Discrete Event Dynamic Systems: Theory and Applications, 13, 263–289, 2003

On the Effect of Communication Delays in Failure Diagnosis of Decentralized Discrete Event Systems

RAMI DEBOUK* rami.debouk@gm.com
Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122, USA

STÉPHANE LAFORTUNE stephane@eecs.umich.edu
Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122, USA

DEMOSTHENIS TENEKETZIS teneketzis@eecs.umich.edu
Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122, USA

Abstract. We study the effect of communication delays on the performance of a coordinated decentralized architecture for failure diagnosis of untimed discrete event systems. The architecture consists of local sites communicating with a coordinator that is responsible for diagnosing the failures occurring in the system. A protocol that realizes the architecture is defined by the diagnostic information generated at the local sites, the communication rules used by the local sites, and the decision rule used by the coordinator to infer the occurrence of failures. Our prior work (Debouk et al., 2000) has addressed the performance of a set of protocols under the assumption that messages are received by the coordinator in the order in which they are sent globally. In this work we relax the abovementioned assumption. We modify the coordinator's decision rule for two of the protocols analyzed in Debouk et al. (2000) to account for the reception of out of order messages. We discover conditions on the system structure under which the modified protocols perform as well as the centralized diagnostic scheme proposed in Sampath et al. (1995).

Keywords: failure diagnosis, discrete event systems, decentralized information, diagnostic protocols, communication delays

1. Introduction

The problem of failure diagnosis has received considerable attention in the literature since detecting and isolating failures plays an important role in the automatic control of large complex systems. Consequently, many approaches for diagnosis have been extensively studied (we refer the interested reader to Willsky, 1976; Pouliezos and Stavrakakis, 1994 and the introduction of Sampath et al., 1995 for a survey of failure diagnosis techniques). Almost all of the failure diagnosis approaches in the literature have been developed for systems where the information used for fault diagnosis is centralized. The majority of technologically complex systems (computer and communication networks, manufacturing

* Rami Debouk is currently with General Motors R&D and Planning, Mail Code 480-106-390, 30500 Mound Road, Warren, MI 48090-9055, USA.

systems, process control and power systems, etc.) are informationally decentralized. In decentralized information systems there are several work stations (decision makers, controllers, diagnosers) each having access to its own local information. The stations may communicate and exchange limited information with one another. If the local information available at a work station is used in conjunction with the complete system model in order to generate the local diagnostic information, then we are dealing with decentralized or distributed diagnosis. Otherwise, if the work station only monitors a local component (or a set of components) of the system and bases its diagnostic on the model of this component (set of components) only, then we are dealing with modular diagnosis. In either case, approaches to failure diagnosis suggested in the literature do not apply directly to these systems, hence the need to develop diagnostic methodologies for informationally decentralized systems. This fact is also recognized in Aghasaryan et al. (1998), Baroni et al. (1999), Deb et al. (1998), Fabre et al. (2000), Holloway and Chand (1994), Mohindra and Clark (1993), Pencolé (2000), Ricker and Fabre (2000), and Sengupta (1998). A discussion of some of these approaches is available in Debouk (2000).

Although similarities exist between our approach and that of Baroni et al. (1999), Deb (1998), Pencolé (2000), and Sengupta (1998), the modeling and/or informational assumptions of Baroni et al. (1999), Deb (1998), Pencolé (2000), and Sengupta (1998) and our work are clearly different: Sengupta (1998) proposes a discrete event systems approach for failure diagnosis and focuses on diagnosis solutions based on inter-diagnoser messaging schemes; the techniques of Deb (1998) rely on the fact that all information about a sub-system and its neighbors is available for processing, or in other words the local model is fully observable; Baroni et al. (1999) and Pencolé (2000) use modular diagnosis approaches that are based on communicating finite system machines with some implied timing information. Theses differences will become more evident in the sequel of the paper.

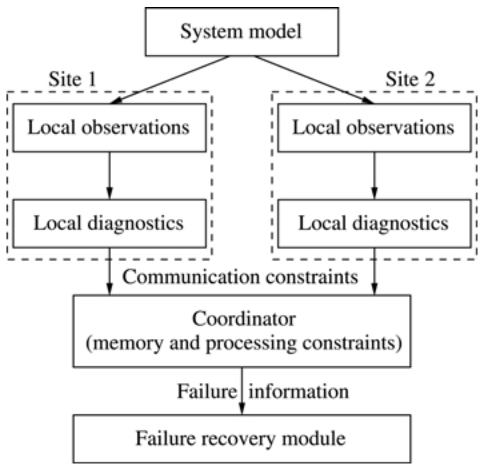

Figure 1. Coordinated decentralized architecture.

In this paper, we restrict attention to a coordinated decentralized architecture for failure diagnosis with two local sites communicating with a coordinator. This architecture is depicted in Figure 1. The top block in Figure 1 represents the complete system model. The system is modeled by a regular language which accounts for both normal and failed modes of operation of the system. The language is defined over an alphabet (event set) which is the union of the disjoint sets of observable events and unobservable events. The set of failures to be diagnosed is a subset of the set of unobservable events. Each site is composed of two modules: an observation module and a diagnostic module. Site i, $i \in \{1, 2\}$, locally observes the system, and based on its own observations generates its own diagnostic information. Both sites communicate some form of their diagnostic information to the coordinator. The task of the coordinator is to process, according to a prescribed decision rule, the messages received from both sites to infer occurrences of failures. If a failure is detected by the coordinator, it is broadcast to the failure recovery module.

In Debouk et al. (2000) we investigated the diagnosability properties of the above architecture under a set of assumptions. A major assumption is that the messages communicated to the coordinator are received in the order they are sent globally. This assumption may be too restrictive for decentralized systems. Local sites are using, in general, different links to route their messages to the coordinator, and consequently messages sent from different sites may be received out of order at the coordinator due to communication or propagation delays. The objective of this paper is to study the effect of these delays, and consequently the effect of out of order reception of messages, on the performance of coordinated decentralized diagnostic protocols. For that matter, we concentrate on two diagnostic protocols presented in Debouk et al. (2000), namely Protocols 1 and 2, and relax the assumption that global ordering of the messages at the coordinator's site is maintained. We call the resulting diagnostic schemes Protocol 1D and Protocol 2D, respectively. We discover conditions sufficient to guarantee that Protocols 1D and 2D perform as well as the centralized diagnostic scheme proposed in Sampath et al. (1995). The diagnostic performance of Protocol 1D is inferior to that of Protocol 1 in the sense that there are conditions on the system structure to guarantee that Protocol 1D performs as well as the diagnostic scheme of Sampath et al. (1995) whereas Protocol 1 performs as well as the diagnostic scheme of Sampath et al. (1995) irrespective of the system structure. Protocols 2 and 2D perform as well as the centralized diagnostic scheme of Sampath et al. (1995) under the same conditions. Consequently, relaxing the assumption on global ordering of messages at the coordinator's site does not result in a performance degradation in the case of Protocol 2D. However, Protocol 2D requires more memory storage at the coordinator's site than Protocol 2. Similarly, in the instances where protocol 1D performs as well as the diagnostic scheme of Sampath et al. (1995), the protocol requires more memory than Protocol 1.

This paper is organized as follows. Section 2 briefly reviews the coordinated decentralized architecture for failure diagnosis of Debouk et al. (2000). Section 3 discusses our approach to accounting for communication delays in the framework of the coordinated decentralized architecture of Figure 1. Sections 4 and 5 define two protocols that realize the coordinated decentralized architecture and analyze their diagnostic properties. Section 6 presents a brief critique of the diagnostic performance achieved by the presented protocols.

2. Preliminaries

We first note that this paper is presented as a complement of Debouk et al. (2000). The definitions, assumptions, notations, and problem setting of Debouk et al. (2000) needed for the development of the technical results of the paper are assumed. The following is a high level description of the diagnostic approach and the coordinated decentralized architecture.

2.1. The Diagnostic Approach

We begin by reviewing the discrete event systems approach to failure diagnosis with centralized information upon which the decentralized diagnostic protocols we present build on. The system to be diagnosed is represented by a deterministic automaton $G = (X, \Sigma, \delta, x_0)$ where X is the state space, Σ is the set of events, δ is the partial transition function, and x_0 is the initial state of the system. G accounts for the normal and failed behavior of the system. The event set Σ over which the automaton is defined is partitioned into two disjoint sets: a set of observable events Σ_o and a set of unobservable events Σ_{uo}. The projection from Σ to Σ_o is denoted by P. Failures of interest are a subset Σ_f of the set of unobservable events. These failures are partitioned into disjoint failure types. The system is said to be diagnosable with respect to a set of observable events if the occurrence of any failure can be detected with a finite delay using the history of observable events. Diagnosis of failures is achieved through the use of diagnosers. The diagnoser G_d (or extended diagnoser G_d^e) of G is a deterministic automaton that estimates the state of the system automaton following the observations and appends failure labels to the state estimates. The diagnoser is at the core of the diagnostic methodology: it is used to analyze the diagnosability properties off-line by performing a test on a specific type of its cycles, and to perform diagnosis when it observes on-line the behavior of the system. Definitions regarding diagnosers and extended diagnosers can be found in Sections 2.4 and 4.1.1 of Debouk et al. (2000).

