On the Effect of Communication Delays in Failure Diagnosis of Decentralized Discrete Event Systems [1]

Rami Debouk, Stéphane Lafortune, and Demosthenis Teneketzis
Department of Electrical Engineering and Computer Science
The University of Michigan
1301 Beal Avenue
Ann Arbor, MI 48109–2122, USA
www.eecs.umich.edu/umdes

Submitted to the Journal of Discrete Event Dynamical Systems: Theory and Applications

Keywords Failure Diagnosis, Discrete Event Systems, Decentralized Information, Diagnostic Protocols, Communication Delays

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 [7] 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 [7] 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 [17].

1 Introduction

Failure detection and isolation is an important task in the automatic control of large complex systems. Diagnosing systems not only improves their performance and productivity, but also protects life and property. For these reasons, approaches to failure diagnosis have been extensively studied in the literature. We refer the interested reader to [23], [12] and the introduction of [17] 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

[1] This research was supported in part by NSF grants ECS-9509975 and ECS-0080406, and by the Department of Defense Research & 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. The authors can be reached by email at ridebouk@eecs.umich.edu, stephane@eecs.umich.edu and teneketzis@eecs.umich.edu.
September 12, 2000

(computer and communication networks, manufacturing 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. Since this information is exchanged in real-time and over channels of limited capacity, there are propagation delays, along with faults and transmission errors. Thus, the information available to each station is incomplete, delayed, and possibly erroneous. Most approaches to failure diagnosis suggested in the literature do not apply directly to informationally decentralized systems, hence the need to develop diagnostic methodologies for these systems. This fact is also recognized in [1], [6], [9], and [19].

In [1] the authors use Petri nets to model concurrent alarm indications in large distributed systems. The diagnosis problem is defined as the computation of the most likely history of the net given a sequence of observed alarms. In [6], the authors propose to implement a centralized real time diagnosis algorithm, TEAM-RT, in a distributed fashion. The system they monitor is decomposed into sub-systems. Each sub-system is monitored by a local unit running TEAM-RT. Each local unit is cognizant of its immediate neighbors, i.e., units monitoring sub-systems that affect or may be affected by the monitored sub-system. By allowing exchange of information among neighboring local units, the authors propose an algorithm that achieves the global diagnosis, that is, the diagnosis of the system as if one unit, running TEAM-RT, is collectively monitoring all the sub-systems. In [9], the authors present a distributed fault monitoring method for manufacturing systems called time templates monitoring. A time template, by definition, consists of a trigger event and a set of consequences. There is a set of distributed interconnected processors available for monitoring. The set of templates modeling the system is distributed among the processors. The processors use the sets of timing and sequencing relationships available through their assigned templates to establish when events are expected to occur, and to determine if an event has occurred in the appropriate context of some previous event. In [19], the authors discuss diagnosis problems in distributed systems composed of several spatially separated sites. At every site, there exists a diagnoser that partially observes the system and is in charge of diagnosing faults associated with the site. Diagnosers are allowed to exchange information. The authors characterize the class of distributed systems where there exists no inter-diagnoser messaging scheme that can replicate the information available to a centralized diagnoser.

Modular approaches to failure diagnosis have been suggested in the literature as well [3, 8, 10, 11, 13]. Modular approaches use local models of work stations or sub-systems, instead of global models as is the case with decentralized systems, to generate diagnostic information regarding the global system. Although a modular architecture approach to failure diagnosis differs conceptually from a decentralized one, yet it could be classified as an informationally decentralized approach. In [3] the authors present a modular technique, amenable to parallel implementation, for the diagnosis of large-scale, distributed, asynchronous event-driven systems which they refer to as active systems. An active system can be modeled as a network of communicating automata. The main goal of the diagnostic technique is the reconstruction of the behavior of the active system starting from a set of observable events. First, the technique generates many representations of parts of the active system based on available observation. Next, these representations are merged to generate a unified representation by using the history of observable events. Finally, the diagnostic information is generated on the basis of fault events possibly incorporated within the reconstructed behavior. In [10], the authors develop a highly modular fault diagnosis methodology using digraph models of process behavior.

The method is developed from graph theory and uses off-line analysis of digraph structure to reduce the on-line computation wherever possible. The on-line work is designed for a distributed implementation in which each sensor and controller currently in an abnormal state examines the states for a small number of adjacent measurements to suggest possible faults, including both measured and unmeasured variables. In [11][2], the authors are interested in diagnosing a telecommunication network composed of many sub-systems. They build local diagnosers for these sub-systems. Each local diagnoser generates local diagnosis. All available local diagnoses are combined, using the history of observable events and the models of the sub-systems, to generate a global diagnosis. The combination is done while minimizing the computation of the overall diagnosis.

