The Fault Detection Problem

Andreas Haeberlen* Petr Kuznetsov†

Abstract

One of the most important challenges in distributed computing is ensuring that services are correct and available despite faults. Recently it has been argued that fault detection can be factored out from computation, and that a generic fault detection service can be a useful abstraction for building distributed systems. However, while fault detection has been extensively studied for crash faults, little is known about detecting more general kinds of faults.

This paper explores the power and the inherent costs of generic fault detection in a distributed system. We propose a formal framework that allows us to partition the set of all faults that can possibly occur in a distributed computation into several *fault classes*. Then we formulate the *fault detection problem* for a given fault class, and we show that this problem can be solved for only two specific fault classes, namely *omission faults* and *commission faults*. Finally, we derive tight lower bounds on the cost of solving the problem for these two classes in asynchronous message-passing systems.

Keywords: Fault classes, fault detection problem, message complexity, lower bounds

1 Introduction

Handling faults is a key challenge in building reliable distributed systems. There are two main approaches to this problem: *Fault masking* aims to hide the symptoms of a limited number of faults, so that users can be provided with correct service in the presence of faults [4, 14], whereas *fault detection* aims at identifying the faulty components, so that they can be isolated and repaired [7, 10]. These approaches are largely complementary. In this paper, we focus on fault detection.

Fault detection has been extensively studied in the context of "benign" crash faults, where it is assumed that a faulty component simply stops taking steps of its algorithm [5, 6]. However, this assumption does not always hold in practice; in fact, recent studies have shown that general faults (also known as Byzantine faults [15]) can have a considerable impact on practical systems [17]. Thus, it would be useful to apply fault detection to a wider class of faults. So far, very little is known about this topic; there is a paper by Kihlstrom et al. [12] that discusses Byzantine fault detectors for consensus and broadcast protocols, and there are several algorithms for detecting certain types of non-crash faults, such as PeerReview [10] and SUNDR [16]. However, many open questions remain; for example, we still lack a formal characterization of the types of non-crash faults that can be detected in general, and nothing is known about inherent costs of detection.

This paper is a first step towards a better understanding of general fault detection. We propose a formal model that allows us to formulate the *fault detection problem* for arbitrary faults, including non-crash faults. We introduce the notion of a *fault class* that captures a set of *faults*, i.e., deviations of system components from their expected behavior. Solving the fault detection problem for a fault class F means finding a transformation τ_F that, given any algorithm A, constructs an algorithm $\bar{A}$ (called an *extension* of A) that

*Max Planck Institute for Software Systems, Campus E1.4, 66123 Saarbrücken, Germany, ahae@mpi-sws.org
†TU Berlin/Deutsche Telekom Laboratories, TEL 4, Ernst-Reuter-Platz 7, 10587 Berlin, Germany, pkuznets@acm.org

works exactly like A but does some additional work to identify and expose faulty nodes. Whenever a fault instance from the class F appears, $\bar{A}$ must expose at least one faulty suspect (completeness), it must not expose any correct nodes infinitely long (accuracy), and, optionally, it may ensure that all correct nodes expose the same faulty suspects (agreement).

Though quite weak, our definition of the fault detection problem still allows us to answer two specific questions: Which faults can be detected, and how much extra work from does fault detection require from the extension? To answer the first question, we show that the set of all fault instances can be divided into four non-overlapping classes, and that the fault detection problem can be solved for exactly two of them, which we call *commission faults* and *omission faults*. Intuitively, a commission fault exists when a node sends messages a correct node would not send, whereas an omission fault exists when a node does *not* send messages a correct node *would* send.

To answer the second question, we study the *message complexity* of the fault detection problem, that is, the ratio between the number of messages sent by the most efficient extension and the number of messages sent by the original algorithm. We derive tight lower bounds on the message complexity for commission and omission faults, with and without agreement. Our results show that a) the message complexity for omission faults is higher than that for commission faults, and that b) the message complexity is (optimally) linear in the number of nodes in the system, except when agreement is required for omission faults, in which case it is quadratic in the number of nodes.

In summary, this paper makes the following four contributions: (1) a formal model of a distributed system in which various kinds of faults can be selectively analyzed, (2) a statement of the fault detection problem for arbitrary faults, (3) a complete classification of all possible faults, including a precise characterization of the set of faults for which the fault detection problem can be solved, and (4) tight lower bounds on the message complexity of the fault detection problem. Viewed collectively, our results constitute a first step toward understanding the power and the inherent costs of fault detection in a distributed system.

The rest of this paper is organized as follows: We begin by introducing our system model in Section 2 and then formally state the fault detection problem in Section 3. In Section 4, we present our classification of faults, and we show for which classes the fault detection problem can be solved. In Section 5, we derive tight bounds on the message complexity, and we conclude by discussing related work in Section 6 and future work in Section 7.

2 Preliminaries

2.1 System model

Let N be a set of *nodes*. Each node has a terminal[1] and a network interface. It can communicate with the other nodes by sending and receiving messages over the network, and it can send outputs to, and receive inputs from, its local terminal. We assume that processing times are negligible; when a node receives an input, it can produce a response immediately.

Each message m has a unique *source* $src(m) \in N$ and a unique *destination* $dest(m) \in N$. We assume that messages are authenticated; that is, each node i can initially create only messages m with $src(m) = i$, although it can delegate this capability to other nodes (e.g., by revealing its key material). Nodes can also forward messages to other nodes and include messages in other messages they send, and we assume that a forwarded or included message can still be authenticated.

A computation unfolds in discrete *events*. An event is a tuple (i, I, O), where $i \in N$ is a node on which the event occurs, I is a set of inputs (terminal inputs or messages) that i receives in the event, and O is a set of outputs (terminal outputs or messages) that i produces in the event. An *execution* e is a sequence

[1] Instead of an actual terminal, nodes may have any other local I/O interface that cannot be observed remotely.

of events $(i_1, I_1, O_1), (i_2, I_2, O_2), \ldots$. We write $e|_S$ for the subsequence of e that contains the events with $i_k \in S$; for $i \in N$, we abbreviate $e_{\{i\}}$ as $e|_i$. When a finite execution e is a prefix of another execution e', we write $e \lfloor e'$. Finally, we write $|e|$ to denote the number of unique messages that are sent in e.

A system is modeled as a set of executions. In this paper, we assume that the network is reliable, that is, a) a message is only received if it has previously been sent at least once, and b) a message that is sent is eventually received at least once. Formally, we assume that, for every execution e of the system and every message m:

$$m \in I_k \Rightarrow [i_k = \mathit{dest}(m) \wedge \exists l < k : (i_l = \mathit{src}(m) \wedge m \in O_l)]$$

$$(m \in O_k \wedge \mathit{src}(m) = i_k) \Rightarrow [\exists l : i_l = \mathit{dest}(m) \wedge m \in I_l]$$

An *open execution* is an execution for which only the first condition holds. Thus, an open execution may contain some messages that are sent, but not received. This definition is needed later in the paper; an actual execution of the system is never open. Finally, we introduce the following notation for brevity:

- $\textsc{recv}(i, m) \in e$ iff m is a message with $i = \mathit{dest}(m)$ and $(i, I, O) \in e$ with $m \in I$.
- $\textsc{send}(i, m, j) \in e$ iff m is a message with $j = \mathit{dest}(m)$ and $(i, I, O) \in e$ with $m \in O$.
- $\textsc{in}(i, t) \in e$ if t is a terminal input and $(i, I, O) \in e$ with $t \in I$.
- $\textsc{out}(i, t) \in e$ if t is a terminal output and $(i, I, O) \in e$ with $t \in O$.

Table 2 (in the appendix) contains an overview of the notation we use in this paper.

2.2 Algorithms and correctness

Each node i is assigned an *algorithm* $A_i = (M_i, TI_i, TO_i, \Sigma_i, \sigma_0^i, \alpha_i)$, where M_i is the set of messages i can send or receive, TI_i is a set of terminal inputs i can receive, TO_i is a set of terminal outputs i can produce, Σ_i is a set of states, $\sigma_0^i \in \Sigma_i$ is the initial state, and $\alpha_i : \Sigma_i \times P(M_i \cup TI_i) \to \Sigma_i \times P(M_i \cup TO_i)$ maps a set of inputs and the current state to a set of outputs and the new state. Here, $P(X)$ denotes the power set of X. For convenience, we define $\alpha(\sigma, \emptyset) := (\sigma, \emptyset)$ for all $\sigma \in \Sigma_i$.

We make the following four assumptions about any algorithm A_i: a) it only sends messages that can be properly authenticated, b) it never sends the same message twice, c) it discards incoming duplicates and any messages that cannot be authenticated, and d) it never delegates the ability to send messages m with $\mathit{src}(m) = i$, e.g., by revealing or leaking i's key material. Note that assumption b) does not affect generality, since A_i can simply include a nonce with each message it sends. We also assume that it is possible to decide whether A_i, starting from some state σ_x, could receive some set of messages X in any order (plus an arbitrary number of terminal inputs) without sending any messages. This trivially holds if $|\Sigma_i| < \infty$.

We say that a node i is *correct* in execution $e|_i = (i, I_1, O_1), (i, I_2, O_2), \ldots$ with respect to an algorithm A_i iff there is a sequence of states $\sigma_0, \sigma_1, \ldots$ in Σ_i such that $\sigma_0 = \sigma_0^i$ and, for all $k \geq 1$, $\alpha_i(\sigma_{k-1}, I_k) = (\sigma_k, O_k)$. Note that correctness of a node i implies that the node is *live*: if i is in a state σ_{k-1} and receives an input I, then i must produce an output O_k such that $\alpha_i(\sigma_{k-1}, I_k) = (\sigma_k, O_k)$. If i is not correct in $e|_i$ with respect to A_i, we say that i is *faulty* in $e|_i$ with respect to A_i.

A *distributed algorithm* is a tuple $(A_1, \ldots, A_{|N|})$, one algorithm per node, such that $M_i = M_j$ for all i, j. When we say that an execution e is an execution of a distributed algorithm A, this implies that each node i is considered correct or faulty in e with respect to the algorithm A_i it has been assigned. We write $\mathit{corr}(A, e)$ to denote the set of nodes that are correct in e with respect to A.