2.2. Diagnosis in the Coordinated Decentralized Architecture

In decentralized systems, the global system information is distributed at several sites. The ''agents'' (decision makers, diagnosers, etc.) at different sites may communicate and exchange information in real time, or just report a processed version of their information to a center that, in general, possesses limited knowledge of the system. In this paper we consider the coordinated decentralized architecture depicted in Figure 1. The top block represents the system model, or G in the notation of Section 2.1. G models the synchronization of the interaction of all the components that constitute the system (see Sampath et al., 1996). Each site is composed of two modules: an observation module and a diagnostic module. Site i, $i \in \{1, 2\}$, locally observes the system based on its available sensing capabilities. Therefore, a projection P_i is associated with site i, where P_i is defined on the set of observable events Σ_{oi}; Σ_{o1} and Σ_{o2} need not be disjoint although sites 1 and 2 may be physically apart. The union of Σ_{o1} and Σ_{o2} is the set of all observable events Σ_o. Site *i* locally processes its own observations and generates its own diagnostic information. Both sites communicate a processed version of their observations to the coordinator. The nature of information communicated is determined by the communication rules used by the sites. The task of the coordinator is to process, according to a prescribed decision rule, the messages received from both sites to infer occurrences of failures. If a failure is detected by the coordinator, it is broadcast to the failure recovery module.

The main set of assumptions under which we intend to investigate the diagnosability properties of the above architecture are as follows.

- The system is diagnosable with respect to the union of observable events at both sites, however it is not diagnosable with respect to either set of locally observable events.
- Messages communicated by a given local site are received at the coordinator in the order they are sent.
- The two sites are allowed to report to the coordinator only some processed version of their raw data.
- The coordinator does not have a model of the system and it has limited memory and limited processing capabilities.

We define a protocol to be a realization of the architecture, that is, a protocol is defined by the local diagnostic rules, the communication rules between the local sites and the coordinator, and the decision rule used by the coordinator to infer the occurrence of failures. A protocol is said to diagnose the system if all failures are detected and isolated by the coordinator within a finite number of steps after the failure event has occurred. Thus, diagnosability requires that the detection of any failure should be achieved by the coordinator within a finite delay of the occurrence of that failure.

We note that in this work, in contrast to Debouk et al. (2000) (cf. Section 3.1), there is no assumption that global order of messages is maintained. That is, the messages transmitted by different (distinct) local sites are not necessarily received at the coordinator's site in the order in which they are sent. In fact, and as mentioned earlier, this paper studies the effect of out of order message reception at the coordinator's site on the performance of the decentralized diagnostic protocols presented in Debouk et al. (2000).

3. Addressing Communication Delay Issues in the Coordinated Decentralized Architecture

In coordinated decentralized architectures, like the one in Figure 1, the coordinator may receive messages out of order. Ordering of messages from one site to the coordinator may be easily achieved with a transport layer protocol, for instance TCP, or a data link control layer protocol such as Go Back *n* or Selective Repeat (Bertsekas and Gallager, 1992). However, global order, i.e., ordering of messages sent by different local sites to the coordinator, is not necessarily maintained. To achieve global ordering of messages we may use one of the following mechanisms:

1. We can introduce clocks at the local sites and time-stamp the messages sent by the local sites to the coordinator. In this situation we need to make sure that clocks are synchronized; we can achieve clock synchronization by the use of global positioning systems (GPS) (see Schmid, 1997). Such a mechanism is not always practical or feasible. For example, synchronizing clocks may be too constraining in low-energy mobile communication networks (Stark et al., 2002) and telecommunication networks (Fabre et al., 2000; Pencolé, 2000). The amount of information exchanged among the nodes of a communication network to achieve the synchronization of the local clocks is considerable; moreover additional processing and memory storage is required at the local sites.

2. We can use untimed discrete event models and design algorithms that order the messages arriving at the coordinator's site. Such an approach is enforced, for instance, in telecommunication networks where timing information, be it global time or local time, is not available at any node (site) of the telecommunication network (Fabre et al., 2000; Pencolé, 2000).

In this paper we will use the second approach. Under this approach, to generate a diagnostic decision we store all incoming messages up until the instance where all possible orders in which these messages are sent by various local sites can be sorted out. Once these orders are sorted out, the coordinator's decision rule is applied to all possible orders. The same procedure is repeated every time incoming messages arrive at the coordinator's site. To highlight our approach to storing out message orderings at the coordinator's site, we assume that messages sent by local sites are received at the coordinator at most ''one-step out of order''. The same approach can be used when messages are received at most ''n-step out of order'' where n is finite, $n > 1$; the memory requirements at the coordinator's site increase as n increases. To illustrate the ''one-step out of order'' assumption consider the following scenario. Assume the system executes the sequence of events ab. Suppose that event a (resp. b) is observed by local site 1 (resp. local site 2). Denote by x (resp. y) the message generated by the occurrence of event a (resp. b). If the order of reception of messages at the coordinator's site is yx, then x is said to be received ''one-step out of order''. Under the assumption of at most ''one-step out of order'' arrival of messages at the coordinator's site, we analyze the performance of two of the protocols presented in Debouk et al. (2000), namely Protocols 1 and 2.

4. Protocol 1D: A Coordinated Decentralized Protocol

In this section, we modify Protocol 1 considered in Debouk et al. (2000) to account for communication delays. We refer to the modified protocol as Protocol 1D (where 1D stands for the fact that it is a modification of Protocol 1 of Debouk et al. (2000) that allows for communication Delays). A key feature of Protocol 1 studied in Debouk et al. (2000) is the following: under the assumption that all communicated messages are received in the correct order by the coordinator, the coordinator is capable of ''tracking'' the state of the system as well as a centralized diagnoser, even though it does not have any knowledge of the dynamics of the system. This feature is the key to understanding and analyzing the performance of Protocol 1D. When the messages are received out of order by the coordinator, the coordinator should consider all possible orders according to which the messages may have been sent and for each possible order it should use the information update rule specified by Protocol 1 (briefly reviewed later). By doing so the coordinator achieves the following (see Section 4.4): (1) if a specific order of messages corresponds to a legal behavior of the system the coordinator's update is non-empty, and for that order it reconstructs the state of the centralized diagnoser associated with that legal behavior; (2) otherwise the coordinator rejects that order as impossible because it gives rise to an empty update. This implies that: (i) the memory requirements at the coordinator's site may increase considerably, because the coordinator has to store all the abovementioned diagnoser states; (ii) the coordinator can identify a failure type F_i only when F_i appears in the components of all the centralized diagnoser states reconstructed as described above. Therefore, we expect that, in general, the performance of Protocol 1D will not be the same as that of Protocol 1 (and consequently that of the centralized diagnostic scheme of Sampath et al. (1995)). To achieve the same performance as a centralized diagnoser we will need to impose further structure on the system. We determine conditions under which the protocol is capable of diagnosing the same types of failures as the ones diagnosed using the centralized diagnostic scheme of Sampath et al. (1995).

As is the case for any protocol that realizes the coordinated decentralized architecture, we need to define the diagnostic rules at the local sites, the communication rules employed by the local sites to communicate their diagnostic information to the coordinator, and the decision rule used by the coordinator to infer the occurrence of failures. This is done in the following three subsections.

4.1. Diagnostic Information at Local Sites

As is the case with Protocol 1, extended diagnosers are implemented at local sites. Consequently, the diagnostic information available at each site is provided by the state of the extended diagnoser. The state information is refined by the unobservable reach (cf. Definition 9 in Debouk et al., 2000).

4.2. Communication Rules

The communication rules as defined for Protocol 1 are used for Protocol 1D. Communication rule [CRi], $i = 1, 2$, says that after the agent at site i observes an event $\sigma \in \Sigma_{oi}$, it communicates to the coordinator the corresponding state q_i of its diagnoser G^e_{di}, its unobservable reach $UR_i(q_i)$ with respect to $\Sigma \backslash \Sigma_{oi}$, and a status bit, SB_i, that takes the values $SB_i = 1$ when $\sigma \in \Sigma_{oj}, j \in \{1, 2\}, j \neq i$, or $SB_i = 0$ when $\sigma \notin \Sigma_{oj}$.

