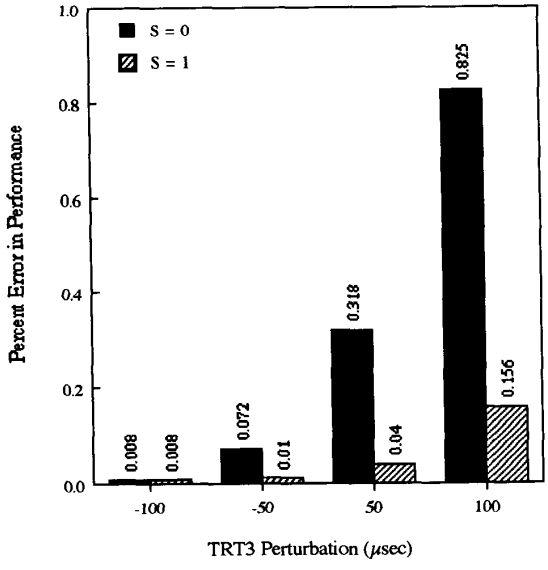

Fig. 4. Percent error of the PA algorithm with TRT3 perturbations.

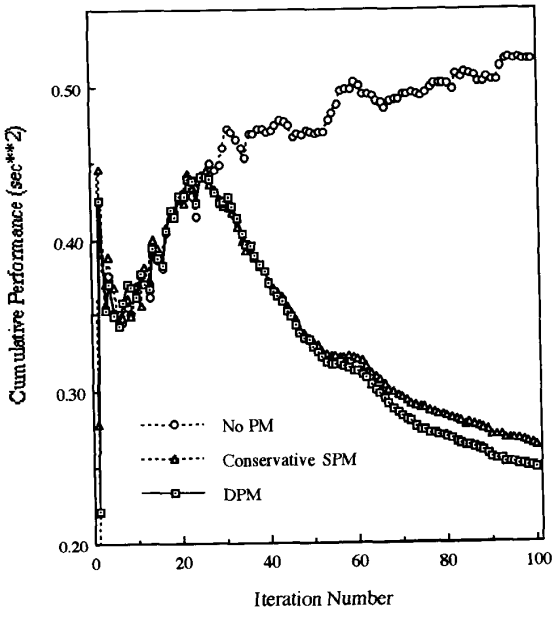

Fig. 5. Comparison of cumulative performance.

perturbed performance, the PA algorithm has been modified to look into the event calendar so that the next message generation at a given queue can be considered in constructing the perturbed path. This idea has been extended to consider message generations further in the future by scheduling more than one message generation for a given queue. The shaded bars for $S = 1$ in Fig. 4 show a significant reduction in the absolute values of the percent error when one future message generation at each queue is considered by the PA algorithm. However, this method for reducing errors is applicable only to simulation experiments. Simulation results show that the PA algorithm is capable of estimating the performance of the LTPB network under perturbed parameters by using the results of a single simulation experiment under the nominal condition. The estimation errors are found to be within 3% from all simulation experiments for perturbations in THT, TRT1, TRT2, and TRT3.

B. Results of Emulation Experiments on the Testbed

Three experiments, conducted on the network testbed, are reported here as typical results. For each of these experiments, the emulated network was run for 100 iterations which are equivalent to 300,000 message transmissions. During the first 25 iterations, the network was operated without any performance management action in order to achieve the steady-state operations. Four timers of the LTPB priority mechanism were adjusted on-line during the remaining 75 iterations. For these experiments, two different MSA algorithms (described in Section III-B) were used: one is referred to as *conservative* and the other as *liberal*. The conservative MSA algorithm has smaller values for initial step size c_i, reduction factor r, and weighting factor w_1 compared to the liberal algorithm. In other words, the conservative algorithm tends to adjust timers in smaller increments and to reduce its step size faster than the liberal algorithm. For the first two experiments, the two algorithms were used separately and one at a time for performance management without the learning scheme. This is referred to as the single-algorithm performance management (SPM). The conservative SPM was found to yield better cumulative performance than the liberal SPM. For the third experiment, both algorithms were employed togehter for performance management but one and only one of the two algorithms is selected by the learning automaton at any given iteration. This is referred to as the dual-algorithm performance management (DPM).