2.3 Extensions

$(\bar{A}, A, \mu_m, \mu_s, XO)$ is called a *reduction* of an algorithm $\bar{A} = (\bar{M}, \bar{TI}, \bar{TO}, \bar{\Sigma}, \bar{\sigma}_0, \bar{\alpha})$ to an algorithm $A = (M, TI, TO, \Sigma, \sigma_0, \alpha)$ iff μ_m is a total map $\bar{M} \mapsto P(M)$, μ_s is a total map $\bar{\Sigma} \mapsto \Sigma$, and the following conditions hold:

X1 $\bar{TI} = TI$, that is, A accepts the same terminal inputs as $\bar{A}$;

X2 $\bar{TO} = TO \cup XO$ and $TO \cap XO = \emptyset$, that is, A produces the same terminal outputs as $\bar{A}$, except XO;

X3 $\mu_s(\bar{\sigma}_0) = \sigma_0$, that is, the initial state of $\bar{A}$ maps to the initial state of A;

X4 $\forall m \in M\ \exists \bar{m} \in \bar{M}\colon \mu_m(\bar{m}) = m$, that is, every message of A has at least one counterpart in $\bar{A}$;

X5 $\forall \sigma \in \Sigma\ \exists \bar{\sigma} \in \bar{\Sigma}\colon \mu_s(\bar{\sigma}) = \sigma$, that is, every state of A has at least one counterpart in $\bar{\Sigma}$;

X6 $\forall \bar{\sigma}_1, \bar{\sigma}_2 \in \bar{\Sigma}, \bar{mi}, \bar{mo} \subseteq \bar{M}, ti \subseteq TI, to \subseteq TO : [\bar{\alpha}(\bar{\sigma}_1, \bar{mi} \cup ti) = (\bar{\sigma}_2, \bar{mo} \cup to)] \Rightarrow [\alpha(\mu_s(\bar{\sigma_1}), \mu_m(\bar{mi}) \cup ti) = (\mu_s(\bar{\sigma_2}), \mu_m(\bar{mo}) \cup (to \setminus XO))]$, that is, there is a homomorphism between $\bar{\alpha}$ and α.

If there exists at least one reduction from an algorithm $\bar{A}$ to an algorithm A, we say that $\bar{A}$ is an *extension* of A. For every reduction $(\bar{A}, A, \mu_m, \mu_s, XO)$ we can construct an *execution mapping* μ_e that maps executions of $\bar{A}$ to (possibly open) executions of A as follows:

1. Start with $e = \emptyset$.
2. For each new event $(i, \bar{I}, \bar{O})$, perform the following steps:
   (a) Compute $I := (\bar{I} \cap TI_i) \cup \mu_m(\bar{I} \cap \bar{M})$ and $O := (\bar{O} \cap TO_i) \cup \mu_m(\bar{O} \cap \bar{M})$.
   (b) Remove from I any $m \in M$ with *dest*$(m) \neq i$ or RECV$(i, m) \in e$.
   (c) Remove from O any $m \in M$ with SEND$(i, m, j) \in e$.
   (d) For each node $j \in N$, compute $O_j := \{m \in O \mid \mathit{src}(m) = j\}$.
   (e) If $I \neq \emptyset$ or $O_i \neq \emptyset$, append (i, I, O_i) to e.
   (f) For each $j \neq i$ with $O_j \neq \emptyset$, append $(j, \emptyset, O_j)$ to e.

A simple example of a reduction is the identity $(A, A, id, id, \emptyset)$. Note that there is a syntactic correspondence between an extension and its original algorithm, not just a semantic one. In other words, the extension not only solves the same problem as the original algorithm (by producing the same terminal outputs as the original), it also solves it in the same way (by sending the same messages in the same order). Recall that our goal is to detect whether or not the nodes in the system are following a given algorithm; we are *not* trying to find a better algorithm. Next, we state a few simple lemmas about extensions.

Lemma 1 *Let $\bar{A}$ and A be two algorithms for which a reduction $(\bar{A}, A, \mu_m, \mu_s, XO)$ exists. Then, if $\bar{e}$ is an execution in which a node i is correct with respect to $\bar{A}$, i is correct in $\mu_e(\bar{e})$ with respect to A.*

Proof sketch: If i is correct in $\bar{e}|_i$ with respect to $\bar{A}$, there exists a matching sequence of states $\bar{\sigma}_1, \bar{\sigma}_2, \ldots$ of $\bar{A}_i$ in $\bar{e}|_i$, and we can use homomorphism X6 to convert it into a sequence of states $\sigma_1, \sigma_2, \ldots$ of A_i in $\mu_e(\bar{e})|_i$. The proof is by induction over the length of $\bar{e}|_i$. For the full proof, see Appendix A.1. □

Note that, if a node i is correct in $\bar{e}$ with respect to $\bar{A}$, then it must be correct in $\mu_e(\bar{e})$ with respect to A, but the reverse is not true. In other words, it is possible for a node i to be faulty in $\bar{e}$ with respect to $\bar{A}$ but still be correct in $\mu_e(\bar{e})$ with respect to A.

Lemma 2 *Let $\bar{A}$ and A be two algorithms for which a reduction $(\bar{A}$,A,μ_m,μ_s,$XO)$ exists, let $\bar{e}_1$ be an execution of $\bar{A}$, and let $\bar{e}_2$ be a prefix of $\bar{e}_1$. Then $\mu_e(\bar{e}_2)$ is a prefix of $\mu_e(\bar{e}_1)$.*

Proof: Follows from the way $\mu_e(\bar{e}_1)$ is constructed (events are always appended, never removed). □

Lemma 3 *Let $\bar{A}$ and A be two algorithms for which a reduction $(\bar{A}$,A,μ_m,μ_s,$XO)$ exists, and let e be an execution of A. Then there exists an execution $\bar{e}$ of $\bar{A}$ such that a) $\mu_e(\bar{e}) = e$ (modulo duplicate messages sent by faulty nodes in e), and b) a node i is correct in $\bar{e}$ with respect to $\bar{A}$ iff it is correct in e with respect to A.*

Proof sketch: Given an execution e of A, we can construct a matching execution $\bar{e}$ of $\bar{A}$ essentially by feeding the same inputs to the correct nodes, and by delivering messages in the same order. This works because of homomorphism X6. The only two complications are that a) we must ensure that messages are eventually delivered in $\bar{e}$ even if they have no direct equivalent (via μ_m) in e, and b) we must choose the behavior of the faulty nodes appropriately. For the full proof, see Appendix A.2 □

2.4 Facts and evidence

To detect faults, and to identify faulty nodes, the correct nodes must collect information about the current execution. Clearly, no correct node can expect to know the entire execution at any point, since it cannot observe events on other nodes. However, each node can locally observe its inputs and outputs, and each input or output rules out some possible executions that *cannot* be the current execution. For example, if a node i receives a message m, this rules out all executions in which m was never sent. If i manages to rule out all executions in which some set S of nodes is correct, it has established that at least one node $s \in S$ must be faulty. Thus, we can use sets of plausible executions to represent a node's knowledge about the current execution.

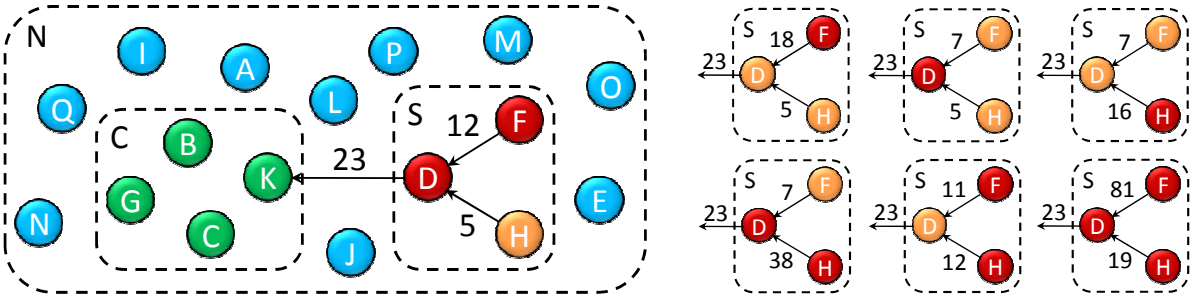
Formally, we define a *fact* ζ to be a set of executions, and we say that a node i *knows* a fact ζ at the end of an execution prefix e iff ζ contains all infinite executions e' where $e|_i$ is a prefix of $e'|_i$ (in other words, e' is consistent with all the inputs and outputs i has seen in e). If a node knows two facts ζ_1 and ζ_2, it can combine them into a new fact $\zeta_3 := \zeta_1 \cap \zeta_2$. If the system is running an extension $\bar{A}$ of an algorithm A, we can map any fact $\bar{\zeta}$ about the current execution $\bar{e}$ of $\bar{A}$ to a fact $\zeta := \{\mu_e(x) \mid x \in \bar{\zeta}\}$ about $\mu_e(\bar{e})$.

Different nodes may know different facts. Hence, the nodes may only be able to detect a fault if they exchange information. However, faulty nodes can lie, so a correct node can safely accept a fact from another node only if it receives *evidence* of that fact. Formally, we say that a message m is evidence of a fact ζ iff for any execution $\bar{e}$ of $\bar{A}$ in which any node receives m, $\mu(\bar{e}) \in \zeta$. Intuitively, evidence consists of signed messages. For more details, please see Section 4.

2.5 Fault instances and fault classes

Not all faults can be detected, and some extensions can detect more faults than others. To quantify this, we introduce an abstraction for an individual 'fault'. A *fault instance* ψ is a four-tuple (A, C, S, e), where A is a distributed algorithm, C and S are sets of nodes, and e is an infinite execution, such that a) C and S do not overlap, b) every $c \in C$ is correct in e with respect to A, and c) at least one node $i \in S$ is faulty in e with respect to A. A *fault class* F is a set of fault instances, and the nodes in S are called *suspects*.

Intuitively, the goal is for the correct nodes in C to identify at least one faulty suspect from S. Of course, an ideal solution would simply identify *all* the nodes that are faulty in e with respect to A; however, this is not always possible. Consider the scenario in Figure 1. In this scenario, the nodes in C know that at least one of the nodes in S must be faulty, but they do not know which ones, or how many. Thus, the size of the set S effectively represents the precision with which the fault can be localized. The best case is $|S| = 1$; this indicates that the fault can be traced to exactly one node. The worst case is $S = N \setminus C$; this indicates that the nodes in C know that a fault exists somewhere, but they are unable to localize it.



(a) Actual execution

(b) Alternative explanations for K's observation

Figure 1: Example scenario. Nodes F and H are supposed to each send a number between 1 and 10 to D, who is supposed to add the numbers and send the result to K. If K receives 23, it knows that at least one of the nodes in $S = \{D, F, H\}$ must be faulty, but it does not know which ones, or how many.

2.6 Environments

Our formulation of the fault detection problem does not require a bound on the number of faulty nodes. However, if such a bound is known, it is possible to find solutions with a lower message complexity. To formalize this, we use the notion of an *environment*, which is a restriction on the fault patterns that may occur in a system. In this paper, we specifically consider environments E_f, in which the total number of faulty nodes is limited to f. If a system in environment E_f is assigned a distributed algorithm A, the only executions that can occur are those in which at most f nodes are faulty with respect to A.