4.3. Decision Rule

The coordinator's decision rule is composed of the following three steps:

1. Store all incoming messages at the coordinator site, and sort out all possible orders in which these messages may be generated by the local sites.

2. Apply the information update rule, as defined in Section 4.1.3 in Debouk et al. (2000), to each and every possible sorted out order, and retain all orders that result in a non-empty update (intersection).

3. Compare the failure properties of all surviving updates from step (2), and declare the occurrence of a failure when all these updates are certain of the occurrence of the failure.

These steps are discussed in detail in the following three sub-sections.

4.3.1. Sorting Out Possible Orders

All communication messages received at the coordinator's site are stored until the instance where all possible orders in which these messages are generated by the local sites can be figured out. Determining the instance where all possible orders can be sorted out depends on the messages received, namely which site observed an event and subsequently sent the message. In general, one can continue sorting out (uncovering) all possible orders after waiting for the arrival of at most three new messages at the coordinator's site. This is a direct consequence of the ''one-step out of order'' assumption and it is explained below.

Table 1 depicts the arrival of three new messages at the coordinator's site. A message with subscript i, $i \in \{1, 2\}$, indicates it has been sent by site i. The left column in Table 1 describes the order in which messages are received at the coordinator's site (left is earliest), the middle column describes all possible orders that may have resulted in the reception of messages by the coordinator, and the right column describes the orders sorted

Table 1. Sorting out possible orders at the coordinator site.

Messages Received	Possible Orders	Sorted Out Orders
$x_1y_2z_2$	$x_1y_2z_2$ or $y_2x_1z_2$	x_1y_2 or y_2x_1
$x_2y_1z_1$	$x_2y_1z_1$ or $y_1x_2z_1$	x_2y_1 or y_1x_2
$x_1y_2z_1$	$x_1y_2z_1$ or $y_2x_1z_1$ or $x_1z_1y_2$	x_1y_2 or y_2x_1 or $_1z_1y_2$
$x_2y_1z_2$	$x_2y_1z_2$ or $y_1x_2z_2$ or $x_2z_2y_1$	x_2y_1 or y_1x_2 or $_2z_2y_1$

out. The second column of the first row in Table 1 means the following: either the reception order is the same as the one in which the messages are sent, or y_2 could have been sent prior to x_1 (due to the ''one-step out of order'' communication delay). In both cases z_2 is sent after y_2 due to preservation of local order. The third column of the first row in Table 1 means the following: the sorted out (uncovered) orders are x_1y_2 or y_2x_1 and z_2 needs to be stored until new messages are received to figure out the order in which it was generated. The second row is the symmetric to the first and the last two rows are interpreted in a similar way. Initially, the ''order sorting'' procedure is to wait for three messages and afterwards uncover all possible orders of the first two messages while keeping the third message for later consideration after new messages arrive. Later, in Section 4.3.2, we will provide a more detailed description of how the sorting procedure is implemented. Also, a brief discussion of the procedure under the assumption of at most ''n-step out of order'' reception of messages appears in Section 6.

We conclude this sub-section with the following remarks.

Remark 1: Under the ''one-step out of order'' assumption, some cases require to wait for the arrival of only two messages to continue the sorting out procedure. For example, if two consecutive messages x and y are received by the coordinator from the same site, say site 1, then the coordinator is sure that message x was sent first, by the local order assumption.

Remark 2: In the case of possible orders $x_1z_1y_2$ and $x_2z_2y_1$ in rows three and four, respectively, of Table 1, the order in which the three messages are sent is uncovered since by the ''one-step out of order'' assumption messages y_2 and y_1, received at the coordinator's site prior to messages z_1 and z_2, should have been sent directly after these messages.

Remark 3: The above sorting procedure assumes that there are no events commonly observed by sites 1 and 2. If there are commonly observed events by both sites these events can be used as a synchronization mechanism as follows: once a message regarding a commonly observed event is received at the coordinator, say from site 1, then under the ''one-step out of order'' assumption the message regarding the same event from site 2 should either follow or could be delayed at most by one step; hence, the correct order of the message regarding the commonly observed event is recovered (since local order is preserved) and the same sorting algorithm can be applied to the remaining messages that have been received but not yet sorted.

4.3.2. Application of the Update Rule

We briefly describe the update rule of Protocol 1 as defined in Table 1 of Debouk et al. (2000). In order to do so we first review from Section 4.1.3 of Debouk et al. (2000) the structure of the coordinator. In addition to the register C where the coordinator stores its current diagnostic information, eight supplementary registers are used for storing messages and previous relevant values necessary for the update of its information. These registers store the latest states and unobservable reaches of G^e_{d1} and G^e_{d2}, the previous coordinator diagnostic information, and necessary information (some status bits) to compute the new coordinator diagnostic information. The update rule follows the following logic: first, intersect the state of the diagnoser that observed the last event with the unobservable reach of the other diagnoser, and then intersect the result with the previous coordinator state. If the event is commonly observed by both sites then the first intersection involves the states of both diagnosers. The reader is referred to Section 4.1.3 of Debouk et al. (2000) for an explanation of the rationale behind the information update rule. At reset, registers are initialized with the initial states and unobservable reaches of G^e_{d1} and G^e_{d2}. Before performing any update of the coordinator diagnostic information, the current coordinator diagnostic information is saved into the register holding the previous coordinator state for later use.

At any instant when all possible orders are sorted out the coordinator applies to each order of them the information update rule of Protocol 1 (cf. Table 1 in Debouk et al., 2000). For every possible order that survives the update rule (that is, it results in a non-empty intersection), the coordinator uses the diagnostic register C to hold the new diagnostic update, along with the other needed registers to hold corresponding states and unobservable reaches and status bits. If a specific sorted out order results in an empty intersection following the application of the update rule, it is rejected by the coordinator. The same procedure as above is applied when new orders are sorted out as a continuation of the already existing ones, and the new updates replace the old ones that are discarded.

Having defined the information update rule, we now provide more details on how the ‘‘order sorting’’ procedure introduced in Section 4.3.1 and the information update rule are implemented. Initially, the coordinator waits until it receives three messages (as depicted in Table 1) before sorting out the possible orders. At that instant orders including only two messages are sorted out, and the information update rule is applied to these orders. The registers for each sorted out order are updated to hold the two newest diagnostic updates along with the corresponding states and unobservable reaches and status bits. Also stored for each order is the third message (cf. Table 1). When two new messages are received at the coordinator's site, the coordinator sorts out the new possible orders (out of the third message that was previously stored for each existing sorted out order and the two new messages that have been received) as a continuation of a previously uncovered order. To every new sorted out order, the coordinator applies the information update rule. By successively applying the procedure just described, the coordinator sorts out the possible orders after receiving two new messages, except for ‘‘reset’’ steps where three new messages are needed. A ‘‘reset’’ step is either the first time the coordinator begins sorting out messages (discussed above) or when a previous sorting attempt results in an uncovered order that contains all the received messages (cf. the last uncovered orders in rows three and four in Table 1 and the explanation in Remarks 2 and 3). In general, at any instant of time when the coordinator has received $2m+1$ messages, the orders of at least $2m$ messages are uncovered by the arguments presented above.

4.3.3. Diagnosing Failures

If all the information updates that result from applying the procedure described in Sections 4.3.1 and 4.3.2 are certain that a failure F has occurred, then the coordinator declares that F has occurred and broadcasts this information to the failure recovery module. Otherwise, a diagnostic decision is postponed.

4.4. Diagnostic Properties of Protocol 1D

We prove that Protocol 1D performs as well as Protocol 1 under certain conditions on the system structure. Thus, when the coordinator receives messages out of (global) order, we have a degradation in the performance of Protocol 1.

To proceed with the analysis of Protocol 1D, we introduce the following concept.

DEFINITION 4.1 *A trace $s \in L(G)$ is said to be ambiguous with respect to the projections P_1 and P_2 and the failure type F_i if there exist a set of traces $S = \{s_1, s_2, \ldots\}$ in $L(G)$ such that $s_1, s_2, \ldots$ are arbitrarily long (Note 1), and the following is true:*

1. $P_1(s) = P_1(s'),\ s' \in S,$
2. $P_2(s) = P_2(s'),\ s' \in S,$
3. $P(s) \neq P(s'),\ s' \in S,$
4. $\| P(s) \| = \| P(s') \|,\ s' \in S,$
5. $F_i \in s,$
6. and there exists $s' \in S$ such that $F_i \notin s'$.