The philosophy of the approach we are suggesting is totally different from that of [9] and [10]. Although similarities exist between our approach and that of [3], [6], [11], and [19], the modeling and/or informational assumptions of [3], [6], [11], and [19] and our work are clearly different.

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

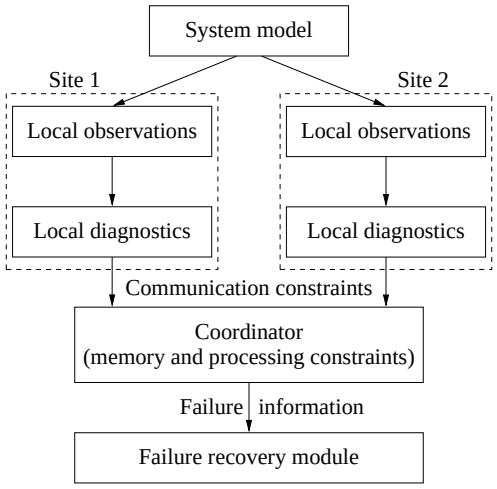

Figure 1: Coordinated decentralized architecture

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 [7] 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

[2]It is worth mentioning that the work in [11] has been motivated by our work on failure diagnosis with decentralized information that appeared in [7].

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 [7], 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 [17]. 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 [17] whereas Protocol 1 performs as well as the diagnostic scheme of [17] irrespective of the system structure. Protocols 2 and 2D perform as well as the centralized diagnostic scheme of [17] 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 [17], the protocol requires more memory than Protocol 1.

This paper is organized as follows. Section 2 introduces some preliminary definitions, assumptions, and notation needed for the development of the technical results of the paper. 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

This section briefly introduces some preliminary definitions, assumptions, and notation needed for the development of the technical results of this paper. The reader is referred to [7, 17, 18] for more details.

2.1 The system model and diagnoser

The system to be diagnosed is modeled as a deterministic finite state machine (FSM)

$$G = (X, \Sigma, \delta, x_0) \tag{1}$$

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. The model G accounts for the normal and failed behavior of the system. The behavior of the system is described by the prefix-closed language $L(G)$ generated by G. $L(G)$ is a subset of Σ^*, where Σ^* denotes the Kleene closure of the set Σ. In this paper we will use the language $L(G)$, or simply L, and the system interchangeably.

The event set Σ is partitioned as $\Sigma = \Sigma_o \cup \Sigma_{uo}$ where Σ_o represents the set of observable events and Σ_{uo} the set of unobservable events. The projection $P : \Sigma \rightarrow \Sigma_o$ is defined in the usual manner [5]. Let $\Sigma_f \subseteq \Sigma$ denote the set of failure events which are to be diagnosed. We assume, without loss of generality,

that $\Sigma_f \subseteq \Sigma_{uo}$. Our objective is to identify, within the context of the coordinated decentralized architecture, the occurrence, if any, of the failure events, given that in the traces generated by the system, only the events in Σ_o are observed. In this regard, we partition the set of failure events into disjoint sets corresponding to different failure types

$$\Sigma_f = \Sigma_{f1} \cup \Sigma_{f2} \cup \ldots \cup \Sigma_{fm}. \tag{2}$$

Let Π_f denote this partition. Hereafter, when we write that a failure of type F_i has occurred, we will mean that some event of the set Σ_{fi} has occurred.

The diagnoser is a deterministic FSM built from the system model G. This machine is at the core of the diagnostic methodology of [16, 17, 18] where the information is centralized. It is used to analyze the diagnosability properties of G and to perform diagnostics when it observes on-line the behavior of the system. Diagnosers and extended diagnosers [14] are also used in realizations of the coordinated decentralized architecture of Figure 1, presented in this paper. Definitions regarding diagnosers and extended diagnosers are relegated to Appendices A and B.

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 [15, 18]). 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.

Following the above discussion, in order to define diagnosability for coordinated decentralized systems, we need to account for the rules used to generate local diagnostic information together with the associated communication rules and the coordinator's decision rule. In the proposed coordinated architecture the local agents do not interact with one another; they only communicate with the coordinator that is assigned the task of detecting and isolating failures. Let C denote the coordinator's diagnostic information. We have the following three definitions from [7].

Definition 2.1 *Within the context of the coordinated decentralized architecture described above and depicted in Figure 1, a* **protocol** *is defined by the diagnostic rules used at the local sites, the rules employed by the local sites to communicate their diagnostic information to the coordinator, and the coordinator's decision rule.*

Definition 2.2 *The coordinator's diagnostic information C is said to be* **F_i-certain** *if based on C, the coordinator is certain that a failure of type F_i has occurred.*

Note that the diagnostic information C is protocol-dependent.

