A Knowledge-Theoretic Analysis of Uniform Distributed Coordination and Failure Detectors

Joseph Y. Halpern*
Cornell University
Dept. of Computer Science
Ithaca, NY 14853
halpern@cs.cornell.edu
http://www.cs.cornell.edu/home/halpern

Aleta Ricciardi
Information Sciences Research Center
Bell Labs – Lucent Technologies
Murray Hill, NJ 07974
aleta@research.bell-labs.com
http://www.bell-labs.com/ aleta

Abstract

It is shown that if there is no bound on the number of faulty processes, then in a precise sense, in a system with unreliable but fair communication, Uniform Distributed Coordination (UDC) can be attained if and only if a system has *perfect* failure detectors. This result is generalized to the case where there is a bound t on the number of faulty processes. It is shown that a certain type of generalized failure detector is necessary and sufficient for achieving UDC in a context with at most t faulty processes. Reasoning about processes' knowledge as to which other processes are faulty plays a key role in the analysis.

1 Introduction

Periodically coordinating specific actions among a group of processes is fundamental to most distributed computing problems, and especially to replication schemes that achieve fault tolerance. Unfortunately, as is well known, it is impossible to achieve coordination in an asynchronous setting even if there can be at most one faulty process [FLP85]. This is true even if communication is reliable. As a result, there has been a great deal of interest recently in systems with *failure detectors* [CT96], oracles that provide suspicions as to which processes in the system are faulty. This interest is heightened by results of [CT96, CHT96] showing that consensus can be achieved with relatively unreliable failure detectors.

Here we consider what kind of failure detectors are necessary to attain *Uniform Distributed Coordination* (UDC) [GT89]. We have UDC of action α if, whenever some process (correct or not) performs α, then so do all the correct processes. If we have reliable communication, then it is easy to see that we can attain UDC no matter how many processes may fail. Thus, in this setting, UDC is strictly easier than consensus. If communication is unreliable but fair, then we show that we can attain UDC with no bound on the number of process failures in the presence of *weak* failure detectors, which have the property that eventually every faulty process is permanently suspect by at least one correct process (*weak completeness*) and at least one correct process is never suspected (*weak accuracy*). Chandra and Toueg [CT96] showed that consensus with an arbitrary number of failures is also achievable using weak failure detectors. They considered a setting with reliable communication, but their results apply with essentially no change to a setting where communication is unreliable but fair.

Chandra and Toueg observed that by having processes communicate their suspicions, a weak failure detector can be converted to a *strong* failure detector, which satisfies weak accuracy and *strong* completeness (all correct processes eventually permanently suspect every faulty process). We further observe that, under a relatively innocuous assumption, in systems with no bound on the number of faulty processes, strong failure detectors are equivalent to *perfect* failure detectors, which satisfy strong completeness and *strong accuracy*—no process is suspected until it crashes.

These results tell us that we can attain UDC if there is no bound on failures using what are effectively equivalent to perfect failure detectors. Are perfect failure detectors really necessary? We show that in a precise sense they are. Under quite minimal assumptions, a system that attains UDC with no bounds on the number of failures can implement perfect failure detectors.[1] It is interesting to note that Schiper and Sandoz' *Uniform Reliable Multicast* [SS93] is a special case of UDC where the only action of interest

*This work was supported in part by NSF under grant IRI-96-25901.

[1] We remark that our notion of "implement" is stronger than the notion of reduction used in [CT96, CHT96]; see Section 3.

is reliable message delivery. Schiper and Sandoz implement Uniform Reliable Multicast by using the Isis virtual synchrony model [BJ87], which simulates perfect failure detection. Our results support their need to implement it in this way.

What happens if there is a bound on the number of faulty processes? Gopal and Toueg [GT89] show that UDC is achievable with no failure detectors if less than half the processes can be faulty. Here we generalize these results, providing a generalized failure detector that we can show is necessary and sufficient to attain UDC if there are at most t failures, for each possible value of t. The generalized failure detector we consider reports suspicions of the form "at least k processes in a set S of processes are faulty" (although it does not specify which k are the faulty ones). Such generalized failure detectors may be appropriate when the system can be viewed as consisting of a number of components, and all we can say is that some process in a component is faulty, without being able to say which one it is.