This definition says that the traces s and s', $s' \in S$, are distinguishable and have the same number of observable events under the projection P; however, these traces are not distinguishable under the projections P_1 and P_2. Furthermore, a failure type F_i is in s while there exists an $s' \in S$ that does not have the failure type F_i. The following example illustrates the concept of ambiguous traces.

Example 4.1: Consider the system shown in Figure 2. The set of events is $\Sigma = \{a, b, c, d, e, \sigma\}$, and σ is the only unobservable and failure event. Let F_1 denote the failure type of the event σ. Define $\Sigma_{o1} = \{a, c, d, e\}$ and $\Sigma_{o2} = \{b, d, e\}$. The trace $s = bac\sigma(de)^n$, for a given integer n, is ambiguous because there exists a trace $s' = abc(de)^n$ such that: (1) $P_1(s) = P_1(s') = ac(de)^n$ (2) $P_2(s) = P_2(s') = b(de)^n$, (3) $P(s) = bac(de)^n \neq abc(de)^n = P(s')$, (4) $\| P(s) \| = \| bac(de)^n \| = \| abc(de)^n \| = \| P(s') \|$, (5) $\sigma \in F_1$ belongs to s, and (6) $F_1 \notin s'$.

The main result concerning the performance of Protocol 1D is summarized by the following theorem.

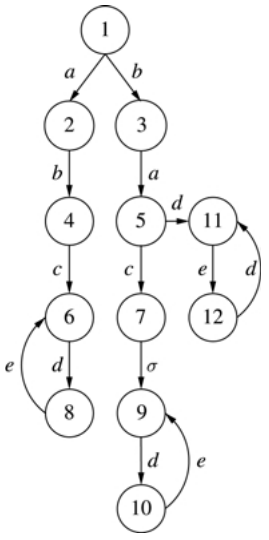

Figure 2. System model for Example 4.1.

THEOREM 4.1 *If L(G) is diagnosable with respect to Protocol 1, then Protocol 1D eventually performs as well as Protocol 1 if and only if there are no ambiguous traces in* $L(G)$ *with respect to all failure types.*

The proof of Theorem 4.1 is based on an important property of Protocol 1D that is summarized by the following proposition.

PROPOSITION 4.1 *Consider any possible uncovered ordering of messages, say w, produced by the rule of Section 4.3.1 when* $s \in L(G)$ *is executed. If w is not the correct order in which messages are sent to the coordinator, then the update rule of Section 4.3.2 applied to w results in a non-empty intersection if and only if w belongs to* $P(L)$. *The non-empty intersection resulting from the application of the update rule of Section 4.3.2 on w gives the state of the centralized extended diagnoser corresponding to w.*

Proof of Proposition 4.1: Consider that the system is executing the trace $s_0u_1au_2bu_3c$ where $a \in \Sigma_{o1}$, $b \in \Sigma_{o2}$, $c \in \Sigma_{o1}$ (Note 2), and $u_1, u_2, u_3 \in \Sigma_{uo}^*$. Denote by x, y, and z the messages generated by the occurrence of events a, b, and c, respectively. Without loss of generality assume that the messages are received in the following order: x, y, then z. This covers the case presented in the third row of Table 1 (the fourth row is the symmetric case of the third row while by inspection one realizes that the cases presented in rows 1 and 2 are sub-cases of those presented in rows 3 and 4, respectively). Let q_1 and q_3 be the states of the diagnoser G_{d1}^e after the execution of the event a and c, respectively, and q_2 be the state of the diagnoser G_{d2}^e after the execution of the event b. Denote by q_{old}, q_{1old}, and q_{2old} the states of the diagnosers G_d^e, G_{d1}^e, and G_{d2}^e, respectively, following the execution of s_0.

We have, from Table 1 and the sorting procedure presented in Section 4.3.1, at most three possible continuations of the orders in which messages were sent. These orders are *xy*, *yx*, and *xzy*. One of them, *xy* (corresponding to the observation of *ab*), is the true one by assumption. From Theorem 3 of Debouk et al. (2000) we know that the application of the update rule of Section 4.3.2 to *xy* results in the state of the centralized extended diagnoser following the observation of $P(s_0)ab$. Hence, we need to check the result of the proposition for the orders *yx* (corresponding to the sequence of observable events *ba*) and *xzy* (corresponding to the sequence of observable events *acb*). In the remaining we do the proof for the order *yx*, and by the same argument we can prove the result for the order *xzy*. Denote by C_a the coordinator state resulting from applying the update rule to the order *yx*. We have the following (from Table 1 in Debout et al., 2000):

$$C_a = (q_1 \cap_e^R UR_2(q_2)) \cap_c C_b =: C_2 \cap_c C_b \tag{1}$$

where

$$C_b = (UR_1(q_{old1}) \cap_e^i q_2) \cap_c C_{old} =: C_1 \cap_c C_{old} \tag{2}$$

and C_{old} is the state of the coordinator following the order $P(s_0)$. Equation (1) is derived using the row DR1 in Table 1 of Debouk et al. (2000) since *b* the last observable event of the order *yx* belongs to $\Sigma_{o2} \backslash \Sigma_{o1}$; the index R is used in the intersection operator $\cap_e^R$ since site 2 observed the event previous to the last (event *a* in the order *yx*). Equation (2) is derived using the row DR4 of Table 1 of Debouk et al. (2000) since the last observable event considered (event *a*) belongs to $\Sigma_{o1} \backslash \Sigma_{o2}$; the index *i* is used in the intersection operator $\cap_e^i$ to denote either L or R: if site 1 observed the last event in s_0, $i = L$, otherwise $i = R$.

We prove the proposition by induction on the number of observable events (in Σ_o) in s_0. We first present the proof of the induction step, and the proof of the basis of induction follows from it by minor modifications.

- *Induction step*. We assume that: (i) for any trace $t_1 \in L(G)$ with $\| P(t_1) \| \leq n$ all possible orderings (different from the correct one) produced by the rule of Section 4.3.1 when t_1 is executed result in a sequence of non-empty intersections if and only if the orderings belong to $P(L)$; and (ii), the coordinator state C following any of the possible orderings belonging to $P(L)$ is equal to the corresponding state of the extended diagnoser. We want to prove that the same result is true for any trace $t_2 \in L(G)$ with $\| P(t_2) \| \geq n+2$ (Note 3).

  Let $t_2 = s_0 u_1 a u_2 b u_3 c$, where $\| P(s_0) \| = n$. We have from the induction hypothesis that C_{old} is equal to the state of the extended diagnoser following the observation $P(s_0)$, that is,

$$C_{old} = \delta_d^e(q_0, P(s_0)) = q_{old} \tag{3}$$

  Sufficiency ($\Leftarrow$). Assume $P(s_0)ba \in P(L)$ (we remind the reader that we prove the proposition for the order *yx* corresponding to the sequence of observable events *ba*).

Since $P(s_0)ba \in P(L)$, we know from the results of Proposition 2 (Case 3-ii) in Debouk et al. (2000) that applying the update rule for the order $P(s_0)ba$ results in the corresponding state of the centralized extended diagnoser.

Necessity ($\Rightarrow$). To prove necessity we exploit (1), (2), and the fact that the intersections C_a and C_b are non-empty. Assume that the update rule of Section 4.3.2 applied to the order $P(s_0)ba$ results in a set of non-empty intersections. We want to prove that $P(s_0)ba \in P(L)$. The update rule applied to $P(s_0)ba$ gives (1). By assumption C_a is non-empty; consequently, C_b and C_1 are non-empty. Hence, by the definition of $\cap_e^i$ (cf. Definition 15 in Debouk et al., 2000), C_1 is an extended diagnoser state, and it is of the form

$$C_1 = \{((x_{old1}, l_{old1}), (x_{new1}, l_{new1})), \ldots, ((x_{oldr}, l_{oldr}), (x_{newr}, l_{newr}))\} \tag{4}$$

Since C_1 is non-empty there exist traces in $L(G)$ whose projection with respect to P_2 equals $P_2(s_0)b$. Then by (2) we obtain

$$C_b = \{((x_{oldn}, l_{oldn}), (x_{newn}, l_{newn})), \ldots, ((x_{oldr'}, l_{oldr'}), (x_{newr'}, l_{newr'}))\} \tag{5}$$

where $\{n, \ldots, r'\} \subseteq \{1, \ldots, r\}$, and

$$x_{oldi} \in SP(q_{old}), \quad i = n, \ldots, r' \tag{6}$$

where $SP(\cdot)$ is defined in Section 4.1.1 of Debouk et al. (2000). We note here that $C_b \subseteq C_1$ and refer the reader to Example 6 in Debouk et al. (2000) for an example. From (2), (5), (6), and the definition of the state of the extended diagnoser, we conclude that there exist traces in $L(G)$ whose projection with respect to P equals $P(s_0)b$. Hence $P(s_0)b \in P(L)$, and

$$C_b = \delta_d^e(q_{old}, b) = \delta_d^e(q_0, P(s_0)b) \tag{7}$$

by the results of Proposition 2 in Debouk et al. (2000).