Definition 2.3 *A prefix-closed and live language L is said to be* **diagnosable** *under a protocol, a set of projections P_1, P_2, and a failure partition Π_f on Σ_f, if the following holds*

$$(\forall i \in \Pi_f)(\exists n_i \in N)(\forall s \in \Psi(\Sigma_{fi}))\ (\forall t \in L/s)(\| t \| \geq n_i \Rightarrow C \text{ is } F_i - certain),$$

where $s \in \Psi(\Sigma_{fi})$ denotes the fact that the last event of s is a failure event of type F_i, $L\backslash s$ denotes the post-language of L after s, and $\| t \|$ denotes the length of trace t.

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 investigate diagnosability properties of the above architecture under the following assumptions.

A1 $L(G)$ is live[3].

A2 G has no cycles of unobservable events with respect to either Σ_{o1} or Σ_{o2}.

A3 $L(G)$ is not diagnosable (in the sense of the centralized setup of [17]) with respect to P_i and Π_f on Σ_f, $i = 1, 2$.

A4 There is reliable communication between the local sites and the coordinator, i.e., all messages sent from a local site are received by the coordinator correctly and in order. However, global order of the messages is not necessarily maintained, i.e., messages sent from various local sites may not be received at the coordinator's site in the order in which they are sent.

A5 The sets of observable events at each site are common knowledge [2, 22] to all sites.

A6 The two sites are allowed to report to the coordinator only a processed version of their raw data.

A7 The coordinator does not have a model of the system, i.e., it does not know the dynamics of the system.

For a discussion of these assumptions the reader is referred to [7]. We note that in this work, in contrast to [7], 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 decentralized diagnostic protocols.*

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

[3] A language L is said to be live if for all $s \in L$, there exists $\sigma \in \Sigma$ such that $s\sigma \in L$.

protocol, for instance TCP, or a data link control layer protocol such as Go Back n or Selective Repeat [4]. 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:

(i) 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 [20]). Such a mechanism is not always practical or feasible. For example, synchronizing clocks may be too constraining in low-energy mobile communication networks [21] and telecommunication networks [8]. 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.

(ii) 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 [8, 11].

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 [7], namely Protocols 1 and 2.

4 Protocol 1D: a coordinated decentralized protocol

In this section, we modify Protocol 1 considered in [7] 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 [7] that allows for communication **D**elays). A key feature of Protocol 1 studied in [7] 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 (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 [17]). 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 [17].

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 defined in Appendix B.

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, $\mathbf{SB_i}$, that takes the values $\mathbf{SB_i} = 1$ when $\sigma \in \Sigma_{oj}$, $j \in \{1, 2\}$, $j \neq i$, or $\mathbf{SB_i} = 0$ when $\sigma \notin \Sigma_{oj}$.

4.3 Decision rule

The coordinator's decision rule is composed of 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 [7], 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. The following analysis assumes that there are no events commonly observed by sites 1 and 2. If there are commonly observed events by both sites the sorting procedure is simplified since the commonly observed events can be used as a synchronization mechanism.

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 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 $x_1z_1y_2$
$x_2y_1z_2$	$x_2y_1z_2$ or $y_1x_2z_2$ or $x_2z_2y_1$	x_2y_1 or y_1x_2 or $x_2z_2y_1$

Table 1: Sorting out possible orders at the coordinator site

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 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 two 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.

4.3.2 Application of the update rule

We briefly describe the update rule of Protocol 1 as defined in [7]. In order to do so we first review from [7] 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 are: *R1*, *R2*, *R3*, *R4*, C_{old}, SB, SB_{1old}, and SB_{2old}. *R1* and *R2* hold the latest states of G^e_{d1} and G^e_{d2}, respectively, *R3* and *R4* hold the latest unobservable reaches of G^e_{d1} and G^e_{d2}, respectively, C_{old} holds the previous coordinator diagnostic information, SB specifies whether the last observed event is observed by both sites (1) or not (0) and SB_{1old}, SB_{2old} provide necessary information to compute the new coordinator diagnostic information (as illustrated below in Table 2). The information update rule is depicted in Table 2. The reader is referred to [7] for an

Last report received from G^e_{d1}						
	SB	SB_1	C	New SB	New SB_{1old}	New SB_{2old}
DR1	0	0	$(R_1 \cap^i_e R_4) \cap_c C_{old}$	0	1	0
DR2	0	1	Wait	1	Unmodified	Unmodified
—	1	0	Impossible	—	—	—
DR3	1	1	$(R_1 \cap^i_e R_2) \cap_c C_{old}$	0	1	1
Last report received from G^e_{d2}						
	SB	SB_2	C	New SB	New SB_{1old}	New SB_{2old}
DR4	0	0	$(R_2 \cap^i_e R_3) \cap_c C_{old}$	0	0	1
DR5	0	1	Wait	1	Unmodified	Unmodified
—	1	0	Impossible	—	—	—
DR6	1	1	$(R_1 \cap^i_e R_2) \cap_c C_{old}$	0	1	1
the i superscript in $\cap^i_e$ depends on the current values of SB_{1old} and SB_{2old}.						