The rest of this paper is organized as follows. In Section 2, we provide the necessary background, reviewing the formal model, failure detectors, the formal language, and the definition of UDC. In Section 3, we present our analysis in the case that there is no bound on the number of faulty processes. Our proof techniques may be of independent interest, since they make nontrivial use of the knowledge-theoretic tools of [FHMV95]. Reasoning about the knowledge of the processes in the system—particularly, their knowledge of which other agents are faulty—plays a key role in the analysis. In Section 4, we extend this analysis to the case where there is a known bound t on the number of faulty processes; we also introduce our generalized failure detectors. We conclude in Section 5. Proofs are relegated to the Appendix.

2 Background

2.1 The model

We adopt the familiar model of an asynchronous distributed system. We assume that there is a fixed finite set $\mathsf{Proc} = \{p_1, \ldots, p_n\}$ of processes with no shared global clock. These processes communicate with one another by passing messages over a completely connected network of channels. Processes fail by crashing and do not recover, but otherwise follow their assigned protocols. There is no bound on the number of processes that may crash. Channels are not reliable. A message that is sent is not necessarily received and, even if it is received, there is no bound on message transmission delay. However, channels do not corrupt messages and they are *fair*, in the sense that if the same message is sent from p to q infinitely often and q does not crash, then the message is eventually received by q.

Processes execute *actions* in a totally ordered sequence; corresponding to each action is an *event* which is recorded in the process's history. The events corresponding to p's actions consist of the communication events $send_p(q, \mathsf{msg})$ (p sends message msg to q) and $recv_p(q, \mathsf{msg})$ (p receives msg from q), *internal* events of the form $do_p(\alpha)$ (p executes action α), $init_p(\alpha)$ (p initiates α; see Section 2.4), and the special event $crash_p$, which models the failure of p.

A *history* for process p, denoted h_p, is a sequence of events corresponding to actions performed by process p. A *cut* is a tuple of finite process histories, one for each $p \in \mathsf{Proc}$. A *run* is a function from time (which we take to range over the natural numbers, for simplicity) to cuts. If r is a run, we use $r_p(m)$ to denote p's history in the cut $r(m)$. A pair (r, m) consisting of a run r and a time m is called a *point*. We write $(r, m) \sim_p (r', m')$ if $r_p(m) = r'_p(m')$. We say that a run r' *extends* a point (r, m) if $r'(m') = r(m')$ for all $m' \leq m$. Thus, r' extends (r, m) if r and r' have the same prefix up to time m.

We assume that a run r satisfies the following.

R1. $r(0) = (\langle\,\rangle, \ldots, \langle\,\rangle)$ (that is, at time 0, each process's history is empty);

R2. $r_p(m+1) = r_p(m)$ or $r_p(m+1)$ is the result of appending one event to $r_p(m)$;

R3. if $recv_q(p, \mathsf{msg})$ is in $r_q(m)$, then the corresponding send event $send_p(q, \mathsf{msg})$ is in $r_p(m)$;

R4. if the event $crash_p$ is in $r_p(m)$, then the only events that follow it in $r_p(m)$ are other $crash_p$ events;

R5. if the number of occurrences of $send_p(q, \mathsf{msg})$ in $r_p(m)$ grows unboundedly as m increases, then either the event $crash_q$ appears in $r_q(m)$ for some m or the number of occurrences of $recv_q(p, \mathsf{msg})$ in $r_q(m)$ grows unboundedly as m increases.

When we consider failure detectors, we add further conditions to runs.

A *system* is a set of runs. Systems are typically generated by *protocols* executed in a certain *context*. Formally, a *protocol for process p* is a function from finite histories to actions. A *joint protocol* is a tuple $(P_1, \ldots, P_n)$ consisting of a protocol for each process in Proc. A run r is consistent with a joint protocol P if for each finite prefix $h_i = h'_i \cdot e$ of p_i's history in r, if e is not a receive or crash event, then e is the event corresponding to the action $P_i(h'_i)$. A *context* for us is simply a bound on the number of processes that can fail and a specification of properties of failure detectors (see Section 2.2; see [FHMV95, FHMV97] for a a more general definition of context). In a given context, a joint protocol generates the system consisting of all the runs satisfying R1–R5 and the constraints of the context that are consistent with the protocol. We say that a protocol has a certain property if the system it generates has that property.