Since by assumption C_a in (1) is non-empty, C_2 in (1) is non-empty. But

$$C_2 = \{((y_{old1}, k_{old1}), (y_{new1}, k_{new1})), \ldots, ((y_{olds}, k_{olds}), (y_{news}, k_{news}))\} \tag{8}$$

Therefore, there exist traces in $L(G)$ whose projection with respect to P_1 equals to $P_1(s_0)a$. By (1), (7), and (8)

$$C_a = \{((y_{oldn}, k_{oldn}), (y_{newn}, k_{newn})), \ldots, ((y_{olds'}, k_{olds'}), (y_{news'}, k_{news'}))\} \subseteq C_2 \tag{9}$$

where $\{n, \ldots, s'\} \subseteq \{1, \ldots, s\}$, and

$$y_{oldi} \in SP(C_b), \quad i = n, \ldots, s' \tag{10}$$

From (5), (7), (9), and (10) we conclude that there exist traces in $L(G)$ whose projection with respect to P equals $P(s_o)ba$. Hence, $P(s_o)ba \in P(L)$, and consequently

$$C_a = \delta_d^e(C_b, a) = \delta_d^e(q_{old}, ba) = \delta_d^e(q_0, P(s_0)ba) \tag{11}$$

by the results of Proposition 2 in Debouk et al. (2000). This completes the proof of the necessity of the induction step.

- *Basis of induction.* The proof of the basis of induction is similar to that of the induction step with the minor change of letting $s_0 = \varepsilon$. Consequently, $C_{old} = q_{old} = \{(x_o, N), (x_0, N)\}$ (x_0 is the initial state of G) in (3) by definition of the protocol, and the rest of the proof remains as is. ■

Remark 4: We note that by the sorting procedure described in Section 4.3.1, the sorted out orders $P(s_0ba)$ and $P(s_0acb)$ in the proof of Proposition 4.1 result from the same set of messages received by the coordinator when $P(s_0abc)$ occurs. Hence the sorted out order $P(s_0ba)$ (resp. $P(s_0acb)$) leads to the diagnosers state q_1 and q_2 (resp. q_3 and q_2), and the following is true

$$\begin{aligned} &P_1(s_0ab) = P_1(s_0ba)\\ &P_2(s_0ab) = P_2(s_0ba)\\ &\| P(s_0ab) \| = \| P(s_oba) \| \end{aligned} \tag{12}$$

for the sorted out order $P(s_0ba)$, and

$$\begin{aligned} &P_1(s_0abc) = P_1(s_0acb)\\ &P_2(s_0abc) = P_2(s_0acb)\\ &\| P(s_0abc) \| = \| P(s_oacb) \| \end{aligned} \tag{13}$$

for the sorted out order $P(s_0acb)$. This fact is used in the proof of Theorem 4.1 which follows.

Proof of Theorem 4.1: Suppose $s \in L(G)$ is executed, where s is arbitrarily long, and $F_i \in s$. According to Proposition 4.1, all possible orders of observable events that survive the information update rule at the coordinator site as a result of the execution of s correspond to some $P(s')$, where $s' \in L(G)$. Furthermore s and s' satisfy Properties 1–4 of Definition 4.1 (cf. Remark 4). Let $S'(s)$ be the set of all such traces $s' \in L(G)$.

Sufficiency ($\Leftarrow$). Suppose $L(G)$ has no ambiguous traces with respect to all failure types. From the definition of ambiguous traces (Definition 4.1), we conclude that F_i belongs to all $s' \in S'(s)$. Consequently, since $L(G)$ is diagnosable with respect to Protocol 1, F_i is diagnosed by the coordinator.

Necessity ($\Rightarrow$). Suppose Protocol 1D performs as well as Protocol 1 which by assumption diagnoses all failure types. Since $F_i \in s$ by assumption, then F_i is diagnosed by Protocol 1D. Therefore, F_i belongs to all traces $s' \in S'(s)$. Hence, s is not an ambiguous trace with respect to the projections P_1 and P_2 and the failure type F_i. Since F_i is arbitrary it follows that s is not ambiguous with respect to the projections P_1 and P_2 and all failure types. ■

Proposition 4.1 has a significant implication on the implementation of Protocol 1D. Since the update rule only reconstructs states of the centralized (extended) diagnoser, the maximum number, at one time, of surviving updates is bounded by the order of the state space of the extended diagnoser. Although this is a very loose bound, it proves that the implementation of the protocol requires finite memory.

5. Protocol 2D: A Second Coordinated Decentralized Protocol

In this section, we modify another protocol considered in Debouk et al. (2000) to account for communication delays. We will refer to the modified protocol as Protocol 2D (where 2D stands for the fact that is a modification of Protocol $\underline{2}$ of Debouk et al., 2000 that allows for communication $\underline{\text{D}}$elays). A key feature of Protocol 2 studied in Debouk et al. (2000) is the following: if the system has no failure ambiguous traces (Definition 18 in Debouk et al., 2000; see Section 5.4 hereafter), and if the communicated messages are received in the correct order by the coordinator, then the coordinator can identify exactly the same failure types as the centralized diagnoser even when communication between the local sites and the coordinator is not continuous (as is the case when only commonly observed events by both sites are communicated (cf. Section 5.6) or when the coordinator polls the sites (cf. Section 5.7)). The above feature is the key to understanding the performance of Protocol 2D. When communication between the local sites and the coordinator is continuous, but messages are received out of order by the coordinator, the coordinator, as in the case of Protocol 1D, should consider all possible orders according to which the messages may have been sent. For each possible order, it uses the information update rule specified by Protocol 2. A failure is diagnosed only when according to all possible sorted out orders it is certain that the failure has occurred. We determine conditions under which Protocol 2D performs as well as the centralized diagnostic scheme of Sampath et al. (1995).

The following sub-sections define Protocol 2D, namely, the diagnostic information generated at the local sites, the communication rules used by the local sites, and the decision rule used by the coordinator to infer the occurrence of failures. The memory requirements for Protocol 2D and two modifications of the protocol, in the case of commonly observed events and non-continuous communication between the local sites and the coordinator, are also discussed.

5.1. Diagnostic Information at Local Sites

As in the case with Protocol 2, diagnosers are implemented at local sites. The diagnostic information available at each site is provided by the state of the diagnoser, and is refined by its unobservable reach (cf. Definition 19 in Debouk et al., 2000).

5.2. Communication Rules

The communication rules of Protocol 2 are also used for Protocol 2D. Communication rule [CRi], $i = 1, 2$, specifies that after the agent at site i observes an event $\sigma \in \Sigma_{oi}$, it communicates to the coordinator the corresponding state q_i of its diagnoser G_{di}, its unobservable reach $UR_i(q_i)$ with respect to $\Sigma \backslash \Sigma_{oi}$, and a status bit, SB_i, that takes the values $SB_i = 1$ when $\sigma \in \Sigma_{oj}, j \in \{1, 2\}, j \neq i$, or $SB_i = 0$ when $\sigma \notin \Sigma_{oj}$.

5.3. Decision Rule

The coordinator's decision rule is composed of the following three steps:

1. Store all incoming messages at the coordinator site, and sort out all possible orders in which these messages could have been generated by the local sites.

2. Apply the information update rule, as defined in Section 5.1.3 in Debouk et al. (2000), to each and every possible order, and retain all orders that result in a non-empty update (intersection).

3. Compare the failure properties of all surviving updates from step 2, and declare the occurrence of a failure when all these updates are certain of the occurrence of the failure.

These steps are discussed in the following three sub-sections. The following analysis assumes that there are no events commonly observed by sites 1 and 2. The case where there are events that are commonly observed at both sites is briefly discussed in Section 5.6.

5.3.1. Sorting Out Possible Orders

Messages are stored and sorted out in the same way as Protocol 1D.

5.3.2. Application of the Update Rule

Before describing the information update rule of Protocol 2D, we first recall from Section 5.2.3 of Debouk et al. (2000) the structure of the coordinator. For every possible order that is sorted out the coordinator has few registers, besides the register C that holds its diagnostic information. The registers are used to store the latest states and unobservable reaches of G_{d1} and G_{d2}. At reset, registers are initialized with the initial states and unobservable reaches of G_{d1} and G_{d2}. The information update rule of Protocol 2 consists of intersecting the state of the diagnoser that observed the last event with the unobservable reach of the other diagnoser and store the information in the register C.