Table 2: Information update rule at the coordinator site

explanation of the rationale behind the information update rule. At reset, *R1* and *R2* are initialized with the initial states of G^e_{d1} and G^e_{d2}, respectively, and *R3* and *R4* hold the initial unobservable reaches of G^e_{d1} and G^e_{d2}, respectively. Before performing any update of the coordinator diagnostic information, the current coordinator diagnostic information is saved into the register C_{old} for later use. Also, the flip-flops (SB, SB_{1old}, and SB_{2old}) are modified once the update of the coordinator diagnostic information is completed.

At any instant when all possible orders are sorted out the coordinator applies to each order of them the information update rule as defined in Table 2. For every possible order that survives the update rule (that is, it results in a non-empty intersection), the coordinator uses a diagnostic register C to hold the new diagnostic update, along with eight registers, as defined in the previous paragraph, 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 (in registers C, *R1*, *R2*, *R3*, *R4*, C_{old}, SB, SB_{1old}, and SB_{2old}) 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 Remark 2). In general, at any instant of time when the coordinator has received $2n + 1$ messages, the orders of at least $2n$ 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 $S = \{s_1, s_2, \dots\}$ in $L(G)$ such that $s_1, s_2, \dots$ are arbitrarily long*[4]*, 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. $F_i \in s$,

[4]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.

6. and there exists at least $s' \in S$ such that $F_i \notin S$.

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

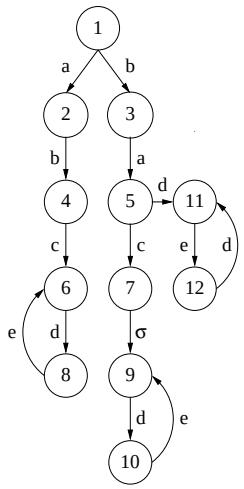

Figure 2: System model for Example 4.1

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.

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 $s_0 u_1 a u_2 b u_3 c$ where $a \in \Sigma_{o1}$, $b \in \Sigma_{o2}$, $c \in \Sigma_{o1}$, 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^e_{d1} after the execution of the the event a and c, respectively, and q_2 be the state of the diagnoser G^e_{d2} after the execution of the event b. Denote by q_{old}, q_{1old}, and q_{2old} the states of the diagnosers G^e_d, G^e_{d1}, and G^e_{d2}, 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 is the true one. Assume without loss of generality that it is xy (corresponding to the observation of ab). From Theorem 3 of [7] 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:

$$C_a = \quad (q_1 \cap^R_e UR_2(q_2)) \cap_c C_b \quad =: C_2 \cap_c C_b, \tag{3}$$

where

$$C_b = \quad (UR_1(q_{old1}) \cap^i_e q_2) \cap_c C_{old} \quad =: C_1 \cap_c C_{old}, \tag{4}$$

and C_{old} is the state of the coordinator following the order $P(s_0)$.

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 $P(t_2) \geq n + 2$[5].

  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,

  $$C_{old} = \delta^e_d(q_0, P(s_0)) = q_{old}. \tag{5}$$

  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 $P(s_0)ba \in P(L)$, applying the update rule for the order $P(s_0)ba$ results in the corresponding state of the centralized extended diagnoser by the results of [7].

  Necessity ($\Rightarrow$). To prove necessity we exploit (3), (4), and the fact that the intersections C_a and C_b

[5] 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.

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 (3). By assumption C_a is non-empty; consequently, C_b and C_1 are non-empty. Hence, by the definition of $\cap_e^i$, C_1 is an extended diagnoser state, and it is of the form

$$C_1 = \{((x_{old1}, l_{old1}), (x_{new1}, l_{new1})), \ldots, ((x_{oldr}, l_{oldr}), (x_{newr}, l_{newr}))\}. \tag{6}$$

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 (4) we obtain

$$C_b = \{((x_{oldn}, l_{oldn}), (x_{newn}, l_{newn})), \ldots, ((x_{oldr'}, l_{oldr'}), (x_{newr'}, l_{newr'}))\}, \tag{7}$$

where $n \geq 1$, $r' \leq r$, and

$$x_{oldi} \in SP(q_{old}), i = n, \ldots, r'. \tag{8}$$

From (4), (7), (8), 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

$$C_b = \delta_d^e(q_{old}, b) = \delta_d^e(q_0, P(s_0)b), \tag{9}$$

by the results of [7].
Since by assumption C_a in (3) is non-empty, C_2 in (3) is non-empty. But

$$C_2 = \{((y_{old1}, k_{old1}), (y_{new1}, k_{new1})), \ldots, ((y_{olds}, k_{olds}), (y_{news}, k_{news}))\}. \tag{10}$$