Fig. 5 shows a comparison between the cumulative network performance of the DPM and the conservative SPM. As a reference, the network without any performance management is also shown in Fig. 5. Up to 50 iterations, both SPM and DPM yield comparable network performance which is significantly superior to that without any performance management. Then onwards, the cumulative performance using DPM improves faster than that using conservative SPM. The rationale is that the DPM selects the liberal SPM (which has relatively larger stp sizes) more frequently to change the timer settings in larger increments, while the conservative SPM changes the settings in smaller increments which are reduced prematurely.

VI. Summary and Conclusions

A performance management algorithm for multiple access networks has been conceptualized, and formulated for a token bus protocol by using the principles of: i) perturbation analysis of discrete-event dynamic systems; ii) stochastic approximation; and iii) learning automata. The procedure is aimed to improve the network performance in handling various types of messages by on-line adjustment of protocol parameters, and has been emulated on a network testbed. The conceptual design presented in this paper offers a step forward to bridging

the gap between management standards and user's demands for efficient network operations since most standards such as ISO and IEEE address only the architecture, services, and interfaces for network management. The proposeed concept for performance management can also be used as a general framework to assist design, operation, and management of various DEDS such as computer integrated manufacturing and battlefield C^3. The following major conclusions are derived from the results of simulation and emulation of performance management of Linear Token Passing Bus (LTPB) protocols.

- The perturbation analysis algorithm can estimate the performance of an LTPB network by using the results of a single simulation experiment under the nominal condition. The estimation errors are found to be within 3% for all simulation experiments.
- The performance of an LTPB network can be maintained by on-line adjustment of its timers. Identification of the network traffic statistics is not required.
- The discrete reward/penalty (DRP) reinforcement scheme is well suited as an ingredient of on-line performance management under unknown and dynamically changing environment.

Some of the topics that are directly related to performance management are briefly discussed and recommended for future research.

- *Extension to Other Protocols:* The techniques used in this research can be applied to other medium access control (MAC) layer protocols. For example, the probability of transmission and backoff time of a p-persistent Carrier-Sense Multiple Access (CSMA) protocol can be adjusted depending on the network traffic. Upper-layer protocols can be managed within a framework similar to that developed in this research. Especially for network- and transport-layer protocols, routing and flow control algorithms can adapt themselves by incorporating a performance management procedure.
- *Application of Evolving Techniques for Decision Making:* The decision-making functions of a performance management procedure can be formulated by using the evolving techniques like fuzzy set theory, expert systems, and neural networks. Decisions for performance management may have to be made with insufficient *a priori* knowledge of the environment, and the problem could be highly nonlinear and time varying.

APPENDIX A
LINEAR TOKEN PASSING BUS PROTOCOL

A token bus protocol is a distributed controlled-access protocol for the Medium Access Control (MAC) layer. The right to use the medium is explicitly controlled by a special bit pattern called a token, and the responsibility of controlling the use of the medium lies with every station. This Appendix describes the priority mechanism of the Linear Token Passing Protocol (LTPB) [8], which is used in this paper to demonstrate efficacy of the proposed performance management tool.

A token bus network consists of a number of stations connected via a broadcast medium on which any transmission from a station can be heard by all stations. The right to transmit a message is given to a station when it receives a special bit pattern called a token. The token is passed from one station to another following a sequence of station addresses. The last station in the sequence sends the token back to the first station to form a logical ring. A station may transmit its messages before it passes the token to the next station in the logical ring sequence. A station with the token has complete control of the medium for a finite period of time. The length of this period depends on the number of waiting messages and the status of several timers. A station can transmit a number of messages or can pass the token to its successor (i.e., the next station in the logical ring sequence) without any transmission.