3 The fault detection problem

Let $\nu := \{\text{FAULTY}(X) \mid X \subseteq N\}$ be a set of *fault notifications*. Then the *fault detection problem* for a fault class F is to find a transformation τ_F that maps any distributed algorithm A to an extension $\bar{A} := \tau_F(A)$ such that $\bar{TO} = TO \cup \nu$ and the following conditions hold:

C1 **Nontriviality:** If $\bar{e}$ is an infinite execution of $\bar{A}$ and $i \in N$ is correct in $\bar{e}$ with respect to $\bar{A}$, then i outputs infinitely many fault notifications in $\bar{e}$.

C2 **Completeness:** If (A, C, S, e) is a fault instance in F, $\bar{e}$ is an infinite execution such that $\mu_e(\bar{e}) = e$, and each node $c \in C$ is correct in $\bar{e}$ with respect to $\bar{A}$, then there exists a correct node $c' \in N$ and a node $j \in S$ such that eventually all fault notifications output by c' contain j.

C3 **Accuracy:** If $\bar{e}$ is an infinite execution of $\bar{A}$ and $c_1, c_2 \in N$ are any two nodes that are correct in $\bar{e}$ with respect to $\bar{A}$, then c_1 outputs infinitely many fault notifications that do not include c_2.

We also consider the *fault detection problem with agreement*, which additionally requires:

C4 **Agreement:** If $c_1 \in N$ and $c_2 \in N$ are correct in an execution $\bar{e}$ with respect to $\bar{A}$ and there exists a node $i \in N$ such that eventually all fault notifications output by c_1 in $\bar{e}$ include some node $i \in N$, then eventually all fault notifications output by c_2 in $\bar{e}$ include i as well.

Note that condition C2 does not require us to detect nodes that are faulty in $\bar{e}$ with respect to $\bar{A}$, but correct in $\mu_e(\bar{e})$ with respect to A. Thus, we avoid the infinite recursion that would result from trying to detect faults in the detector itself. Note also that condition C3 is weaker than the definition of eventual strong accuracy in [6], which requires that correct nodes eventually output only faulty nodes. This change is necessary to make the problem solvable in an asynchronous environment.

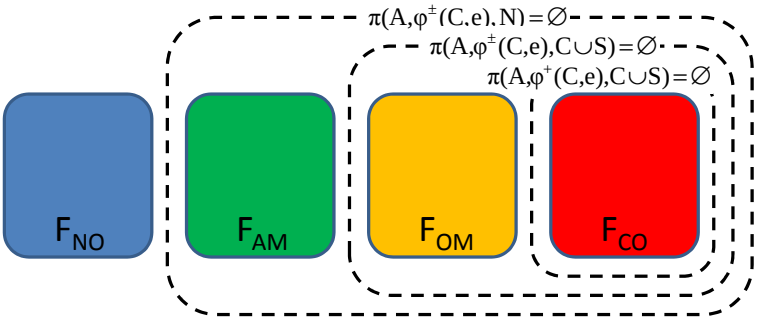


Figure 2: Classification of all fault instances. The fault detection problem cannot be solved for fault instances in F_{NO} (Theorem 2) or F_{AM} (Theorem 3), but solutions exist for F_{OM} and F_{CO} (Theorem 4).

4 Which faults can be detected?

In the rest of this paper, we assume that the only facts for which evidence can exist are a) message transmissions, and b) message receptions. Specifically, a properly authenticated message $\bar{m}$ with $\mu_m(\bar{m}) = m$ and $\mathit{src}(m) = i$ in an execution $\bar{e}$ is evidence of a fact $\{e \mid \textsc{send}(i, m, \mathit{dest}(m)) \in e\}$ about $\mu_e(\bar{e})$, and a properly authenticated message $\bar{m}'$ with $\mathit{src}(\bar{m}') = i$, $m \in \bar{m}'$, and $\mathit{dest}(m) = i$ in an execution $\bar{e}$ is evidence of a fact $\{e \mid \textsc{recv}(i, m) \in e\}$ about $\mu_e(\bar{e})$. Note that in some systems it may be possible to construct evidence of additional facts (e.g., when the system has more synchrony or access to more sophisticated cryptographic primitives). In such systems, the following results may not apply.

4.1 Definitions

We define two *fact maps* ϕ^+ and ϕ^- as follows. Let e be an infinite execution or an execution prefix, and let C be a set of nodes. Then $\phi^+(C, e)$ is the intersection[2] of all facts ζ for which at least one node in C can construct evidence in e (note that there is usually no single node that can construct evidence of *all* facts), and $\phi^-(C, e)$ is the intersection of all facts ζ such that, if the complement $\bar{\zeta}$ were a fact in e (i.e., $e \in \zeta$), then at least one node in C could construct evidence of $\bar{\zeta}$ in e, but $\bar{\zeta} \notin \phi^+(C, e)$. For brevity, we write $\phi^\pm(C, e)$ to represent both kinds of facts, that is, $\phi^\pm(C, e) := \phi^+(C, e) \cap \phi^-(C, e)$.

Intuitively, $\phi^\pm$ represents the sum of all knowledge the nodes in C can have in e if they exchange all of their evidence with each other. Since we have restricted the admissible evidence to messages earlier, $\phi^+(C, e)$ effectively represents knowledge about all the messages sent or received in e by the nodes in C, while $\phi^-(C, e)$ effectively represents knowledge about all the messages *not* sent or received in e by the nodes in C.

We also define the *plausibility map* π as follows. Let A be a distributed algorithm, Z a set of facts, and C a set of nodes. Then $\pi(A, Z, C)$ represents all infinite executions $e \in Z$ in which each node $c \in C$ is correct in e with respect to A. Intuitively, $\pi(A, Z, C)$ is the set of executions of A that are plausible given the facts in Z, and given that (at least) the nodes in C are correct.

A few simple properties of ϕ and π are: 1) $C_1 \subseteq C_2 \Rightarrow \phi(C_2, e) \subseteq \phi(C_1, e)$, that is, adding evidence from more nodes cannot reduce the overall knowledge; 2) $p_1 \lfloor p_2 \Rightarrow \phi(C, p_2) \subseteq \phi(C, p_1)$, that is, knowledge can only increase during an execution; 3) $C_1 \subseteq C_2 \Rightarrow \pi(A, Z, C_2) \subseteq \pi(A, Z, C_1)$, that is, assuming that more nodes are correct can only reduce the number of plausible executions; and 4) $Z_1 \subseteq Z_2 \Rightarrow \pi(A, Z_1, C) \subseteq \pi(A, Z_2, C)$, that is, learning more facts can only reduce the number of plausible executions.

[2]Recall that facts are combined by forming the intersection. Since facts are sets of plausible executions, an execution that is plausible given two facts ζ_1 and ζ_2 must be a member of $\zeta_1 \cap \zeta_2$.

4.2 Fault classes

We define the following fault classes (see also Figure 2):

$$\begin{aligned}
F_{NO} &:= \{(A,C,S,e) \mid \pi(A,\phi^{\pm}(C,e),N) \neq \emptyset\} \\
F_{AM} &:= \{(A,C,S,e) \mid \pi(A,\phi^{\pm}(C,e),N) = \emptyset \wedge \pi(A,\phi^{\pm}(C,e),C \cup S) \neq \emptyset\} \\
F_{OM} &:= \{(A,C,S,e) \mid \pi(A,\phi^{\pm}(C,e),C \cup S) = \emptyset \wedge \pi(A,\phi^{+}(C,e),C \cup S) \neq \emptyset\} \\
F_{CO} &:= \{(A,C,S,e) \mid \pi(A,\phi^{+}(C,e),C \cup S) = \emptyset\}
\end{aligned}$$

F_{NO} is the class of *non-observable faults*. For executions in this class, the nodes in C cannot even be sure that the system contains any faulty nodes, since there exists a correct execution of the entire system that is consistent with everything they see. We will show in Section 4.3 that the fault detection problem cannot be solved for faults in this class.

F_{AM} is the class of *ambiguous fault instances*. When a fault instance is in this class, the nodes in C know that a faulty node exists, but they cannot be sure that it is one of the nodes in S. We will show in Section 4.4 that the fault detection problem cannot be solved for fault instances in this class. Note that the problem here is not that the faults cannot be observed from C, but that the set S is too small. If S is sufficiently extended (e.g., to $N \setminus C$), these fault instances become solvable.

F_{OM} is the class of *omission faults*. For executions in this class, the nodes in C could infer that one of the nodes in S is faulty if they knew all the facts, but the positive facts alone are not sufficient; that is, they would also have to know that some message was *not* sent or *not* received. Intuitively, this occurs when the nodes in S refuse to send some message they are required to send.

F_{CO} is the class of *commission faults*. For executions in this class, the nodes in C can infer that one of the nodes in S is faulty using only positive facts. Intuitively, this occurs when the nodes in S send some combination of messages they would never send in any correct execution.

Theorem 1 *$(F_{NO}, F_{AM}, F_{OM}, F_{CO})$ is a partition of the set of all fault instances.*

Proof: First, we show that no fault instance can belong to more than one class. Suppose $\psi := (A,C,S,e) \in F_{NO}$; that is, there is a plausible correct execution e' of the entire system. Then ψ can obviously not be in F_{AM}, since $\pi(A,\phi^{\pm}(C,e),N)$ cannot be both empty and non-empty. Since all nodes are correct in e', the nodes in $C \cup S$ in particular are also correct, so $\psi \notin F_{OM}$ (Section 4.1, Property 3), and they are still correct if negative facts are ignored, so $\psi \notin F_{CO}$. Now suppose $\psi \in F_{AM}$. Obviously, ψ cannot be in F_{OM}, since $\pi(A,\phi^{\pm}(C,e),C \cup S)$ cannot be both empty and non-empty. But ψ cannot be in F_{CO} either, since using fewer facts can only increase the number of plausible executions (Section 4.1, Property 1). Finally, observe that ψ cannot be in both F_{OM} and F_{CO}, since $\pi(A,\phi^{+}(C,e),C \cup S)$ cannot be both empty and non-empty.