Therefore, there exist traces in $L(G)$ whose projection with respect to P_1 equals to $P_1(s_0)a$. By (3), (9), and (10)

$$C_a = \{((y_{oldn}, k_{oldn}), (y_{newn}, k_{newn})), \ldots, ((y_{olds'}, k_{olds'}), (y_{news'}, k_{news'}))\} \tag{11}$$

where $n \geq 1$, $s' \leq s$, and

$$y_{oldi} \in SP(C_b), i = n, \ldots, s'. \tag{12}$$

From (7), (9), (11), and (12) 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

$$C_a = \delta_d^e(C_b, a) = \delta_d^e(q_{old}, ba) = \delta_d^e(q_0, P(s_0)ba), \tag{13}$$

by the results of [7]. 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 $s_0 = \epsilon$. Consequently, $C_{old} = q_{old} = (1N, 1N)$ in (5) by definition of the protocol, and the rest of the proof remains as is. **Q.E.D.**

Remark 3 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

$$\begin{aligned} P_1(s_0ab) &= P_1(s_0ba), \\ P_2(s_0ab) &= P_2(s_0ba), \\ |P(s_0ab)| &= |P(s_oba)|, \end{aligned} \tag{14}$$

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

$$\begin{aligned} P_1(s_0abc) &= P_1(s_0acb), \\ P_2(s_0abc) &= P_2(s_0acb), \\ |P(s_0abc)| &= |P(s_oacb)|, \end{aligned} \tag{15}$$

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 3)). 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. **Q.E.D.**

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 [7] 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 **2** of [7] that allows for communication **D**elays). A key feature of Protocol 2 studied in [7] is the following: if the system has no failure ambiguous traces (as defined in [7]; 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. 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 [17].

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 defined in [7]; see Appendix A for details.

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, $\mathbf{SB_i}$, that takes the values $\mathbf{SB_i} = 1$ when $\sigma \in \Sigma_{oj}$, $j \in \{1, 2\}$, $j \neq i$, or $\mathbf{SB_i} = 0$ when $\sigma \notin \Sigma_{oj}$.

5.3 Decision rule

The coordinator's decision rule is composed of 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 [7], 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 defining the information update rule of Protocol 2D, we first recall from [7] the structure of the coordinator. For every possible order that is sorted out the coordinator has five registers, ($R1$, $R2$, $R3$, $R4$, SB), besides the register C that holds its diagnostic information. The five registers are used to store incoming messages from the local sites and previous relevant values necessary for the update of its information. R_1

and R_2 hold the latest states of G_{d1} and G_{d2}, respectively, R_3 and R_4 hold the latest unobservable reaches of G_{d1} and G_{d2}, respectively, and SB specifies whether to apply the information update rule to the available information in the registers ($SB = 0$) or wait for the next incoming message ($SB = 1$). At reset, *R1* and *R2* are initialized with the initial states of G_{d1} and G_{d2}, respectively, while *R3* and *R4* hold the unobservable reaches of the initial state of G_{d1} and G_{d2}, respectively. The register SB is initially set to 0. The information update rule is specified in Table 3.

Last report received from G_{d1}					**Last report received from G_{d2}**				
Rule	*SB*	SB_1	*C*	*New SB*	**Rule**	*SB*	SB_2	*C*	*New SB*
DR1	0	0	$R_1 \cap R_4$	0	**DR4**	0	0	$R_2 \cap R_3$	0
DR2	0	1	Wait	1	**DR5**	0	1	Wait	1
DR3	1	1	$R_1 \cap R_2$	0	**DR6**	1	1	$R_1 \cap R_2$	0

Table 3: Information update rule at the coordinator site (Protocol 2)

The coordinator applies the information update rule of Table 3 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 and status bits. This is possible by definition of the update rule that only requires information from the last update to generate the new one (cf. Table 3). 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 [17]. We note that Protocol 2 in [7] performs as well as the centralized diagnostic scheme of [17] 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 [7] for examples.

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 [17] 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

$$C_1 = UR_1(q_{11}) \cap q_2. \tag{16}$$

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.