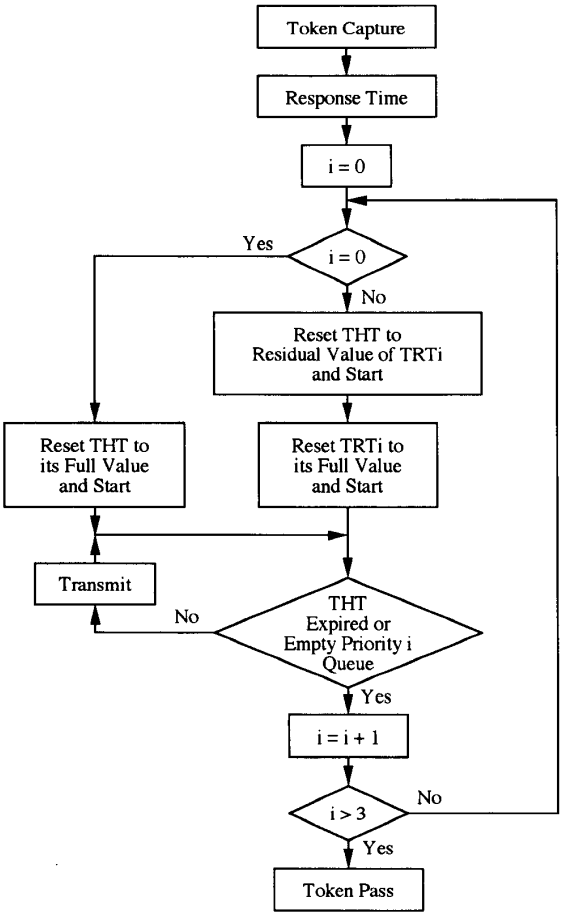

Fig. 6. Priority mechanism of LTPB protocol.

The LTPB protocol has a priority mechanism of four levels, namely 0, 1, 2, and 3, among which the priority level 0 has the highest priviledge of medium access. Each priority level has a queue to provide temporary waiting space for messages of the corresponding priority level. A Token Holding Timer (THT) and three Token Rotation Timers, i.e., TRT1, TRT2, and TRT3, regulate message transmissions for the priority level 0, 1, 2, and 3, respectively. The priority level 0 messages are allowed to start transmission within a period equal to the length of THT. For the lower priority level messages,

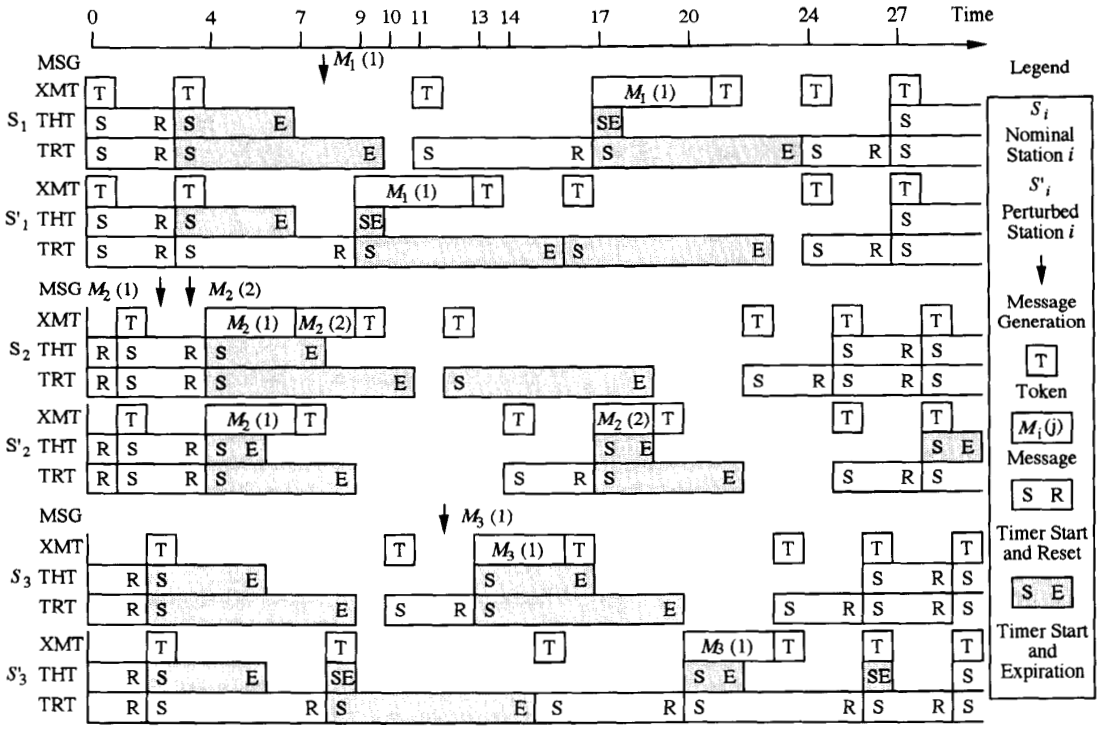

Fig. 7. Effects of TRT perturbation on network operations.