It remains to be shown that any fault instance belongs to at least one of the four classes. Suppose there is a fault instance $\psi \notin (F_{NO} \cup F_{AM} \cup F_{OM} \cup F_{CO})$. Since ψ is not in F_{NO}, we know that $\pi(A,\phi^{\pm}(C,e),N) = \emptyset$. But if this is true and ψ is not in F_{AM}, it follows that $\pi(A,\phi^{\pm}(C,e),C \cup S) = \emptyset$. Given this and that ψ is not in F_{OM}, we can conclude that $\pi(A,\phi^{+}(C,e),C \cup S) = \emptyset$. But then ψ would be in F_{CO}, which is a contradiction. □

4.3 Non-observable faults

Theorem 2 *The fault detection problem cannot be solved for any fault class F with $F \cap F_{NO} \neq \emptyset$.*

Proof sketch: The proof works by showing that, for any fault instance $\psi := (A,C,S,e) \in F_{NO}$, we can construct two executions $\bar{e}_{good}$ and $\bar{e}_{bad}$ of $\bar{A} := \tau(A)$ such that a) all the nodes are correct in $\bar{e}_{good}$, b) the fault occurs in $\bar{e}_{bad}$, and c) the two executions are indistinguishable from the perspective of the nodes

in C (that is, $\bar{e}_{good}|_C = \bar{e}_{bad}|_C$). Hence, the nodes in C would have to both expose some node in S (to achieve completeness in $\bar{e}_{bad}$) and *not* expose any node in S (to achieve accuracy in $\bar{e}_{good}$) based on the same information, which is impossible. For the full proof, see Appendix A.3. □

4.4 Ambiguous faults

Theorem 3 *The fault detection problem cannot be solved for any fault class F with $F \cap F_{AM} \neq \emptyset$.*

Proof sketch: The proof is largely analogous to that of Theorem 2, except that we now construct two executions $\bar{e}_{\in S}$ and $\bar{e}_{\notin S}$ of $\bar{A} := \tau(A)$ such that a) in $\bar{e}_{\in S}$ the faulty node is a member of S, b) in $\bar{e}_{\notin S}$ all the nodes in S are correct, and c) the two executions are indistinguishable from C. For the full proof, see Appendix A.4. □

4.5 Omission and commission faults

Corollary 1 *If the fault detection problem can be solved for a fault class F, then $F \subseteq F_{OM} \cup F_{CO}$.*

Theorem 4 *There is a solution to the fault detection problem with agreement for the fault class $F_{OM} \cup F_{CO}$.*

For a transformation that solves the fault detection problem for this class, please refer to the proof of Theorem 8, which appears in Appendix A.8.

5 Message complexity

In this section, we investigate how expensive it is to solve the fault detection problem, that is, how much additional work is required to detect faults. The metric we use is the number of messages that must be sent by correct nodes. (Obviously, the faulty nodes can send arbitrarily many messages). Since the answer clearly depends on the original algorithm and on the actions of the faulty nodes in a given execution, we focus on the following two questions: First, what is the maximum number of messages that may be *necessary* for some algorithm, and second, what is the minimum number of messages that is *sufficient* for any algorithm?

5.1 Definitions

If τ is a solution of the fault detection problem, we say that the *message complexity* $\gamma(\tau)$ of τ is the largest number such that for all k, there exists an algorithm A, an execution e of A, and an execution $\bar{e}$ of $\tau(A)$ such that

$$(\mu_e(\bar{e}) = e) \wedge (|e| \geq k) \wedge \left[\frac{|\{\bar{m} \mid \textsc{send}(i, \bar{m}, j) \in \bar{e} \wedge i \in \mathit{corr}(\tau(A), \bar{e})\}|}{|e|} \geq \gamma(\tau) \right]$$

In other words, the message complexity is the maximum number of messages that must be sent by correct nodes in any $\bar{e}$ *per* message sent in the corresponding $e := \mu_e(\bar{e})$. The message complexity of the fault detection problem as a whole is the minimum message complexity over all solutions.

5.2 Commission faults

In this section, we present a collection of tight lower bounds for solving various instances of the fault detection problem. First we show that message complexity of the fault detection problem in the environment E_f for both commission and omission faults is optimally linear in f.

Theorem 5 *Any solution τ of the fault detection problem for F_{CO} in the environment E_f has message complexity $\gamma(\tau) \geq f + 2$, provided that $f + 2 < |N|$.*

Proof sketch: We show that no solution τ can achieve completeness unless, for each pair of messages (m_1, m_2) received by correct nodes, there is at least one correct node that learns about both m_1 and m_2. Since up to f nodes can be faulty in E_f, the cheapest way to achieve this is to forward each message to the same set of $f + 1$ nodes. Since each message must also be sent to its destination, the total message complexity is at least $f + 2$. For the full proof, see Appendix A.5. □

Theorem 6 *The message complexity of the fault detection problem with agreement for F_{CO} in the environment E_f is at most $f + 2$, provided that $f + 2 < |N|$.*

Proof sketch: We construct a solution τ_1 that requires the correct nodes to forward a copy of each incoming message to a set ω of $f + 1$ different nodes. Since at most f nodes can be faulty, there is at least one correct node $c \in \omega$ that has enough information to detect each commission fault. When this node detects a fault on another node j, it constructs a proof of misbehavior and uses reliable broadcast to forward the proof to all the other nodes. Since the proof contains at most two messages, and since each node can be exposed at most once, the broadcast step does not affect the message complexity. For the full proof, see Appendix A.6. □

Corollary 2 *The message complexity of the fault detection problem (with or without agreement) for F_{CO} in environment E_f is $f + 2$, provided that $f + 2 < |N|$.*

5.3 Omission faults

Theorem 7 *Any solution τ of the fault detection problem for F_{OM} in the environment E_f has message complexity $\gamma(\tau) \geq 3f + 4$, provided that $f + 2 < |N|$.*

Proof sketch: To achieve completeness and accuracy, every solution τ must ensure that at least one correct node learns all messages that were sent or received by the correct nodes. However, when a node c learns about a message m that was supposedly sent by some other node j, it cannot know whether j is correct, so c cannot be sure that m will actually reach *dest*(m) unless it forwards m to *dest*(m) itself. Since up to f nodes can be faulty in E_f, at least some set ω with $|\omega| \geq f + 1$ must learn each message, and, for each $c \in \omega$, each message must be forwarded three times: once from the source to c, once from c to the destination, and once from the destination to c. Thus, the message complexity is at least $1 + 3 \cdot (f + 1) = 3f + 4$. For the full proof, see Appendix A.7. □

Theorem 8 *The message complexity of the fault detection problem for F_{OM} in the environment E_f is at most $3f + 4$, provided that $f + 2 < |N|$.*

Proof sketch: We construct a solution τ_2 that forwards each message up to three times from or to a set of $f + 1$ nodes. Thus, the message complexity of τ_2 is $3f + 4$. For the full proof, see Appendix A.8. □

Interestingly, if we additionally require agreement, then the optimal message complexity of the fault detection problem with respect to omission faults is quadratic in $|N|$, under the condition that at least half of the nodes may fail. Intuitively, if a majority of N is known to be correct, it should be possible to delegate fault detection to a set ω with $|\omega| = 2f + 1$, and to have the remaining nodes follow the majority of ω. This would reduce the message complexity to approximately $|N| \cdot (2f + 1)$.

Theorem 9 *Any solution τ of the fault detection problem with agreement for F_{OM} in the environment E_f has message complexity $\gamma(\tau) \geq (|N| - 1)^2$, provided that $\frac{|N|-1}{2} < f < |N| - 2$.*

Fault class	Fault detection problem	Fault detection problem with agreement
F_{CO}	$f+2$ (Corollary 2)	$f+2$ (Corollary 2)
F_{OM}	$3f+4$ (Theorems 7 and 8)	$(\|N\|-1)^2$ (Theorems 9 and 10)

Table 1: Message complexity in environments with up to f faulty nodes.

Proof sketch: In contrast to commission faults, there is no self-contained proof of an omission fault; when a node is suspected of having omitted a message m, the suspicion can always turn out to be groundless when m eventually arrives. We show that, under worst-case conditions, such a 'false positive' can occur after every single message. Moreover, since agreement is required, a correct node must not suspect (or stop suspecting) another node unless every other correct node eventually does so as well. Therefore, after each message, the correct nodes may have to ensure that their own evidence is known to all the other correct nodes, which in the absence of a correct majority requires reliable broadcast and thus at least $(|N|-1)^2$ messages. For the full proof, see Appendix A.9. □

Theorem 10 *The message complexity of the fault detection problem with agreement for F_{OM} in the environment E_f is at most $(|N|-1)^2$, provided that $f+2<|N|$.*

Proof sketch: We construct a solution τ_3 that sends each message via reliable broadcast, and we show that the message complexity of τ_3 is $(|N|-1)^2$. For the full proof, see Appendix A.10. □

5.4 Summary

Table 1 summarizes the results in this section. Our two main results are that a) detecting omission faults has a substantially higher message complexity than detecting commission faults, and that b) the message complexity is generally linear in the failure bound f, except when the fault class includes omission faults *and* agreement is required, in which case the message complexity is quadratic in the system size $|N|$.

6 Related work

There is an impressive amount of work on fault detection in the context of *failure detectors* (starting from the original paper by Chandra and Toueg [6]). However, literature on failure detectors conventionally assumes crash-fault models, and usually studies theoretical bounds on the information about failures that is necessary to solve various distributed computing problems [5], without focusing on the costs of implementing failure detectors.

Faults beyond simple crashes have been extensively studied in the context of arbitrary (Byzantine) fault tolerance (starting from the original paper by Lamport et al. [15]). Byzantine fault-tolerant systems aim to keep faults from becoming "visible" to the system users. One example is Castro and Liskov's Practical Byzantine fault-tolerance (PBFT) [4] that extends Lamport's state-machine replication protocol [14] to the Byzantine failure model. However, BFT systems do not detect and expose faulty nodes.

In the context of *synchronous* Byzantine agreement algorithms, Bar-Noy et al [2] use the terms "fault detections" and "fault masking" in a more restrictive manner than this paper does. In [2], a processor in an agreement protocol is said to be "detected" if all correct processors agree that the processor is faulty. All subsequent actions of this processor are then ignored and thus "masked".

Also with respect to Byzantine agreement algorithms, Bracha [3] describes a protocol in which all messages are broadcast, and in which all nodes track the state of every other node in order to identify messages that could not have been sent by a correct node.

Intrusion detection systems (IDS) can detect a limited class of protocol violations, for example by looking for anomalies [7] or by checking the behavior of the system against a formal specification [13].

A technique that statistically monitors quorum systems and raises an alarm if the failure assumptions are about to be violated was introduced in [1]. However, this technique cannot identify which nodes are faulty.

To the best of our knowledge, Kihlstrom et al. [12] were the first to explicitly focus on Byzantine fault detection. The paper also gives informal definitions of the commission and omission faults. However, the definitions in [12] are specific to consensus and broadcast protocols.

Our notions of facts and evidence in a distributed system are inspired by the epistemic formalism of Halpern and Moses [11].

The results in this paper have important consequences for research on *accountability* in distributed computing. Systems like PeerReview [10] provide accountability by ensuring that faults can eventually be detected and irrefutably linked to a faulty node. Since fault detection is an integral part of accountability, this paper establishes an upper bound on the set of faults for which accountability can be achieved, as well as a lower bound on the worst-case message complexity. Note that practical accountability systems have other functions, such as providing more detailed fault notifications, which we do not model here.

7 Conclusion and future work

In reasoning about computing systems, it is very important to find the right language. Somewhat dangerously, intuitive claims sometimes become "folklore" before they are actually stated precisely and proved. For example, exact bounds on the information about crash failures needed for solving agreement, though informally anticipated earlier [8, 14], were captured precisely only with the introduction of failure detectors [6], and especially the notion of the weakest failure detector [5].