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

$$C_2 = UR_2(q_2) \cap q_{11}. \tag{17}$$

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

$$P_1(s'_1) = P_1(s), \tag{18}$$

$$P_2(s'_2) = P_2(s), \tag{19}$$

$$\delta(x_0, s'_1) = \delta(x_0, s'_2). \tag{20}$$

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

$$C_3 = UR_1(q_{12}) \cap q_2. \tag{21}$$

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

$$P_1(t'_1) = P_1(su_3c), \tag{22}$$

$$P_2(t'_2) = P_2(s), \tag{23}$$

$$\delta(x_0, t'_1) = \delta(x_0, t'_2). \tag{24}$$

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

$$P_2(t'_2) = P_2(su_3c). \tag{25}$$

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 [17]. **Q.E.D.**

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 [17] 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. 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} [7], where G_{test2} is a FSM that generates, among other information, the states of the coordinator when global order is preserved. 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

The procedure presented in Section 5.3 assumed that all messages communicated to the coordinator regard only events that are observed by either site 1 or site 2, but not both. In fact, 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; 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 no failure-ambiguous traces (with respect to all failure types), Protocol 2D-C eventually identifies the same failures as the centralized diagnostic scheme of [17].*

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 presented in Table 3, 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 [7], 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. **Q.E.D.**

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. 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 [17] 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 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. If either a, b, or both are common events it is easy to verify that the coordinator can figure out the correct order in which events were executed by the system and consequently applies the correct information update rule. Since s is not failure-ambiguous we have from the results in [7] that the failure F_i is diagnosed. 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

$$C_1 = UR_1(q_1) \cap q_2. \tag{26}$$

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.

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

$$C_2 = UR_2(q_2) \cap q_1. \tag{27}$$

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

$$P_1(s'_1) = P_1(s) \quad (28)$$

$$P_2(s'_2) = P_2(s) \quad (29)$$

$$\delta(x_0, s'_1) = \delta(x_0, s'_2) \quad (30)$$

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 [17]. **Q.E.D.**

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 [17] 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 [17]. 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 [17] 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 [7] which achieves the same diagnostic performance as the diagnostic scheme of [17] 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 [17] and that of Protocol 1 of [7]. (3) The memory required at the coordinator's site to implement the protocol is larger than that required in Protocol 1 of [7].

The key features of Protocol 2D are: (1) it achieves the same performance as the centralized diagnostic scheme of [17] 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 [7]). (2) The delay of its diagnostic decision is higher than the delay of the centralized diagnostic scheme of [17]. (3) The memory required at the coordinator's site to implement the protocol is larger than that required in Protocol 2 of [7]. 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. 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 timestamps were available, Protocols 1 and 2 of [7] 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. For instance, in low energy mobile networks [21] or telecommunication networks [8, 11], it may be infeasible to use timestamps.

In this paper we only considered the "one-step out of order" assumption for reasons of simplicity and compactness. In fact, the same results 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 the results presented here 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 [7] for the case where global order is preserved.

Appendices

A Diagnosers

The diagnoser is a deterministic FSM built from the system model G. Let $\Delta_f = \{F_1, F_2, \ldots, F_m\}$, where $m = |\Pi_f|$, denote the set of failure labels, and let $\Delta = \{N\} \cup 2^{\Delta_f}$. The label N indicates normal (non-faulty)

behavior of the system, while F_i, $i \in \{1, 2, \ldots, m\}$ means that "a failure of type F_i has occurred". Define $Q_o = 2^{X_o \times \Delta}$, where

$$X_o = \{x_0\} \cup \{x \in X : \ x \text{ has an observable event into it}\}. \tag{31}$$

The centralized diagnoser for G is the FSM

$$G_d = (Q_d, \Sigma_o, \delta_d, q_0), \tag{32}$$

where Q_d, Σ_o, δ_d, and q_0 have the usual interpretation of state space, event set, transition function and initial state. The initial state of the diagnoser is defined to be $\{(x_0, \{N\})\}$. The transition function δ_d of the diagnoser is constructed in a manner similar to the transition function of an observer of G (see, e.g., [5]), with an additional aspect that includes attaching failure labels to the states and propagating these labels from state to state (cf. [17]). The state space Q_d is the resulting subset of Q_o composed of the states of the diagnoser that are reachable from q_0 under δ_d. A state q_d of G_d is of the form $q_d = \{(x_1, l_1), \ldots, (x_n, l_n)\}$, where $x_i \in X_o$ and $l_i \in \Delta$. We refer the reader to [17, 18] for a formal presentation of the construction of the diagnoser.

In the context of the coordinated decentralized architecture of Figure 1, the unobservable reach of the state of a diagnoser at site j, $j \in \{1, 2\}$ is defined as follows.