the initiation of a transmission must not occur beyond the residual period (i.e., the time left until its expiration) of the corresponding timer. If a timer expires while the corresponding priority message is being transmitted, the transmission will be continued to completely finish the current message and no further transmission is allowed until the instant of next token reception.

If a station receives the token, it performs self-diagnostics during the period of Response Time (RT) before any transmission. At the end of RT, the station resets THT to its full value and checks whether any message is waiting in the priority 0 queue. If the queue is empty, the chance of transmission is given to priority level 1; otherwise, the station starts its THT and begins to transmit the oldest message in the priority 0 queue. At the completion of a message transmission, the station checks whether THT has expired and whether there are more messages waiting in the priority 0 queue. If THT is not expired and the queue is not empty, the station starts another message transmission. This procedure continues either until the queue becomes empty or until THT expires.

After the station finishes this procedure for the priority level 0 messages, it checks if TRT1 has expired. If it is expired or the priority 1 queue is empty, then TRT1 is reset to its full value and restarted, and the chance of transmission is passed to the priority level 2 messages without any transmission of the priority 1 message. If TRT1 is not expired and the priority 1 queue is not empty, then THT is restarted after being reset to the remaining value of TRT1, and TRT1 is reset to its full value and restarted. The priority level 1 messages can be transmitted consecutively either until THT expires or until there is no message in the priority 1 queue.

When one of two conditions (namely THT expiration and empty priority 1 queue) is satisfied, the station begins the same procedure for TRT2 and the priority 2 queue, and continues for TRT3 and the priority 3 queue. After the station completes the procedure for priority level 3, the token is passed to the successor station. This priority mechanism is summarized by a flowchart in Fig. 6.

Appendix B
Perturbation Analysis via Timer Setting Changes of the LTPB Protocol

The network under consideration in this example consists of three stations which have only one Token Rotation Timer (TRT) and one Token Holding Timer (THT), which is used as a dummy timer to store the remaining time of TRT. For simplicity, it is assumed that message transmissions are solely controlled by the status of TRT. Therefore, each station is considered to have only one queue. Fig. 7 depicts the evolution of a network with nominal TRT setting (nominal path) and an evolution with perturbed TRT setting (perturbed path). The status changes of the station $i, i = 1, 2, 3$ are represented by S_i and S_i' for the nominal and perturbed paths, respectively. S_i consists of four elements, namely Message Generation Instant (MSG), Message Transmission Instant (XMT), Token Holding Timer status (THT), and Token Rotation Timer status (TRT). For the perturbed path, S_i' consists of three perturbed elements

where MSG is excluded because the instants of message generation are identical for both.

MSG is essentially the instant of message insertion into the queue. It is denoted by a down arrow with an appropriate identifier, $M_i(j)$, for the jth message at station i. XMT indicates the time interval of transmission and contains two kinds of transmissions. A block with a letter T denotes transmission of the token, and message transmission is depicted by a block with an appropriate message identifier. For THT and TRT, a block represents the time period while the corresponding timer counts down, which always starts with a letter S (start). The block ends with a letter R (reset) if the timer is reset before expiration. Otherwise, i.e., if the timer is expired, the block is shaded and ends with a letter E (expire).

Fig. 7 illustrates network operations from the initial time $t = 0$ when none of the three stations have any waiting message(s) and station 1 just received the token. Each token pass is assumed to take one unit of time, including the time required for the source station to transmit the token and that for the destination station to respond. The normal length of TRT is taken to be 7 units ot time.