Similarly, this paper has developed a language for reasoning about fault detection with general fault models (beyond simple crash faults). We have proposed a framework in which generic faults can be precisely defined and classified. Unlike crash faults, generic faults cannot be defined without reference to an algorithm, which is why we have introduced the expected system behavior into the definition. To determine the inherent costs of generic fault detection, we have proposed a weak definition of the fault detection problem, and we have derived exact bounds on the cost of solving it in asynchronous message-passing systems where nodes are able to digitally sign their messages.

The framework we have presented can also be used to study fault detection in other system models. If the model is weakened or strengthened (e.g., by varying the assumptions about the network, the degree of synchrony, or the available cryptographic primitives), the kinds of evidence available to correct nodes can change, as can the set of executions that are plausible given some specific evidence. This change, in turn, affects the ability of correct nodes to detect and isolate faulty nodes. For instance, if bounds on communication and processing times are known, it is possible to establish in finite time that an omission fault has occurred, and the culprits can safely be suspected forever. The model could also be changed by introducing bounds on the message size and/or the set of states Σ. These changes would likely increase the message complexity and reduce the size of the fault classes for which detection is possible.

Our framework can be used to study different variants of the fault detection problem. The (weak) formulation of the problem chosen in this paper was primarily instrumental for establishing impossibilities and complexity lower bounds that capture inherent costs of detection in the asynchronous systems. In other scenarios, however, different formulations may make more sense. For example, accuracy could be strengthened such that eventually no correct node is suspected by any correct node; this would require

stronger synchrony assumptions [6, 8]. On the other hand, completeness could be relaxed in such a way that faults must only be detected with high probability. Preliminary evidence suggests that such a definition would substantially reduce the message complexity [10].

In conclusion, we believe that this work is a step toward a better understanding of the costs and limitations of fault detection in distributed systems. We also believe that this work could be used as a basis for extending the spectrum of fault classes with new intermediate classes, ranging between the "benign" crash faults (which have proven to be too restrictive for modern software) and the generic but rather pessimistic Byzantine faults.

References

[1] Lorenzo Alvisi, Dahlia Malkhi, Evelyn Tumlin Pierce, and Michael K. Reiter. Fault detection for Byzantine quorum systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):996–1007, 2001.

[2] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, pages 42–51, August 1987.

[3] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, November 1987.

[4] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, November 2002.

[5] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[7] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[8] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, January 1987.

[9] Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Computer Science Department, May 1994.

[10] Andreas Haeberlen, Petr Kuznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 175–188, October 2007.

[11] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, July 1990.

[12] Kim Potter Kihlstrom, Louise E. Moser, and P. Michael Melliar-Smith. Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16–35, January 2003.

[13] Calvin Ko, George Fink, and Karl Levitt. Automated detection of vulnerabilities in privileged programs using execution monitoring. In *Proceedings of the 10th Annual Computer Security Application Conference*, December 1994.

[14] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[15] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[16] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Sasha. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th USENIX Symposium on Operating System Design and Implementation (OSDI '04)*, pages 121–136, December 2004.

[17] Ben Vandiver, Hari Balakrishnan, Barbara Liskov, and Sam Madden. Tolerating Byzantine faults in transaction processing systems using commit barrier scheduling. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 57–72, October 2007.

A Proofs

A.1 Reduction preserves correctness

Lemma 1 *Let $\bar{A}$ and A be two algorithms for which a reduction $(\bar{A}, A, \mu_m, \mu_s, XO)$ exists. Then, if $\bar{e}$ is an execution in which a node i is correct with respect to $\bar{A}$, i is correct in $\mu_e(\bar{e})$ with respect to A.*

Proof: If i is correct in $\bar{e}$ with respect to $\bar{A}$, there exists a sequence of states $\bar{\sigma}_1, \bar{\sigma}_2, \ldots$ such that for all $k \geq 1$, $\bar{\alpha}_i(\bar{\sigma}_{k-1}, \bar{I}_k) = (\bar{\sigma}_k, \bar{O}_k)$ (where $\bar{I}_k$ and $\bar{O}_k$ are the inputs and outputs in the k.th event of $\bar{e}|_i$). First, we observe that events in $\mu_e(\bar{e})|_i$ can only be generated by rule 2e, and never by rule 2f. To see why, assume the contrary, that is, some event $(i, \emptyset, O_i)$ is added to $\mu_e(\bar{e})$ in response to another event (j, I_j, O_j) in $\bar{e}$ because there exists a message $m_j \in O_j$ such that $\exists m'_j \in \mu_m(m_j)$ with $\mathit{src}(m'_j) = i$. But, since we have assumed that $\bar{A}$ never requires any node to delegate its capability to sign messages, and that i is correct with respect to $\bar{A}$, j could only have sent m_j if it previously received m'_j directly or indirectly from i. But that means that, at the time the event is added, $\text{SEND}(i, m'_j, \mathit{dest}(m'_j)) \in \mu_e(\bar{e})$, so m_j would have been removed in rule 2c, which is a contradiction.

Now we are ready to prove the claim by induction. Let $\xi(k)$ be the index of the event in $\bar{e}|_i$ that caused the event with index k to be added to $e|_i$ (obviously, $\xi(k+1) > \xi(k)$). Also, let $\xi(0) := 0$, and let $\sigma_k := \mu_s(\bar{\sigma}_{\xi(k)})$. Our induction hypothesis is that a) i is correct in the prefix of $e|_i$ that consists of the first k events, *and* that $\sigma_0, \ldots, \sigma_k$ is the corresponding sequence of states. The proof is by induction over k. For $k = 0$, the claim holds because of requirement X3.

Assume the hypothesis holds up to $k-1$. We begin by showing that $\mu_s(\bar{\sigma}_{\xi(k)-1}) = \mu_s(\bar{\sigma}_{\xi(k-1)})$. If $\xi(k) = \xi(k-1)+1$, this is trivially true. Otherwise, we know that rule 2e was never triggered for the events in between, so, in events $\xi(k-1)+1 \ldots \xi(k)-1$, i did not receive any terminal inputs, did not produce any terminal outputs in TO, and any messages i received were either duplicates or mapped to $\perp$ by μ_m, neither of which would cause a state transition in A. Thus, because of homomorphism X6 and because $\alpha(\sigma, \emptyset) = (\sigma, \emptyset)$, we know that $\mu_s(\bar{\sigma}_{\xi(k-1)+j}) = \mu_s(\bar{\sigma}_{\xi(k-1)+j+1)})$ for all $0 \leq j < \xi(k) - \xi(k-1)$.

Now consider the transition from event $\xi(k)-1$ to event $\xi(k)$ in $\bar{e}$. Let $\bar{I^m} := \bar{I}_{\xi(k)-1} \cap \bar{M}$ be the messages received by i in this event, and let $\bar{I^t} := \bar{I}_{\xi(k)-1} \cap \bar{T}I$ be the terminal inputs it received. Also, let $\bar{O^m} := \bar{O}_{\xi(k)-1} \cap \bar{M}$ be the messages it sent, and let $\bar{O^t} := \bar{O}_{\xi(k)-1} \cap TO$ and $\bar{O^x} := \bar{O}_{\xi(k)-1} \cap XO$ be the terminal outputs it produced. From rule 2a, we know that $I_k = \mu_m(I^m) \cup I^t$ (minus any duplicates and

messages with *dest*$(m) \neq i$, to which A would not have responded anyway) and $O_k = \mu_m(O^m) \cup O^t$. Since i is correct in $\bar{e}|_i$ with respect to $\bar{A}$, we also know that $\bar{\alpha}(\bar{\sigma}_{\xi(k)-1}, \bar{I^m} \cup \bar{I^t}) = (\bar{\sigma}_{\xi(k)}, \bar{O^m} \cup \bar{O^t} \cup \bar{O^x})$, and therefore, according to homomorphism X6, $\alpha(\mu_s(\bar{\sigma}_{\xi(k)-1}), \mu_m(\bar{I^m}) \cup \bar{I^t}) = (\mu_s(\bar{\sigma}_{\xi(k)}), \mu_m(\bar{O^m}) \cup \bar{O^t})$. Putting everything together, we get $\alpha(\sigma_{k-1}, I_k) = (\sigma_k, O_k)$, which is the hypothesis for k. □

A.2 Existence of executions of the extension that map to a specific execution

Lemma 3 *Let $\bar{A}$ and A be two algorithms for which a reduction $(\bar{A}, A, \mu_m, \mu_s, XO)$ exists, and let e be an execution of A. Then there exists an execution $\bar{e}$ of $\bar{A}$ such that a) $\mu_e(\bar{e}) = e$ (modulo duplicate messages sent by faulty nodes in e), and b) a node i is correct in $\bar{e}$ with respect to $\bar{A}$ iff it is correct in e with respect to A.*

Proof: Let e be an arbitrary execution of A, and let C be the set of nodes that are correct in e with respect to A. We can iteratively construct an execution $\bar{e}$ of $\bar{A}$ with $\mu_e(\bar{e}) = e$ and $\bar{C} = C$ (where $\bar{C}$ is the set of nodes that are correct in $\bar{e}$ with respect to $\bar{A}$) as follows. During the construction, we will maintain the following three invariants: After mapping each prefix $x := e_1, \ldots, e_k$ of e to a prefix $\bar{x}$ of $\bar{e}$, 1) $\mu_e(\bar{x}) = x$ (modulo duplicates sent by faulty nodes), 2) for each distinct message m that is in flight in x, RECV(*dest*$(m), m) \notin \bar{x}$ and there exists a message $\bar{m}$ with $\mu_m(\bar{m}) = m$ and *dest*$(\bar{m}) =$ *dest*(m) that is in flight in $\bar{x}$, and 3) if i is correct, σ is the state i is in after x, and $\bar{\sigma}$ is the state i is in after $\bar{x}$, then $\mu_s(\bar{\sigma}) = \sigma$.

Initially, $x = \bar{x} = \emptyset$, so all three invariants trivially hold. Then we perform the following two steps for each $e_k = (i_k, I_k, O_k)$ of e. First, we construct a set $\bar{I}_k$ by taking the terminal inputs from I_k and, for each message $m \in I_k$, a message $\bar{m}$ with $\mu_m(\bar{m}) = m$ and *dest*$(\bar{m}) =$ *dest*(m) that has not yet been delivered in $\bar{e}$ (which must exist according to invariant 2). If $i_k \in C$, we then construct $\bar{O}_k$ by evaluating $\bar{\alpha}_{i_k}$; otherwise we simply use the terminal outputs in O_k plus, for each message m in O_k, some message $\bar{m}$ with $\mu_m(\bar{m}) = m$. Then we add $(i_k, \bar{I}_k, \bar{O}_k)$ to $\bar{e}$.