The coordinator applies the information update rule to each one of the orders that are sorted out. For every sorted out order that survives the update rule, the coordinator keeps the last update along with the corresponding states and unobservable reaches. This is possible by definition of the update rule that only requires information from the last update to generate the new one. If a specific order of messages results in an empty intersection, the coordinator rejects that as an impossible order. The information is stored by the coordinator until the next instant of time when new sorted out orders are extracted (as a continuation of the already existing ones). Then the same procedure as above is applied to these new orders and the new updates replace the old ones that are discarded. The implementation of the sorting procedure and update rule follows closely those of Protocol 1D.

5.3.3. Diagnosing Failures

If all the information updates that result from applying the procedure described in Sections 5.3.1 and 5.3.2 are certain that a failure F has occurred, then the coordinator declares that F has occurred and broadcasts this information to the failure recovery module. Otherwise, a diagnostic decision is postponed.

5.4. Diagnostic Properties of Protocol 2D

The decision rule discussed in Section 5.3 has the following two features: (1) In comparison with Protocol 2, the memory requirements at the coordinator site may increase considerably because the coordinator has to store all the information updates, discussed in Section 5.3.2, and the corresponding diagnoser states and unobservable reaches. However, the amount of memory required remains finite since the models used are finite-state automata (cf. Section 5.5). (2) The coordinator identifies a failure only when that failure is identified by all surviving information updates. Therefore, we expect that, in general, the performance of Protocol 2D will not be the same as that of Protocol 2 where global order is preserved. Interestingly enough, we prove next that the absence of failure-ambiguous traces is a condition sufficient to ensure that Protocol 2D performs as well the centralized diagnostic scheme of Sampath et al. (1995). We note that Protocol 2 in Debouk et al. (2000) performs as well as the centralized diagnostic scheme of Sampath et al. (1995) under the same condition. Thus, relaxing the assumption on global ordering of messages at the coordinator's site does not result in any performance degradation of the protocol (with the notable exception of additional memory requirements). To proceed with the analysis of Protocol 2D we first recall the definition of failure-ambiguous traces. We refer the reader to Example 9 of Debouk et al. (2000) for an illustration of the concept.

DEFINITION 5.1 *A trace $s \in L(G)$ is said to be failure-ambiguous with respect to the projections P_1 and P_2 and the failure type F_i if there exist two traces, s' and s'' in $L(G)$ such that s' and s'' are arbitrarily long, not necessarily distinct, and the following is true:*

1. $P_1(s) = P_1(s')$ but $P(s) \neq P(s')$,

2. $P_2(s) = P_2(s'')$ but $P(s) \neq P(s'')$,

3a. $F_i \in s$ but $F_i \notin s'$.

3b. $F_i \in s$ but $F_i \notin s''$.

4. s' and s'' share the same failure properties, i.e., a failure of type F_j, $j \in \{1, \ldots, m\}$, $j \neq i$, belongs to s' if and only if a failure of type F_j (not necessarily the same failure event) belongs to s''.

The next theorem summarizes the main result concerning the diagnostic performance of Protocol 2D.

THEOREM 5.1 *Under the assumption that messages are received by the coordinator at most ''one-step out of order'', Protocol 2D eventually identifies all failure types that are detected by the centralized diagnostic scheme of Sampath et al. (1995) if there are no failure-ambiguous traces (with respect to all failure types).*

Proof of Theorem 5.1: Consider that the system is executing the trace $s_0 u_1 a u_2 b u_3 c := s u_3 c$ where $a \in \Sigma_{o1}$, $b \in \Sigma_{o2}$, $c \in \Sigma_{o1}$, $F_i \in s_0$, and $u_1, u_2, u_3 \in \Sigma_{uo}^*$. Denote by x, y, and z the messages generated by the occurrence of events a, b, and c, respectively. Without loss of generality assume that the messages are received in the following order: x, y, then z. This corresponds to the case presented in the third row of Table 1 (the fourth row is the symmetric case of the third row while by inspection one realizes that the cases presented in rows one and two are respectively sub-cases of those presented in rows three and four). Also assume that s is arbitrarily long, i.e., s cannot be a failure-ambiguous trace. Let q_{11} and q_{12} be the states of the diagnoser G_{d1} after the execution of the the events a and c, respectively, and q_2 be the state of the diagnoser G_{d2} after the execution of the event b. The correct order in which these messages are sent is xyz, and from Table 1 and the sorting procedure presented in Section 4.3.1 we have that the coordinator considers the following orders: xy, yx, and xzy. Denote by C_1, C_2, and C_3 the coordinator state resulting from applying the update rule to the orders xy, yx, and xzy, respectively.

For the first order xy we have (from Table 3 in Debouk et al., 2000)

$$C_1 = UR_1(q_{11}) \cap q_2 \tag{14}$$

Since this is the correct decision then C_1 is not empty. Moreover since s is not failure-ambiguous, C_1 is F_i-certain by the results of Proposition 4 (Case 3) of Debouk et al. (2000).

For the second order, namely yx, the coordinator considers results in

$$C_2 = UR_2(q_2) \cap q_{11} \tag{15}$$

from Table 3 in Debouk et al. (2000). Assume that C_2 is not empty. Then, there exist at least two traces s_1' (ending with the event a in Σ_{o1}) and s_2' (ending with an event in Σ_o) sharing the same failure properties such that

$$P_1(s_1') = P_1(s) \tag{16}$$

$$P_2(s_2') = P_2(s) \tag{17}$$

$$\delta(x_0, s_1') = \delta(x_0, s_2') \tag{18}$$

Since by assumption there are no failure-ambiguous traces then either $P(s) = P(s_1')$ (negation of condition 1 in Definition 5.1), or $P(s) = P(s_2')$ (negation of condition 2 in Definition 5.1), or s, s_1', s_2' share the same failure properties (negation of conditions $3a$, $3b$ in Definition 5.1), or combinations of these facts are true. Condition 4 in Definition 5.1 cannot be violated because then C_2 is empty. $P(s) = P(s_1')$ is impossible, as it contradicts the fact that s_1' ends with the event a in Σ_{o1}. If $P(s) = P(s_2')$ then necessarily s, s_1', s_2' share the same failure properties since the language is diagnosable with respect to Σ_o and the failure partition, and hence C_2 is F_i-certain. If s, s_1', s_2' share the same failure properties then we have that C_2 is F_i-certain.

For the third order, namely xzy, the coordinator considers results in

$$C_3 = UR_1(q_{12}) \cap q_2 \tag{19}$$

from Table 3 in Debouk et al. (2000). Assume that C_3 is not empty. Then, there exist at least two traces t_1' (ending with an event in Σ_o) and t_2' (ending with the event b in Σ_{o2}) sharing the same failure properties such that

$$P_1(t_1') = P_1(su_3c) \tag{20}$$

$$P_2(t_2') = P_2(s) \tag{21}$$

$$\delta(x_0, t_1') = \delta(x_0, t_2') \tag{22}$$

But $su_3c \in L(G)$ and $P_2(su_3c) = P_2(s)$; consequently,

$$P_2(t_2') = P_2(su_3c) \tag{23}$$

Since there are no failure-ambiguous traces, then either $P(su_3c) = P(t_1')$, or $P(su_3c) = P(t_2')$, or su_3c, t_1', t_2' share the same failure properties, or combinations of these facts are true. If $P(su_3c) = P(t_1')$ then necessarily su_3c, t_1', t_2' share the same failure properties since the language is diagnosable with respect to Σ_o and the failure partition, and hence C_3 is F_i-certain. $P(su_3c) = P(t_2')$ is impossible as it contradicts the fact that t_2' ends with an event in Σ_{o2}. If su_3c, t_1', t_2' share the same failure properties then we have that C_3 is F_i-certain.

From the above analysis we conclude that C_1, C_2, and C_3 are all F_i-certain. Consequently, the failure type F_i is diagnosed by Protocol 2D. Since F_i was arbitrarily chosen, Protocol 2D can diagnose any failure type under the assumption that there are no failure-ambiguous traces. Consequently Protocol 2D performs as well as the centralized diagnostic scheme of Sampath et al. (1995). ■

Note here that a trace that is not failure-ambiguous may contain prefixes that are failure-ambiguous. Hence, the above proof holds true once the system has executed enough events to overcome the failure-ambiguous sub-traces. Therefore, Protocol 2D identifies, under the condition of this theorem, the same failures as the centralized diagnostic scheme of Sampath et al. (1995) but with higher delay.