Definition A.1 [7] *Let $q = \{(x_1, l_1), \cdots, (x_n, q_n)\}$ be a state of the diagnoser at site j. Define the set*

$$S_j(q) \ = \ \{s \in (\Sigma \backslash \Sigma_{oj})^* : s \in L_\sigma(G, x_k) \text{ for some } \sigma \in \Sigma_{oi}, i \in \{1, 2\} \backslash \{j\}, \text{ and some } k \in \{1, \ldots, n\}\},$$

where $L_\sigma(G, x)$ denote the set of all traces s that originate from state x in G such that $s = u\sigma$, $u \in \Sigma_{uo}^*$, and $\sigma \in \Sigma_o$. Then the **unobservable reach** *of q with respect to $\Sigma \backslash \Sigma_{oj}$ is defined as follows:*

$$UR_j(q) = \{q\} \cup \bigcup_{s \in S_j(q)} \{(y_s, l_s)\},$$

where (i) y_s is the successor of some x_k, $k \in \{1, \ldots, n\}$, after sub-trace $s \in S_j(q)$, and (ii) l_s is the failure label corresponding to y_s, obtained by propagating the label l_k of x_k according to the label propagation function defined in [17].

By definition the diagnoser state only represents those states that are reached following an observable event; the unobservable reach of a diagnoser state at site j appends to the diagnoser state the states that are reached through unobservable events (to site j) following that observable event up to an event observable by the other site. Also note that while we call $UR_j(q)$ the unobservable reach of q with respect to $\Sigma \backslash \Sigma_{oj}$, its definition stipulates that the sub-traces that are used to generate it end with an event in Σ_{oi}, the other set of observable events.

Examples of the concepts illustrated in this appendix can be found in [17] and [7].

B Extended diagnosers

A variation of the diagnoser is the extended diagnoser. The extended diagnoser for G was first introduced in [14], and it is the FSM

$$G_d^e = (Q_d^e, \Sigma_o, \delta_d^e, q_0^e), \tag{33}$$

where $Q_d^e, \Sigma_o, \delta_d^e$, and q_0^e have the usual interpretation of state space, event set, transition function, and initial state. The initial state of the extended diagnoser is defined to be $\{(x_0, N), (x_0, N)\}$. A state $q \in Q_d^e$ is of the form

$$q = \{((x_1, l_1), (x_1', l_1')), ((x_2, l_2), (x_2', l_2')), ((x_2, l_2), (x_2'', l_2'')), \ldots, ((x_n, l_n), (x_n', l_n'))\},$$

where each (x, l) pair is in Q_o, i.e., $x \in X_o$ and $l \in \Delta$. A tuple of (x, l) pairs, say $((x_1, l_1), (x_1', l_1'))$, has the following meaning: x_1' is a component of a system state estimate after the occurrence of an observable event and l_1' is its failure label, while x_1 is the immediate predecessor state of x_1' in G' and l_1 is its corresponding failure label. G' is the nondeterministic FSM that results from projecting the system G onto Σ_o, the set of observable events. The transition function δ_d^e of the extended diagnoser is constructed in a manner similar to the transition function of the diagnoser G_d, with the additional aspect that every state of G that appears in a state component of G_d is associated with its immediate predecessor state in G' (along the sub-traces of events under consideration) and both states carry their labels; these labels are attained following the same label propagation rules as in [17]. The state space Q_d^e is the resulting subset of $Q_o \times Q_o$ composed of the states of the extended diagnoser that are reachable from q_0^e under δ_d^e. By construction, $L(G_d^e) = L(G_d)$.

We define the state projection $SP : Q_o \times Q_o \to Q_o$ as follows:

$$q = \{((x_1, l_1), (x_1', l_1')), \ldots, ((x_n, l_n), (x_n', l_n'))\} \mapsto SP(q) = \{(x_1', l_1'), \ldots, (x_n', l_n')\}. \tag{34}$$

Then, with a slight abuse of notation, we have that $SP(G_d^e) = G_d$; hence, one diagnoser state may be associated with more than one extended diagnoser states. Therefore, an extended diagnoser state potentially carries more information than a diagnoser state.

The following two definitions introduce two operators on the states of the extended diagnosers.

Definition B.1 [7] *Let* $q_1 = \{((x_1, l_1), (x_1', l_1')), \ldots, ((x_n, l_n), (x_n', l_n'))\}$ *and* $q_2 = \{((y_1, l_1), (y_1', l_1')), \ldots ((y_m, l_m), (y_m', l_m'))\}$ *belong to* $Q_o \times Q_o$. *We denote by* $\cap_e^i$, $i \in \{L, R\}$ *the intersection scheme that acts on* q_1 *and* q_2, *and we define it as follows:*

$$q_1 \cap_e^i q_2 \stackrel{\triangle}{=} \{((z, l), (z', l')) \in Q_o \times Q_o\text{: } (z', l') = (x_i', l_i') = (y_j', l_j') \text{ for some } i, j,\ i \in \{1, 2, \ldots, n\},\ j \in \{1, 2, \ldots, m\}, \text{ and } (z, l) = (x_i, l_i) \text{ if } i{=}L, \text{ otherwise } (z, l) = (y_j, l_j)\}.$$

This intersection scheme is a regular intersection of the components of the two system state estimates along with their failure labels. However, the intersection applies to the components corresponding to the current system state estimates and not to their immediate predecessors. The components of $q_1 \cap_e^i q_2$ corresponding to the immediate predecessors are determined by operator i.