After the first step, all three invariants still hold. To see why the first invariant holds, we consider what rules in μ_e could have been invoked by the addition of the new event. Rule 2f cannot have been invoked because nodes did not share their key material in $\bar{e}$, rule 2c affects only duplicate messages sent in e, and rule 2b cannot have been invoked because, according to invariant 2, the messages we just delivered were sent to i_k and have not been delivered before. Hence, *exactly one* event (i_y, I_y, O_y) has been added to e (through rule 2e). Since events generated by rule 2e always occur on the same node as the original event, we know that $i_y = i_k$. Furthermore, by construction, the set of terminal in-/outputs in $\bar{I}_k$ and $\bar{O}_k$ is the same as in I_k and O_k (except for possible fault notifications in $\bar{O}_k$), and we know $\mu_m(\bar{I}_k \cap \bar{M}) = (I_k \cap M)$ because of the way we chose the messages in $\bar{I}_k$. It remains to be shown that $\mu_m(\bar{O}_k \cap \bar{M}) = (O_k \cap M)$ if i_k is correct; this is the case because of the homomorphism X6 between $\bar{\alpha}$ and α. Hence, according to the definition of rule 2a, $I_y = I_k$ and $O_y = O_k$. The second invariant holds because the only new messages in flight in x are the ones sent in e_k. We have already seen that for each $m \in O_k$, $\bar{O}_k$ contains a message that maps to m, which is now in flight in $\bar{x}$. Finally, the third invariant holds because of homomorphism X6.

If we used only the first step, messages of $\bar{A}$ that do not map to a message of A would never get delivered. Hence the second step, which works as follows: We begin by determining the set $\bar{X}$ of messages $\bar{m}$ that are currently in flight in $\bar{p}$ and have $\mu_m(\bar{m}) = \perp$. For each node j, we then add an event $(j, \bar{I}_j, \bar{O}_j)$ to $\bar{e}$, where $\bar{I}_j$ contains the messages in $\bar{X}$ whose destination is j, and $\bar{O}_j$ is calculated by invoking $\bar{\alpha}_j$. Because of homomorphism X6, and because $\alpha_j(\sigma, \emptyset) = (\sigma, \emptyset)$, we know that for any message $\bar{m} \in \bar{O}_j$, $\mu_m(\bar{m}) = \perp$; further, if $\bar{\sigma}_{old}$ is the old state of j and $\bar{\sigma}_{new}$ is its new state, we know that $\mu_s(\bar{\sigma}_{old}) = \mu_s(\bar{\sigma}_{new})$, that is, j's state with respect to the original algorithm A does not change.

After the second step, all three invariants still hold. The first invariant holds because no event has been added by μ_e (recall that there were no terminal inputs, no terminal outputs from TO, and that for all $\bar{m}$ we added, $\mu_m(\bar{m}) = \perp$; hence, after rule 2a, the sets I and O would be empty). The second invariant holds

because the set of in-flight messages that map to A's messages has not changed. The third invariant holds because it held after the first step and, as explained earlier, $\mu_s(\bar{\sigma}_{old}) = \mu_s(\bar{\sigma}_{new})$.

We still need to show that $\bar{e}$ is an execution, and that $\bar{C} = C$. $\bar{e}$ is an execution because a) any messages that map to A's messages are delivered at the same point as in e, and b) any other messages are delivered after at most $|N| + 1$ events. Hence, no messages remain undelivered, and by construction, no messages can be delivered unless they have previously been sent. We know that $C \subseteq \bar{C}$ because we have derived all events on correct nodes using their transition functions. To see why $(N \setminus C) \subseteq (N \setminus \bar{C})$, consider that each $j \in (N \setminus C)$ is faulty because it has performed an incorrect state transition, which will mapped to an incorrect state transition in $\bar{e}$. □

A.3 The fault detection problem cannot be solved for non-observable faults

Theorem 2 *The fault detection problem cannot be solved for any fault class F with $F \cap F_{NO} \neq \emptyset$.*

Proof: The proof works by showing that, for any fault instance $\psi := (A, C, S, e) \in F_{NO}$, we can construct two executions $\bar{e}_{good}$ and $\bar{e}_{bad}$ of $\bar{A} := \tau(A)$ such that a) all the nodes are correct in $\bar{e}_{good}$, b) the fault occurs in $\bar{e}_{bad}$, and c) the two executions are indistinguishable from the perspective of the nodes in C (that is, $\bar{e}_{good}|_C = \bar{e}_{bad}|_C$). Hence, the nodes in C would have to both expose some node in S (to achieve completeness in $\bar{e}_{bad}$) and *not* expose any node in S (to achieve accuracy in $\bar{e}_{good}$) based on the same information, which is impossible.

Suppose there is a solution τ to the fault detection problem for some class F with $\psi := (A, C, S, e) \in F_{NO} \cap F$, and let $e_{bad} := e$. Since $\psi \in F_{NO}$, there exists an execution $e_{good} \in \pi(A, \phi^{\pm}(C, e), N)$. By Lemma 3, there exists an execution of $\bar{A}$ that is mapped to e_{good} by μ_e and in which all nodes are correct with respect to $\bar{A}$; we can simply choose that execution to be $\bar{e}_{good}$.

Next, we construct $\bar{e}_{bad}$ from $\bar{e}_{good}$. We assume that all the nodes in $N \setminus C$ collude; hence, we can freely choose the behavior of these nodes without considering $\bar{A}$. First, we construct an execution $\bar{e}_1$ as follows: We remove from $\bar{e}_{good}$ all events (i_k, I_k, O_k) where $i_k \in N \setminus C$, and then we add a $\text{RECV}(m, j)$ event for every m sent by a node in C to a node $j \in N \setminus C$, as well as a $\text{SEND}(i, m, j)$ event for every m received by a node $i \in C$ from a node $j \in N \setminus C$. When adding events for messages m with $\mu_m(m) \neq \perp$, we add them in the same order as in e_{bad}. This already ensures that $\bar{e}_1|_C = \bar{e}_{good}|_C$.

However, $\bar{e}_1$ is not a valid execution yet because in $\bar{e}_1$, a node $k \in N \setminus C$ can send a message m that was originally sent by a node $i \in C$ to another node $j \in N \setminus C$, $j \neq k$. Since we assumed that faulty nodes cannot forge the signature of a correct node, k cannot do this without having received m from j first. Therefore, after every $\text{RECV}(m, j)$ event with $j \in N \setminus C$, we add a $\text{SEND}(j, \text{FWD}(m), k)$ / $\text{RECV}(k, \text{FWD}(m))$ pair for every node $k \in N \setminus C$, $k \neq j$, where FWD is a message that is not in $\bar{M}$, thus arriving at an execution $\bar{e}_2$. Note that, since the system is asynchronous, we can choose the message delays in $\bar{e}_2$ arbitrarily.

$\bar{e}_2$ is a valid execution, but we do not yet have $\mu_e(\bar{e}_2) = e_{bad}$. However, the only missing events are terminal in/outputs on the nodes in $N \setminus C$, as well as SEND and RECV events for messages sent among the nodes in $N \setminus C$. The former can easily be added by inserting IN and OUT events. To add the latter, we proceed as follows: For each message m sent from a node j to a node k, $j, k \in N \setminus C$, we pick any $\bar{m} \in \bar{M}$ with $\mu_m(\bar{m}) = m$ and insert $\text{SEND}(j, m, k)$ and $\text{RECV}(k, m)$ events. Of course, any inserted events must be added in the same sequence in which they occurred in E_{bad}. Note that, since the nodes in $N \setminus C$ have shared their key material, there is no need to broadcast the messages among them. The result is the execution $\bar{e}_{bad}$, and we have $\mu_e(\bar{e}_{bad}) = e_{bad}$.

Now let $c \in C$ be a correct node. Because of nontriviality, c must output infinitely many fault notifications in $\bar{e}_{good}$ and $\bar{e}_{bad}$. Because of completeness, the notifications in $\bar{e}_{bad}$ must eventually all contain some non-empty subset of S. Let s be a node in that subset. Because of accuracy, infinitely many of the

notifications in $\bar{e}_{good}$ must not contain s. But because $\bar{e}_{good}|_C = \bar{e}_{bad}|_C$ and $\bar{A}$ is deterministic, c must output the *same* fault notifications in $\bar{e}_{good}$ and $\bar{e}_{bad}$. This is a contradiction. □

A.4 The fault detection problem cannot be solved for ambiguous fault instances

Theorem 3 *The fault detection problem cannot be solved for any fault class F with $F \cap F_{AM} \neq \emptyset$.*

Proof: The proof is largely analogous to that of Theorem 2, except that we now construct two executions $\bar{e}_{\in S}$ and $\bar{e}_{\notin S}$ of $\bar{A} := \tau(A)$ such that a) in $\bar{e}_{\in S}$ the faulty node is a member of S, b) in $\bar{e}_{\notin S}$ all the nodes in S are correct, and c) the two executions are indistinguishable from C.

Suppose there is a solution τ to the fault detection problem for some class F with $\psi := (A, C, S, e) \in F_{AM} \cap F$, and let $e_{\in S} := e$. Since $\psi \in F_{AM}$, we have $\pi(A, \phi^{\pm}(C, e_{\in S}), N) = \emptyset$, but $\pi(A, \phi^{\pm}(C, e_{\in S}), C \cup S) \neq \emptyset$. Let $e_{\notin S}$ be an execution in $\pi(A, \phi^{\pm}(C, e_{\in S}), C \cup S)$; note that in $e_{\notin S}$, all the nodes in $C \cup S$ are correct with respect to A. Let $\bar{e}_{\notin S}$ be an execution with $\mu_e(\bar{e}_{\notin S}) = e_{\notin S}$ in which all the nodes in $C \cup S$ are correct with respect to $\bar{A}$. Such an execution must exist according to Lemma 3.

Our goal is to construct an execution $\bar{e}_{\in S}$ such that $\mu_e(\bar{e}_{\in S}) = e_{\in S}$ and $\bar{e}_{\in S}|_C = \bar{e}_{\notin S}|_C$. If we assume that all the nodes in $N \setminus C$ are faulty and collude in $\bar{e}_{\in S}$, this can be done as follows. We start with $\bar{e}_{\in S} := \bar{e}_{\notin S}|_C$ and, for every message m that was sent by a node $s \in N \setminus C$ to a node $c \in C$, we add a transmission event $\text{SEND}(s, m, c)$ to $\bar{e}_{\in S}$. Similarly, for every message m that was sent from C to S, we add a receive event to $\bar{e}_{\in S}$. Finally, for each message m that is sent between two nodes $s_1, s_2 \in N \setminus C$ in $e_{\in S}$, s_1 and s_2 can send some message $\bar{m}$ with $\mu_m(\bar{m}) = m$ in $\bar{e}_{\in S}$. This ensures that $\mu_e(\bar{e}_{\in S}) = e_{\in S}$.