We conjecture that the result of Theorem 5.1 still holds even when messages are received at the coordinator at most ''n-steps out of order''. We expect that the delays associated with diagnostic decisions and the memory requirements at the coordinator's site will increase with n.

5.5. Memory Issues in Protocol 2D

In contrast to Protocol 1D where all non-empty updates (intersections) result in a state of the centralized extended diagnoser, possible orders that are considered by the sorting procedure used by step (1) of the decision rule of Protocol 2D may result in a non-empty update (intersection) that is different from any state of the centralized diagnoser (Note 4). Therefore, one may end up by having a considerably large number of possible updates to be saved. To avoid that problem, we suggest to use some side information at the coordinator's site. All possible states of the coordinator, when global order is preserved, following any possible legal behavior of the system are computed off-line and stored at the coordinator site. This side information helps reducing the number of possible updates to be stored: in case an update results in a state that does not belong to the side information this update is rejected. By doing so we are only tracking legal behavior that is exhibited under Protocol 2, and most importantly we only require finite memory to hold updates since these updates are bounded by the order of the state space of G_{test2}, where G_{test2} is a FSM that generates, among other information, the states of the coordinator when global order is preserved (cf. Section 5.4 of Debouk et al., 2000). As explained in the case of Protocol 1D, the suggested bound is a loose one, nevertheless it proves that the implementation of the protocol requires finite memory.

5.6. Procedure Using Common Events

As discussed in Remark 3, if there are events that are commonly observed by both sites then these events can be used as a synchronization mechanism. Since local order is preserved, the coordinator knows how to order the messages that are due to commonly observed events. As a result of the information update rule at the coordinator's site, messages generated by commonly observed events need to contain only the states of the local diagnosers. Therefore, under the assumption that commonly observed events are executed frequently along all traces, the protocol can be modified as follows: local sites use the diagnosers to generate their diagnostic information; they communicate their diagnosers' states to the coordinator only after the occurrence of commonly observed events, hence the communication is not continuous; the decision rule of the coordinator is to apply the information update rule as specified in Table 1. Denote by Protocol 2D-C (where C stands for commonly observed events) the above specified protocol. Then, we have the following result.

THEOREM 5.2 *If there are no failure-ambiguous traces (with respect to all failure types), Protocol 2D-C eventually identifies the same failures as the centralized diagnostic scheme of Sampath et al. (1995).*

Proof of Theorem 5.2: Since local order is preserved, messages sent by distinct sites regarding the same commonly observed event can be easily matched. By inspecting the information update rule (Table 3 in Debouk et al., 2000), one realizes that the update following a commonly observed event only requires information received by the messages and no past information is needed. Then according to Theorem 7 in Debouk et al. (2000), if there are no failure-ambiguous traces the update rule eventually results in an F_i-certain coordinator state following the execution by the system of an event that belongs to the failure type F_i. ■

We note that the word eventually is also used in the statement of the theorem for the same reasons explained after Theorem 5.1.

This variation of the protocol saves on communication, processing power, and memory storage (the same memory storage as in the case of Protocol 2 is needed) at the expense of delaying the diagnostic decision in the case where the frequency of occurrence of commonly observed events is low. Note here that the ''one-step out of order'' assumption is not needed as long as communication delays are bounded.

5.7. Polling Procedure

Another approach is to advise the coordinator to poll the sites requesting each site to communicate its current state and unobservable reach plus the status bit specifying whether the event that led to the current state is observed by the other site or not. Polling both sites is either arbitrarily triggered or based on the coordinator's current state. It is assumed that there is a finite bound between two polling instances, that is, although communication is not continuous enough polling instances occur to diagnose the failures. Denote this protocol by Protocol 2D-P (where P stands for polling). Under the assumption that the system does not execute a new observable event (in Σ_o) between the instant the polling message is generated and the instants it is received by the sites, we have the following result.

THEOREM 5.3 *Under the assumption that the system does not execute a new observable event (in Σ_o) between the instant the polling message is generated and the instants it is* *received by the sites, Protocol 2D-P eventually identifies the same failures as the centralized diagnostic scheme of Sampath et al. (1995) if there are no failure-ambiguous traces (with respect to all failure types).*

Proof of Theorem 5.3: Consider that the system is executing the trace $s_0u_1au_2b$ where $a, b \in \Sigma_o$, $F_i \in s_0$, and $u_1, u_2 \in \Sigma_{uo}^*$. Assume the coordinator polls the sites right after the event b was observed. By assumption the polling message is received at the two sites before the system executes a new observable event. Also assume that s is arbitrarily long, i.e., s cannot be a failure-ambiguous trace. Since s is not failure-ambiguous we have from the results of Theorem 6 in Debouk et al. (2000) that the failure F_i is diagnosed. If b is a commonly observed event then there is no need to sort out messages and the coordinator just intersects the states of the two diagnosers. If both a and b were observed by only one site, then the correct order in which events were executed by the system is known and consequently the coordinator applies the correct information update rule by intersecting the state of the diagnoser that sent the message generated by the occurrence of event b with the unobservable reach of the other diagnoser's message. The only case where the coordinator cannot figure out the correct order in which the events were executed by the system is when $a \in \Sigma_{o1}$ and $b \in \Sigma_{o2}$, or $a \in \Sigma_{o2}$ and $b \in \Sigma_{o1}$. Without loss of generality assume that $a \in \Sigma_{o1}$ and $b \in \Sigma_{o2}$. Denote by x and y the messages generated by the occurrence of events a and b, respectively. Without loss of generality assume that the messages are received in the following order: x then y. Let q_1 and q_2 be the states of the diagnosers G_{d1} and G_{d2} after the execution of the events a and b, respectively. The coordinator considers the following orders: xy, yx. Denote by C_1 and C_2 the coordinator state resulting from applying the update rule to the orders xy and yx, respectively.

For the first order xy we have (from Table 3 in Debouk et al., 2000)

$$C_1 = UR_1(q_1) \cap q_2 \tag{24}$$

Since this is the correct decision then C_1 is not empty. Moreover since s is not failure-ambiguous, C_1 is F_i-certain by the results of Proposition 4 (Case 3) of Debouk et al. (2000).

For the second order, namely yx, the coordinator considers results in

$$C_2 = UR_2(q_2) \cap q_1 \tag{25}$$

from Table 3 in Debouk et al. (2000). Assume that C_2 is not empty. Then, there exist at least two traces s_1' (ending with the event a in Σ_{o1}) and s_2' (ending with an event in Σ_o) sharing the same failure properties such that

$$P_1(s_1') = P_1(s) \tag{26}$$

$$P_2(s_2') = P_2(s) \tag{27}$$

$$\delta(x_0, s_1') = \delta(x_0, s_2') \tag{28}$$

Since by assumption there are no failure-ambiguous traces then either $P(s) = P(s_1')$

(negation of condition 1 in Definition 5.1), or $P(s) = P(s'_2)$ (negation of condition 2 in Definition 5.1), or s, s'_1, s'_2 share the same failure properties (negation of conditions $3a$, $3b$ in Definition 5.1), or combinations of these facts are true. Condition 4 in Definition 5.1 cannot be violated because then C_2 is empty. $P(s) = P(s'_1)$ is impossible, as it contradicts the fact that s'_1 ends with an event in Σ_{o1}. If $P(s) = P(s'_2)$ then necessarily s, s'_1, s'_2 share the same failure properties since the language is diagnosable with respect to Σ_o and the failure partition and hence C_2 is F_i-certain. If s, s'_1, s'_2 share the same failure properties then we have that C_2 is F_i-certain.

From the above analysis we conclude that C_1 and C_2 are all F_i-certain. Consequently, the failure type F_i is diagnosed by Protocol 2D-P. Since F_i was arbitrarily chosen, Protocol 2D-P can diagnose any failure type under the assumption that there are no failure-ambiguous traces. Consequently Protocol 2D-P performs as well as the centralized diagnostic scheme of Sampath et al. (1995). ■

We note again that a trace that is not failure-ambiguous may contain prefixes that are failure-ambiguous. Hence, the above-presented proof holds true once the system has executed enough events to overcome the failure-ambiguous sub-traces. Therefore, Protocol 2D-P identifies, under the condition of this theorem, the same failures as the centralized diagnostic scheme of Sampath et al. (1995) but with higher delay.

Protocol 2D-P saves on communication and processing power and memory storage (a comparable memory storage to the case of Protocol 2 is needed). It avoids as well the delaying of diagnostic decisions when the frequency of occurrence of commonly observed events is low. Also, the ''one-step out of order'' assumption is not needed as long as communication delays are bounded.