On the nominal path (focusing on S_1, S_2, and S_3 in Fig. 7), since the queue of station 1 is empty, the token is passed to station 2 immediately after resetting and restarting its THT and TRT. At $t - 1$, station 2 receives the token. This process continues until station 2 receives the token again at $t = 4$. At this moment, its queue contains two messages and TRT is not yet expired. Therefore, station 2 resets and starts its THT having the remaining time of its TRT (4 units) and its disposal, and resets and starts its TRT. Almost simultaneously (it is assumed that timers are reset and started instantaneously), station 2 begins to transmit its first message $M_2(1)$ which takes 3 units of time to be transmitted. When the first message is finished, THT still has one unit of time left and the second message $M_2(2)$ is started. After finishing $M_2(2)$, the token is passed to station 3 which starts its TRT and passes the token to station 1. Upon token arrival at $t = 11$, station 1 has one waiting message. However, its transmission is not allowed because of TRT expiration at $t = 10$ before token reception. When station 3 receives the token $t = 13$, it can transmit $M_3(1)$ of two units since the message is already in the queue and TRT still has 4 units of time left to its expiration. When the token returns to station 1 at $t = 17$, station 1 finds one unit of time left on its TRT and starts transmitting $M_1(1)$ of four units. After this transmission, the network becomes empty, resulting in token circulations without any message transmission.

For the perturbed path (S_1', S_2', and S_3' in Fig. 7), the TRT in station 2 has been perturbed by -2 units of time while keeping TRT unchanged in stations 1 and 3. This change is solely for the purpose of illustration, since a timer is usually set identically for all stations in standard protocols. Effects of the perturbation do not appear on the perturbed path until $M_2(1)$ is finished at $t = 7$ (compare S_2 and S_2' at $t = 7$). Due to the reduced length of TRT, THT of station 2 has started with only 2 units when the token is received and expires while $M_2(1)$ is being transmitted. Therefore, no further transmission is allowed and $M_2(2)$ should wait for the next opportunity. Due to the deferred transmission, stations 3 and 1 receive the token 2 time units earlier [which is required for transmission of $M_2(2)$] on the perturbed path. At $t = 9$, station 1 has one unit of remaining TRT due to earlier token reception and can transmit $M_1(1)$. When the token returns to station 2 at $t = 14$, station 2 is again disallowed to transmit $M_2(2)$ because of TRT expiration. Station 3 also loses the opportunity to transmit $M_3(1)$ for the same reason. Eventually, $M_2(2)$ and $M_3(1)$ are transmitted at $t = 17$ and $t = 20$, respectively.

After transmission of $M_3(1)$ on the perturbed path, the perturbed instant of the token reception by station 1 at $t = 24$ conicides with that on the nominal path even though the statuses of its TRT's are different from each other. This implies that the instants of token reception by a given station on the nominal and perturbed paths become identical after transmission of all messages that are affected by timer perturbation. The timer status on the perturbed path also becomes identical to that on the nominal path after one more token circulation (from $t = 27$), except station 2 where the timer is perturbed.

It follows from Fig. 7 that it is impossible to predict the time of $M_2(2)$ transmision on the perturbed path without considering the queue contents and timer status, after detecting that $M_2(2)$ cannot be transmitted at $t = 7$ due to the perturbation in TRT. This is because the order of transmissions is dependent on the status of TRT, which is, in turn, dependent on the previous token circulation. Therefore, in order to compute the perturbed performance, the only choice is to construct the perturbed path, which is essentially an Extended Perturbation Analysis with Brute Force Algorithm (EPA/BFA) [13]. However, the brute force construction of the perturbed path is required only for the portion of the perturbed path which differs from the nominal one because the queue contents, timer status and event sequence of the perturbed path become identical to those on the nominal one as shown in Fig. 7. Therefore, the PA algorithm for this problem needs to check whether the perturbed path begins to differ from the nominal one while both paths are identical. After a difference between the two paths is detected, the algorithm constructs the perturbed path until it concides with the nominal one again.

REFERENCES

[1] A. Ray, "Networking for computer-integrated manufacturing," *IEEE Network*, vol. 2, no. 3, pp. 40–47, May 1988.

[2] *Manufacturing Automation Protocol (MAP) 3.0 Implementation Release*, Available through MAP/TOP Users Group, Dearborn, MI.

[3] *Technical and Office Protocols (TOP) 3.0 Implementation Release*, Available through MAP/TOP Users Group, Dearborn, MI.