Now consider any $c \in C$. Because of nontriviality, c must output infinitely many fault notifications in $\bar{e}_{\in S}$ and $\bar{e}_{\notin S}$. Because of completeness, the notifications in $\bar{e}_{\in S}$ must eventually all contain some $s \in S$, and because of accuracy, infinitely many of the notifications in $\bar{e}_{\notin S}$ must not contain s. But because $\bar{e}_{\in S}|_C = \bar{e}_{\notin S}|_C$ and $\bar{A}$ is deterministic, c must output the *same* fault notifications in $\bar{e}_{\in S}$ and $\bar{e}_{\notin S}$. This is a contradiction. □

A.5 Lower bound for commission faults

Theorem 5 *Any solution τ of the fault detection problem for F_{CO} in the environment E_f has message complexity $\gamma(\tau) \geq f + 2$, provided that $f + 2 < |N|$.*

Proof: The claim follows if, for any given k, we can construct an execution e_k of some algorithm A such that any solution $\tau(A)$ must send at least $(f + 2) \cdot |e_k|$ messages in any execution $\bar{e}_k$ with $\mu_e(\bar{e}_k) = e_k$.

We begin by choosing the algorithm A as follows. Each node i locally maintains a set B_i of bitstrings, which is initially empty. When i receives an input (j, x) from its local terminal, it checks whether $x \in B_i$; if so, it ignores the input, otherwise it adds x to B_i and sends a message x to node j. Note there is only one type of commission fault, namely sending the same bitstring to two different correct nodes. For any $k \geq 1$, we can construct an e_k such that a single node i receives k inputs whose bitstrings are all different and whose node identifiers are from a specific set X, which we define below. Note that i sends k messages in e_k, so $|e_k| = k$.

First, we observe that $\tau(A)$ cannot achieve completeness unless it ensures that, for each pair of messages (m_1, m_2) that is received by correct nodes, there is at least one correct node c that learns about both m_1 and m_2. If this were not the case for some pair of messages, then it could be that this pair was a duplicate, and τ would not be able to detect this. However, since τ cannot be sure that any particular node is correct with respect to $\tau(A)$, the only way to achieve this goal is to ensure that each pair of messages is made known to at least $f + 1$ different nodes. Since at most f nodes can be faulty in E_f, this ensures that at least one of them is correct.

Next, we show that τ cannot send messages in batches. According to homomorphism X6, $\tau(A)$ must send at least one message whenever A sends one, and since the terminal inputs arrive one by one in e_k, A also sends messages one by one, so $\tau(A)$ has no opportunity to combine the original transmissions. We have seen earlier that correct nodes must forward messages to certain other nodes, and τ could potentially batch those; however, since the original transmissions are driven by terminal inputs, no node can be sure at any point that there will be a next message. Thus, if any transmission were delayed, it could be that this message was the last one, and it might never be sent.

The cheapest way to ensure that each pair of messages is seen by $f+1$ different nodes is to forward each message to the *same* set of nodes ω with $|\omega| = f+1$. This can be done with f messages if the original recipient belongs to ω; however, whatever the set is, we can adjust the set X of recipients in e_k such that it does not contain any nodes from ω (this is possible only if there are other nodes besides the $f+1$ nodes in ω and the sender i; hence the restriction to $f+2 < |N|$). Therefore, $\tau(A)$ will need at least one message for each of the k original transmissions in e_k plus $f+1$ messages to forward each of them to ω, which amounts to $(f+2) \cdot k$ messages in total. □

A.6 Upper bound for commission faults

Theorem 6 *The message complexity of the fault detection problem with agreement for F_{CO} in the environment E_f is at most $f+2$, provided that $f+2 < |N|$.*

Proof: The claim follows if we can construct at least one transformation that solves the problem with agreement for F_{CO} in E_f and, in any execution $\bar{e}$ with $\mu_e(\bar{e}) = e$, at most $(f+2) \cdot |e|$ messages are sent by correct nodes in $\bar{e}$.

We choose the transformation τ_1 that works as follows: Every correct node i takes each message m it receives, attaches to it the complete sequence of steps it has taken so far, and then forwards m to a fixed set of nodes ω with $|\omega| = f+1$. Note that, since messages are authenticated, i effectively commits to a particular execution. Whenever the nodes in ω receive a forwarded message, they recursively extract all the sequences of steps it contains (note that each message in a sequence of steps contains another sequence of steps), and they combine them with the sequences they have seen previously. Finally, for every node $j \in N$, they check the following two conditions: whether 1) there is a sequence of steps from j that is not correct with respect to A_j, and whether 2) they find two sequences of steps from j neither of which is a prefix of the other. If either of these is the case, they take the (at most two) messages that contain these sequences of steps and use reliable broadcast to forward them to all the other nodes in the system. When any correct node receives such messages via reliable broadcast, it repeats the above checks and, if successful, it exposes j; if the tests fail, it does not forward the message further.

Complexity: Clearly, τ_1 has message complexity $f+2$; there is no way for the faulty nodes to cause a correct node to send additional messages without also adding new messages to $\mu_e(\bar{e})$, and the reliable broadcast step requires at most a fixed number of messages, since each of the nodes in N can be exposed at most once. Hence, we only need to show that τ_1 solves the fault detection problem with agreement for F_{CO} in environment E_f.

Completeness: We begin by showing that, to detect faults in F_{CO}, it is sufficient to know the messages received by correct nodes. Recall that $(A, C, S, e) \in F_{CO}$ iff $\pi(A, \phi^+(C, e), C \cup S) = \emptyset$, but according to completeness, it must be exposed only if all the nodes in C are correct, so $\phi^+(C, e)$ corresponds to all the message sent or received by correct nodes. Now let $\phi^R(C, e)$ be only the facts that correspond to messages *received* by correct nodes. If there were some execution $e' \in \pi(A, \phi^R(C, e), C \cup S)$, we can construct an execution $e'' \in \pi(A, \phi^+(C, e), C \cup S)$ by taking the prefix x of e' until the last message is sent from S to C, and then delivering any extra messages sent from C to S after x. The only way for the nodes in C to

distinguish e'' from e' is if any required messages are sent from S to C after x that are not received by C; however, this would require a negative fact.

Now let c be a correct node in ω (which must exist because $|\omega| > f$). Since each correct node forwards all messages it receives to each node in ω, c eventually learns $\phi^R(C, e)$. c may learn other facts as well (e.g., messages received by faulty nodes), but, according to the lemma 4 in Section 4.1, knowing more facts can only shrink the set of plausible executions, and thus cannot prevent c from detecting a commission fault.

Now consider a set of nodes S that has performed a commission fault, and recall that each node must commit to a particular execution prefix whenever it sends a message. If any node in S has committed to two prefixes e_1 and e_2 such that neither $e_1 \lfloor e_2$ nor $e_2 \lfloor e_1$, that node will be exposed by c, and completeness holds. Otherwise there is a longest prefix for each node in S. If all these prefixes were correct, $\pi(A, \phi^+(C, e), C \cup S)$ could not be empty; hence, one of them has to be faulty, and the corresponding node will be exposed by c. Again, completeness holds.

Accuracy and agreement: A correct node will never commit to two execution prefixes e_1 and e_2 such that neither $e_1 \lfloor e_2$ nor $e_2 \lfloor e_1$, and any prefix to which it commits will be correct. So a correct node can never be exposed by a correct node, and accuracy holds. Also, a correct node will only expose a node j after having made sure that every other correct node will eventually obtain evidence of misbehavior against j. Hence, agreement holds. □

A.7 Lower bound for omission faults

Theorem 7 *Any solution τ of the fault detection problem for F_{OM} in the environment E_f has message complexity $\gamma(\tau) \geq 3f + 4$, provided that $f + 2 < |N|$.*

Proof: In the following, we choose the family of algorithms A_k to be one where each node i keeps two local counters β_i and ν_i, both of which are initially zero. When i receives an input (j, x) from its local terminal, it sends a message $\text{ADD}(x)$ to node j. When j receives this message, it outputs x to its local terminal, then increments ν_j, and finally adds x to β_j. If $\nu_j \equiv 0 \bmod k$ and $\beta_j \equiv 0 \bmod 2$, j responds to i with a $\text{ADD}(1)$ message. Note that with A_k, the only possible omission fault is when a node receives k numbers whose sum is even, but does not return an $\text{ADD}(1)$ message.

Now consider an execution e_k of A_k in which some set X of nodes sends k numbers to a node $j \notin X$ such that the sum is even. We observe that in the corresponding execution $\bar{e}_k$, it is necessary that at least one correct node learns every single message sent or received by a correct node to decide whether or not an omission fault exists. Indeed, if at least one message is missing, we can always find a compatible correct execution in which the node is correct, which allows the node to omit messages without any risk. Therefore, any solution τ must ensure that at least one correct node c receives a copy of each message sent or received by any correct node. Since up to f nodes can be faulty in E_f, this requires that at least some set ω of nodes with $|\omega| \geq f + 1$ receives each of these messages, which requires a message complexity of at least $1 + 2 \cdot (f + 1) = 2f + 3$.

However, this is not yet sufficient because, since a correct node c may not know the exact set of nodes that are correct with respect to $\tau(A_k)$ (only a superset of it), it cannot avoid considering messages sent or received by faulty nodes. If c considers a message m that a faulty node j claims to have received from a node i, this is not a problem because, if m is authentic, c can safely conclude that $\text{SEND}(i, m, j)$ must appear in any plausible execution. However, if c considers a message m that a faulty node j claims to have *sent* to a node i, c *cannot* conclude that $\text{RECV}(i, m)$ must appear, since j may never actually have sent m to i. This can cause c to violate both completeness and accuracy. Note that waiting to see whether i forwards m after receiving it is not an option in this case, since c would have to suspect j in the meantime, and may end up suspecting j forever if i is faulty and ignores m, which would violate accuracy.

The only way c can be sure that every forwarded message m eventually reaches *dest*(m) is if at least one correct node has sent m to *dest*(m). But c does not know that any individual node (except c itself) is correct. So c can either wait for $f+1$ different nodes to assert that they have sent m to *dest*(m), which would require at least $f+1$ messages, or it can itself forward m to *dest*(m), which is clearly cheaper. Since every node in ω must do this, the overall message complexity is at least $1+3\cdot(f+1)=3f+4$. □

A.8 Upper bound for omission faults

Theorem 8 *The message complexity of the fault detection problem for F_{OM} in the environment E_f is at most $3f+4$, provided that $f+2<|N|$.*

Proof: We can construct a solution τ_2 as follows. For each algorithm A, τ_2 picks an arbitrary set of nodes ω with $|\omega|=f+1$. Whenever A sends or receives a message m on a correct node i, $\tau_2(A)$ sends m to *dest*(m), then it attaches i's complete execution prefix with respect to A up to the transmission or reception of m, and then forwards m to each node in ω. Each correct node $c\in\omega$ maintains a set M_{out}^{approx} of messages than c knows to have been sent in $\mu_e(\bar{e})$, a set M_{in}^{approx} c knows to have been received by their destination in $\mu_e(\bar{e})$, and a set EP of prefixes c has received. When c receives a forwarded message m, it recursively extracts all the prefixes (i.e., the prefix ep_m attached to m, the prefixes attached to any message sent or received in ep_m, etc) and adds them to EP. Also, c recursively extracts all the sent messages from these prefixes and, for each such message m' that is not yet in M_{out}^{approx}, c forwards m' to *dest*(m') and then adds m' to M_{out}^{approx}. Finally, c recursively extracts all the received messages from these prefixes and adds them to M_{in}^{approx}.