Definition B.2 [7] *Let* $q_1 = \{((x_1, l_1), (x_1', l_1')), \ldots, ((x_n, l_n), (x_n', l_n'))\}$ *and* $q_0 = \{((y_1, l_1), (y_1', l_1')), \ldots, ((y_m, l_m), (y_m', l_m'))\}$ *belong to* $Q_o \times Q_o$. *We denote by* $\cap_c$ *the intersection scheme that acts on* q_1 *and* q_0, *and we define it as follows:*

$$q_1 \cap_c q_0 \stackrel{\triangle}{=} \{((z, l), (z', l')) \in Q_o \times Q_o : (z, l) = (x_i, l_i) = (y_j', l_j'), \text{ for some } i, j,\ i \in \{1, 2, \ldots, n\},\ j \in \{1, 2, \ldots, m\}, \text{ and } (z', l') = (x_i', l_i')\}.$$

In the context of the coordinated decentralized architecture of Figure 1, the unobservable reach of the state of an extended diagnoser at site j, $j \in \{1, 2\}$ is defined as follows.

Definition B.3 [7] *Let* $q = \{((x_1, l_1), (x'_1, l'_1)), \ldots, ((x_n, l_n), (x'_n, l'_n))\}$ *be a state of the extended diagnoser at site* j, $j \in \{1, 2\}$. *Let* $L_\sigma(G, x)$ *denote the set of all traces* s *that originate from state* x *in* G *such that* $s = u\sigma$, $u \in \Sigma^*_{uo}$, *and* $\sigma \in \Sigma_o$. *Define the set*

$$S_j(q) \;=\; \{s \in (\Sigma \backslash \Sigma_{oj})^* : s \in L_\sigma(G, x'_k) \text{ for some } \sigma \in \Sigma_{oi}, i \in \{1, 2\} \backslash \{j\}, \text{ and some } k \in \{1, \ldots, n\}\}.$$

Then the **unobservable reach** *of* q *with respect to* $\Sigma \backslash \Sigma_{oj}$ *is defined as follows:*

$$UR_j(q) = \{q\} \cup \bigcup_{s \in S_j(q)} \{((y_s, l_s), (y'_s, l'_s))\}$$

where (i) y'_s *is the successor of some* x'_k, $k \in \{1, \ldots, n\}$, *after sub-traces* $s \in S_j(q)$, *(ii)* y_s *is the immediate predecessor along* s *of* y'_s *in* G'_1[6], *and (iii)* l_s, l'_s *are the failure labels corresponding to* y_s, y'_s, *obtained by propagating the label* l'_k *of* x'_k *according to the label propagation function defined in [17].*

The unobservable reach appends to the components of each state of the extended diagnoser at site j some additional components (along with failure labels and predecessors) that may have been reached following an additional event or a sequence of events that are not observable by the local site j. Note here that in the above definition, y_s may not be equal to x'_k. Also note that while we call $UR_j(q)$ the unobservable reach of q with respect to $\Sigma \backslash \Sigma_{oj}$, its definition stipulates that the sub-traces that are used to generate it end with an event in Σ_{oi}, the other set of observable events.

Examples of the concepts illustrated in this appendix can be found in [17] and [7].

References

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

[2] R. J. Aumann. Agreeing to disagree. *The Annals of Statistics*, 4(6):1236–1239, 1976.

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

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

[5] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, MA, 1999.

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

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

[6] G'_j is the nondeterministic FSM that results from projecting the system G onto Σ_{oj}.

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

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

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

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

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

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

[14] M. Sampath. The extended diagnoser, November 1993. Unpublished memorandum.

[15] M. Sampath. *A Discrete Event Systems Approach to Failure Diagnosis.* PhD thesis, The University of Michigan, 1995. Available at `http://www.eecs.umich.edu/umdes`.

[16] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Trans. Automat. Contr.*, 43(7):908–929, July 1998.

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

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

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

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

[21] W. Stark, Project Director. Low energy electronics design for mobile platforms. Project Report of ARO-MURI for the period 9/96 to 7/99, Electrical Engineering and Computer Science Department, The University of Michigan, 1999.

[22] R. B. Washburn and D. Teneketzis. Asymptotic agreement among communicating decision makers. *Stochastics*, 13(1–2):103–129, 1984.

[23] A. Willsky. A survey of design methods for failure detection in dynamic systems. *Automatica*, 12:601–611, 1976.