6. Concluding Remarks

In this paper, we have extended the theory of diagnosability of decentralized discrete event systems. We have presented two coordinated decentralized protocols, namely Protocol 1D and Protocol 2D, that are capable, each under certain conditions, of diagnosing all failure types diagnosed by the centralized diagnostic scheme of Sampath et al. (1995). The on-line diagnostic process is carried through the diagnosers implemented at the local sites, i.e., the scheme is indeed implemented in a decentralized fashion.

The key features of Protocol 1D are: (1) it achieves the same performance as the centralized diagnostic scheme of Sampath et al. (1995) when there are no ambiguous traces. Hence, the absence of global ordering of the messages received at the coordinator's site prevents the protocol from achieving the same performance as Protocol 1 of Debouk et al. (2000) which achieves the same diagnostic performance as the diagnostic scheme of Sampath et al. (1995) under no restrictions on the system structure. (2) The delay of the diagnostic decision of Protocol 1D is higher than the delay of the centralized diagnostic scheme of Sampath et al. (1995) and that of Protocol 1 of Debouk et al. (2000). (3) The memory required at the coordinator's site to implement the protocol is larger than that required in Protocol 1 of Debouk et al. (2000).

The absence of ambiguous traces is a necessary and sufficient condition for Protocol 1D to diagnose all failure types. If the behavior of the system exhibits such traces, then changing the observation spaces at the local sites can overcome the ambiguity constraint. Algorithms to achieve that can be devised and are the topic of future research.

The key features of Protocol 2D are: (1) it achieves the same performance as the centralized diagnostic scheme of Sampath et al. (1995) when there are no failure-ambiguous traces. That is, the absence of global ordering of the messages received at the coordinator's site does not degrade this aspect of the protocol's performance (compared to Protocol 2 of Debouk et al., 2000). (2) The delay of its diagnostic decision is higher than the delay of the centralized diagnostic scheme of Sampath et al. (1995). (3) The memory required at the coordinator's site to implement the protocol is larger than that required in Protocol 2 of Debouk et al. (2000). The first feature of Protocol 2D is a bit surprising, whereas its last two features are not unexpected.

Protocol 1D (as well as Protocol 1) requires continuous communication between the coordinator and the local sites: the update rule at any instant of time requires information from the previously updated coordinator state to generate the new coordinator state. Therefore, interruption of communication is not feasible under Protocol 1D, and consequently savings on communication cannot be achieved. On the contrary, in the case of Protocol 2D (and Protocol 2 as well) savings on communication may be achieved since interruption of communication is possible (either through only communicating commonly observed events or polling). Two modifications of Protocol 2D, namely Protocols 2D-C and 2D-P, result in communication and memory savings while maintaining the same diagnostic performance. However, the diagnostic delays are further increased in Protocol 2D-C, whereas an additional assumption, that may be unrealistic in some cases, is required by Protocol 2D-P.

The approach we used in this paper to account for communication delays in the case of Protocol 1D and Protocol 2D requires a considerable amount of additional memory and processing at the coordinator site. In contrast, if time stamps were available, Protocols 1 and 2 of Debouk et al. (2000) would work without any modifications. However, in that case local clocks would need to be synchronized, and this requires additional processing and memory storage at the local sites. Therefore, a tradeoff exists between these two mechanisms to handle communication delays, and the type of application usually determines the mechanism to use.

In this paper we only considered the ''one-step out of order'' assumption for reasons of simplicity and compactness. We conjecture that the same results should hold in the case of ''n-step out of order'' messages. The general sorting rule in such a case is to wait for the arrival of $n+2$ messages and then figure out all possible orders. By doing so the number of these orders increases considerably, but most importantly it remains finite. Afterwards, we apply the update rule for the first two messages in every possible order and we follow the same procedure as in the case of the ''one-step out of order'' assumption. Consequently, the diagnostic decision is further delayed and memory requirements increase drastically; nevertheless we believe the results presented here should hold.

Finally, although we considered the generic case of a coordinator with two sites, the results are scalable to the case where there are m sites. By realizing that the ''one-step out of order'' assumption or even the ''n-step out of order'' assumption is not affected by the number of available sites, the extension to *m* sites could be justified in the same way it was justified in Section 7.3 of Debouk et al. (2000) for the case where global order is preserved.

Acknowledgment

This research was supported in part by NSF grants ECS-0080406 and CCR-0082784, by the Department of Defense Research and Engineering (DDR&E) Multidisciplinary University Research Initiative (MURI) on Low Energy Electronics Design for Mobile Platforms and managed by the Army Research Office (ARO) under grant ARO DAAH04-96-1-0377, and by AFOSR grant F49620-96-1-0028.

Notes

1. Whenever we say that there exists a trace s of arbitrarily long length having a given property, we mean the following: for all integers n, there exists s, such that the length of s is greater than n and s possesses the given property.
2. The proof assumes there are no commonly observed events since this constitutes the general case. The proof for the case where there are commonly observed events is a special case (cf. Remark 3) and can be replicated using the same arguments.
3. Proving the result for the order yx requires considering a trace having $n + 2$ observable events, while proving the result for the order xzy requires considering a trace having $n + 3$ observable events. This is due to the fact that the uncovered order in the case of yx includes two additional messages while that of the order xzy includes three additional messages.
4. We note that these updates cannot create spurious diagnoses as the state of the coordinator is a superset of that of the centralized diagnoser (cf. equations (24), (32), and (33) in Debouk et al., 2000) and they share the same failure certainty properties as proven in Proposition 4 of Debouk et al. (2000).

References

Aghasaryan, A., Fabre, E., Benveniste, A., Boubour, R., and Jard, C. 1998. Fault detection and diagnosis in distributed systems: An approach by partially stochastic petri nets. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, August, 203–231.

Baroni, P., Lamperti, G., Pogliano, P., and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110: 135–183.

Bertsekas, D., and Gallager, R. 1992. *Data Networks*. Englewood Cliffs, NJ: Prentice Hall.

Deb, S., Mathur, A., Willett, P., and Pattipati, K. R. 1998. De-centralized real-time monitoring and diagnosis. In *Proc. IEEE Conf. on Systems, Man and Cybernetics*, October, 2998–3003.

Debouk, R. 2000. *Failure Diagnosis of Decentralized Discrete Event Systems*. PhD thesis, Electrical Engineering and Computer Science Department, The University of Michigan. Available at http://www.eecs.umich.edu/umdes.

Debouk, R., Lafortune, S., and Teneketzis, D. 2000. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications* 10: 33–86.

Fabre, E., Benveniste, A., Jard, C., Ricker, S. L., and Mark Smith. 2000. Distributed state reconstruction for discrete event systems. In *Proc. 39th IEEE Conf. on Decision and Control*, December 2252–2257, Sydney, Australia.

Holloway, L., and Chand, S. 1994. Time templates for discrete event fault monitoring in manufacturing systems. In *Proc. 1994 American Control Conference*, 701–706.

Mohindra, S., and Clark, P. A. 1993. A distributed fault diagnosis method based on digraph models: Steady-state analysis. *Computers and Chemical Engineering* 17(2): 193–209.

Pencolé, Y. 2000. Decentralized diagnoser approach: Application to telecommunication networks. In *Proc. of DX'2000, Eleventh International Workshop on Principles of Diagnosis*, June, 185–192.

Pouliezos, A. D., and Stavrakakis, G. S. 1994. *Real time fault monitoring of industrial processes*. Boston, MA: Kluwer Academic Publishers.

Ricker, S. L., and Fabre, E. 2000. On the construction of modular observers and diagnosers for discrete-event systems. In *Proc. 39th IEEE Conf. on Decision and Control*, December, Sydney, Australia.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Trans. Automat. Contr.* 40(9): 1555–1575.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. 1996. Failure diagnosis using discrete-event models. *IEEE Trans. Contr. Syst. Tech.* 4(2): 105–124.

Sengupta, R., 1998. Diagnosis and communication in distributed systems. In *Proc. of WODES 1998, International Workshop on Discrete Event Systems*, 144–151, August. London, England: IEE.

Stark, W. E., Wang, H., Worthen, A., Lafortune, S., and Teneketzis, D. 2002. Low energy wireless communication network design. *IEEE Wireless Communication Magazine*, August 60–72.

Schmid, U., Guest Editor. 1997. Special issue on global time in large scale distributed real time systems. *Real Time Systems* 12(I, II, and III): 1–351.

Willsky, A. 1996. A survey of design methods for failure detection in dynamic systems. *Automatica* 12: 601–611.