[4] D. M. Thompson, "LAN management standards—Architecture and protocols," in *Proc. IEEE INFOCOM '86*, pp. 355–363.

[5] T. Saydam and A. S. Sethi, "Token bus/ring local area network management concepts and architecture," *IEEE INFOCOM '87*, pp. 988–993.

[6] S. M. Klerer, "The OSI management architecture: An overview," *IEEE Network*, vol. 2, no. 2, pp. 20–29, Mar. 1988.

[7] ANSI/IEEE Standard 802.1," Supplement B: Systems Management," Draft M, Jan. 1987.

[8] *Linear Token Passing Multiplexed Data Bus Standard*, Version 3.0, Soc. of Automobile Engin., May 1987.

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

[10] L. Kleinrock, *Queueing Systems, Vol. I: Theory*. New York: Wiley, 1975.

[11] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. New York: McGraw-Hill, 1991.

[12] Y-C. Ho and X-R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*. New York: Kluwer, 1991.

[13] Y-C. Ho and S. Li, "Extensions of infinitesimal perturbation analysis," *IEEE Trans. Automat. Contr.*, vol. AC-33, no. 5, pp. 427–438, May 1988.
[14] R. Y. Rubinstein, *Monte Carlo Optimization, Simulation and Sensitivity of Queueing Networks*. New York: Wiley, 1986.
[15] K. S. Narendra and M. A. S. Thatachar, *Learning Automata*. Englewood Cliffs, NJ: Prentice Hall, 1989.
[16] Y. C. Ho and X.-R. Cao, "Perturbation analysis and optimization of queueing networks," *J. Optim. Theory and Applic.*, vol. 40, no. 4, pp. 559–582, Aug. 1983.
[17] B. J. Oommen and J. P. Christensen, ε -optimal discretized linear reward-penalty learning automata," *IEEE Trans. Syst., Man and Cybernet.*, vol. SMC-18, no. 3, pp. 451–458, May/June 1988.
[18] *MP-400 Programming Reference Manual*. XXX; Industrial Networking Inc., July 1987.

Suk Lee (M'92) was born in Seoul, Korea, in 1961. He received the B.S. degree in mechanical engineering from Seoul National University in 1984, and the M.S. and Ph.D. degrees in mechanical engineering from the Pennsylvania State University in 1985 and 1990, respectively.

From 1985 to 1990, he was a Graduate Research Assistant working on network management and evaluation of fly-by-wire flight control systems. Upon completion of graduate study, he joined the Center for Advanced Manufacturing Systems at the University of Cincinnati as a Research Assistant Professor. His current research interests include network management and control, modeling and performance evaluation of communication networks, perturbation analysis, computer networking for manufacturing, design, and evaluation of flexible manufacturing systems, and control of machining processes for composite materials.

Dr. Lee is an Associate Member of ASME and a Senior Member of SME.

Asok Ray (SM'83) received the Ph.D. degree in mechanical engineering from Northeastern University, Boston, MA, in 1976, and has received degrees in electrical engineering, computer science, and mathematics. He joined the Pennsylvania State University in July 1985, and is currently a Professor of Mechanical Engineering. Prior to joining Penn State, he held research and academic positions at the Massachusetts Institute of Technology and Carnegie-Mellon University as well as research and management positions at GTE Strategic Systems Division, Charles Stark Draper Laboratory, and MITRE Corporation. His research experience and interests include control and optimization of continuously varying and discrete-event dynamic systems in both deterministic and stochastic settings, intelligent instrumentation for real-time distributed processes, and design of fault-accommodating and robust control systems as applied to aeronautics and astronautics, power and processing plants, and autonomous manufacturing. He has authored or co-authored over two hundred research publications, including eighty-five scholarly articles in refereed journals and a research monograph entitled *An Integrated System for Intelligent Seam Tracking in Robotic Welding* (Springer-Verlag).He is currently serving as an Associate Editor for two journals, namely, *Journal of Dynamic Systems, Measurement and Control, Transactions of the American Society of Mechanical Engineers*, and *International Journal of Flexible Manufacturing Systems*. He is registered as a Professional Electrical Engineer in the Commonwealth of Massachusetts.

Dr. Ray is an Associate Fellow of AIAA and a member of ASME.