Let j be any node other than c, let ep_j be the longest prefix of j in EP, and let X_j be the set of all messages m that have been sent to j (i.e., $j\in M_{out}^{approx}$ and *dest*$(m)=j$) but not yet received in ep_j. Then c suspects j iff a) EP contains a prefix of j that is not correct with respect to A_j, b) EP contains a prefix that is not itself a prefix of ep_j, or c) there is no infinite extension ep'_j of ep_j such that ep'_j is correct with respect to A_j, the inputs in $ep'_j\setminus ep_j$ are the messages in X_j in any order plus an arbitrary number of terminal inputs, and j does not send any messages in $ep'_j\setminus ep_j$ (recall that, in Section 2.2, we assumed that this is decidable). As a special case, c remembers the first message m_1 in X_j and, if m_1 appears in the next instance of ep_j, c precedes its next fault notification by one that does not contain j.

It is easy to see that each $\tau_2(A)$ is an extension of A, and that it satisfies the nontriviality requirement. Therefore, we only need to show that τ_2's complexity is at most $3f+4$, and that each $\tau_2(A)$ satisfies the completeness and accuracy requirements.

Complexity: Correct nodes send each outgoing message to the corresponding destination, and they forward each incoming and outgoing message to the $f+1$ nodes in ω, which requires at most $2f+3$ messages for each message in $\mu_e(\bar{e})$. In addition, each node in ω forwards each message in $\mu_e(\bar{e})$ to its destination at most once, which requires another $f+1$ messages and brings the total to $3f+4$. The only way for a faulty node to cause the nodes in ω to send additional messages is to add more messages to the attached prefixes, but these messages are inevitably mapped to $\mu_e(\bar{e})$ as well.

Completeness: First, consider what it means for $\pi(A,\phi^{\pm}(C,e),C\cup S)$ to be empty when $\pi(A,\phi^{+}(C,e),C\cup S)$ is not: it means that any execution e' that is plausible given the messages sent and received by correct nodes is ruled out by the fact that some message m should have been sent from or to C in e' but was not sent from or to C in e. Clearly, the first case will be detected because of the prefixes included with each message; if the nodes in S sent some message m' that they could only send they had previously received some message m from the nodes in C, then the prefix included with m' would have to include m', which is impossible because the faulty nodes cannot forge messages by a correct node. Hence, only the second case remains: Every $e'\in\pi(A,\phi^{+}(C,e),C\cup S)$ must contain at least one message m sent from some $s\in S$ to some $c\in C$ such that m is never received by C in e.

At least one correct node $c \in \omega$ needs to know $\phi^+(C, \mu_e(\bar{e}))$, that is, the sets M_{out} and M_{in} of messages sent and received by correct nodes, respectively. However, c can safely approximate them by M_{out}^{approx} and M_{in}^{approx}. Clearly, we eventually get $M_{out} \subseteq M_{out}^{approx}$ and $M_{in} \subseteq M_{in}^{approx}$, since correct nodes forward incoming and outgoing messages to c. The approximate sets may also contain messages sent between faulty nodes, but this does not affect completeness, since additional facts can only reduce the set of executions $\pi(A, \phi^+(C, e), C \cup S)$; hence, if the above condition holds for the entire set, it most certainly holds for any of its subsets as well. Furthermore, the message m in the above condition is a message sent to a correct node j, and, since the messages of a correct node cannot be forged, the faulty nodes cannot cause m to be added to M_{in}^{approx} unless j has actually received it.

Now consider a fault instance $(A, C, S, e) \in F_{OM}$. If $|S| = 1$, c will suspect the node in S because of condition c). If $|S| > 1$, there is no correct execution of the nodes in S that is consistent with the facts known to c, so at least one of the nodes in S must eventually commit to an incorrect prefix, which will cause c to suspect it due to condition a) or b), or stop sending messages to c altogether, which causes c to suspect it due to condition c). The special case does not affect completeness because, if a faulty node j refuses to *send* a message that is required for some input, the corresponding prefix cannot be correct with respect to A_j, and if j refuses to *receive* a message, that message will eventually be the first message in X_j, and the special case will no longer apply.

Accuracy: Clearly, a correct node j will never commit to an incorrect prefix, so no $c \in \omega$ will ever suspect j due to conditions a) or b). If c suspects j due to condition c), j will eventually receive each of the messages in X_j because c has forwarded them to j, and j will send at least one more message m, which it will also forward to c. m will contain a new prefix, which will cause c to re-evaluate j. It is possible, of course, that c will continue to suspect j after learning of m because new messages have been added to X_j in the meantime; however, in this case, the special case will cause c to occasionally emit fault notifications that do not contain j.

Note that τ_2 is a generalization of the transformation τ_1 we described in the proof of Theorem 6, that is, it also solves the fault detection problem for F_{CO}. □

A.9 Lower bound for omission faults (with agreement)

Theorem 9 *Any solution τ of the fault detection problem with agreement for F_{OM} in the environment E_f has message complexity $\gamma(\tau) \geq (|N| - 1)^2$, provided that $\frac{|N|-1}{2} < f < |N| - 2$.*

Proof: Let τ be any solution of the fault detection problem with agreement for F_{OM} in environment E_f, and let A be the following simple algorithm. When a node i receives a node identifier $j \in N$ from its local terminal, it sends a PING message to j, to which j must respond with a PONG message to i. Given any $k \geq 1$, we now construct an execution of $\tau(A)$ with at least k messages such that a status change is necessary after every single message. Let a and b two different nodes, and let $\bar{p}$ be an execution prefix that is initially empty.

We begin by adding a $\text{IN}(a, b)$ event to $\bar{p}$, which causes a to send a PING message m_1 to b. We know that, if b were to crash-fault now and never send a PONG message, this would be an omission fault; hence, τ must ensure that at least one correct node learns about m_1 and exposes b (completeness), and that any correct node that exposes b first establishes that all the other correct nodes will eventually expose b as well (agreement). Since $2f + 1 > N$, the only way to achieve this is reliable broadcast [9] among the nodes in $N \setminus \{b\}$, which requires $(|N|-1)(|N|-2)$ messages. Further, to ensure accuracy, each of the $|N|-1$ nodes would have to forward m_1 to b at some point, which requires another $|N| - 1$ messages and thus brings the total to $(|N| - 1)^2$. We now extend our prefix $\bar{p}$ by simulating a crash fault on b (i.e., by not delivering any message that is sent towards b), and by letting $\tau(A)$ run until it sends no more messages. At this point, assuming that all the nodes are correct, every node except b must have exposed b, and there must be $|N| - 1$ copies of m_1 in flight towards b.

Now we start delivering these messages. Since b is correct, it will respond to each by conveying the fact that it has sent a PONG message m_2 to a, which requires $|N| - 1$ messages. To ensure accuracy, each other node must remove b from its fault notifications once it learns about m_2; however, it must only do so when it is sure that all other nodes know about m_2 as well. Again, since $2f + 1 > |N|$, this requires reliable broadcast and thus another $(|N| - 1)(|N| - 2)$ messages. Thus, we now have an execution prefix $\bar{p}$ of $\tau(A)$ in which two messages are sent in $\mu_e(\bar{p})$ but at least $2(|N| - 1)^2$ messages are sent in $\bar{p}$. Since the situation at the end of $\bar{p}$ is exactly the same as in the beginning (no messages in flight, no nodes exposed), we can extend $\bar{p}$ arbitrarily by repeating the above steps, until we have a prefix with at least k messages. Hence, the message complexity is at least $(|N| - 1)^2$. □

A.10 Upper bound for omission faults (with agreement)

Theorem 10 *The message complexity of the fault detection problem with agreement for F_{OM} in the environment E_f is at most $(|N| - 1)^2$, provided that $f + 2 < |N|$.*

Proof: We can construct a solution τ_3 by extending the transformation τ_2 from the proof of Theorem 8 as follows: Rather than forwarding messages in the way specified above, we require them to be sent via reliable broadcast. This requires $(|N| - 1)^2 \cdot k$ messages, which is exactly our budget. Furthermore, since reliable broadcast implies that each node sends each new message to each of the other nodes, it implies in particular that correct nodes forward each incoming and outgoing message m to each node in ω, and that the nodes in ω send each forwarded m to *dest*(m). Hence, we know that the nodes in ω obtain enough information such that one of them will eventually detect any fault in F_{OM}. However, because of the reliable broadcast, we also know that any given fact that is known to one correct node is eventually known to all; hence, if one correct node suspects a node i forever, all the others will eventually suspect i forever as well. Therefore, agreement holds.

Note that τ_3 also solves the fault detection problem with agreement for F_{CO} as a special case (although it requires more messages than the solution to Theorem 6). This proves Theorem 4. □

Symbol	Meaning
N	Set of nodes
i, j	Node identifiers
A_i	Algorithm of a node i
α_i	Transition function of a node i
Σ_i	Set of states of a node i
σ_i	State of a node i
TI, TO	Sets of terminal inputs and outputs
M	Set of messages
m	Individual message
$\mathit{src}(m)$	Source of a message m
$\mathit{dest}(m)$	Destination of a message m
I, O	Sets of inputs and outputs
e	Execution
$e\|_X$	Projection of an execution onto a set of nodes X
$\|e\|$	Number of messages in an execution
$e_1 \lfloor e_2$	Execution e_1 is a prefix of execution e_2
$\mathit{corr}(A, e)$	Set of nodes that are correct in e with respect to A
C	Set of correct nodes
c	Individual correct node
S	Set of suspect nodes
ψ	Fault instance
F	Fault class
ν	Set of fault notifications
τ	Transformations that solve the fault detection problem
μ_m	Message map
μ_e	Execution map
μ_s	State map
$\bar{e}, \bar{A}, \ldots$	Equivalent of $e, A, \ldots$ for the extension
ζ	Individual fact
Z	Set of facts
$\phi^+(C, e)$	Fact map (messages sent or received by some $c \in C$)
$\phi^-(C, e)$	Fact map (messages *not* sent or received by any $c \in C$)
$\phi^\pm(C, e)$	Fact map ($\phi^+(C, e) \cap \phi^-(C, e)$)
$\pi(A, Z, C)$	Plausibility map
E_f	Environment with failure bound f
f	Maximum number of faulty nodes
$\gamma(\tau)$	Message complexity of a solution τ
ω	Set of witnesses
$P(X)$	Power set of a set X
k, l	General-purpose index variables
X	General-purpose set
$\perp$	Denotes messages that μ_m does not map to anything

Table 2: Notation used in this paper