2.2 Failure Detectors

Informally, a *failure detector* [CT96] is a per-process oracle that emits suspicions regarding other processes' faultiness. These suspicions, in general, are just that. The fact that a process q is suspected by process p's failure detector does not mean that q is in fact faulty. We then can impose various conditions on the accuracy of a failure detector's suspicions and the completeness of these suspicions, that is, whether a process that is faulty is (eventually) suspected.

Rather than model the failure detector with a special tape, as Chandra and Toueg do [CT96], we model the act of p reading its failure detector by the event $suspect_p(S)$, where $S \subseteq \mathsf{Proc}$. This event should be interpreted as p's failure detector saying that the processes in S are suspected of being faulty. At any point (r, m), define $\mathsf{Suspects}_p(r, m) = S$ if and only if $suspect_p(S)$ is the most recent failure-detector event in $r_p(m)$. $F(r)$ denotes the faulty processes in run r.

Consider the following properties of failure detectors (the first four are from [CT96]):

Strong Accuracy: No process is suspected before it crashes. Formally, if $q \in \mathsf{Suspects}_p(r, m)$, then $crash_q$ is in $r_q(m)$ for all processes p and q and times m.

Weak Accuracy: If there is a correct process, then some correct process is never suspected. Formally, if $F(r) \neq \mathsf{Proc}$ then there is some $q \notin F(r)$ such that $q \notin \mathsf{Suspects}_p(r, m)$, for all processes p and times m.[2]

Strong Completeness: All faulty processes are eventually permanently suspected by all correct processes. Formally, if $q \in F(r)$ and $p \notin F(r)$, then there is a time m such that for all $m' \geq m$, $q \in \mathsf{Suspects}_p(r, m')$.

Weak Completeness: Each faulty process is eventually permanently suspected by some correct process. Formally, if $q \in F(r)$, then there exists some $p \notin F(r)$ and a time m such that for all $m' \geq m$, $q \in \mathsf{Suspects}_p(r, m')$.

Impermanent Strong Completeness: All faulty processes are eventually suspected (but not necessarily permanently) by all correct processes. Formally, if $q \in F(r)$ and $p \notin F(r)$, then $q \in \mathsf{Suspects}_p(r, m)$ for some time m.

Impermanent Weak Completeness: A faulty processes is eventually suspected (but not necessarily permanently) by some correct process. Formally, if $q \in F(r)$, then there is some $p \notin F(r)$ such that $q \in \mathsf{Suspects}_p(r, m)$ for some time m.

[2] The assumption that $F(r) \neq$ Proc does not appear in [CT96], since Chandra and Toueg assume that there always is at least one correct process. We have added it here so as to be able to handle the case that all processes in a run fail.

Chandra and Toueg [CT96] define a *perfect failure detector* as one that satisfies strong completeness and strong accuracy, a *strong failure detector* as one that satisfies strong completeness and weak accuracy, and a *weak failure detector* as one that satisfies weak completeness and weak accuracy. We define an *impermanent-strong failure detector* as one that satisfies impermanent strong completeness and weak accuracy and an *impermanent-weak failure detector* as one that satisfies impermanent weak completeness and weak accuracy. We say that $\mathcal{R}$ is a *has perfect (resp., strong, weak, impermanent-strong, impermanent-weak) failure detectors* if for all $r \in \mathcal{R}$, strong completeness and strong accuracy (resp., strong completeness and weak accuracy; weak completeness and weak accuracy; impermanent strong completeness and weak accuracy; impermanent weak completeness and weak accuracy) hold.

Chandra and Toueg show that, in many cases of interest, we can convert a failure detector satisfying weak completeness to one satisfying strong completeness, while still preserving accuracy properties; the same holds for weak impermanent completeness and strong impermanent completeness. Thus, we do not consider weak (impermanent) completeness further. Note that in the presence of strong accuracy, we can trivially convert a failure detector that satisfies impermanent strong completeness to one that satisfies strong completeness by always outputting the list of all previously suspected processes. As we show in Section 3, under a minimal assumption that should surely be satisfied in practice, if there is no bound on the number of faults, a failure detector that satisfies weak accuracy must also satisfy strong accuracy. Thus, if there is no bound on the number of faults, then there is essentially no difference between impermanent-strong failure detectors and perfect failure detectors.[3]

2.3 The Formal Language

Our language for reasoning about distributed coordination involves time and knowledge. The underlying notion of time is linear (so our language extends linear time temporal logic). We find it useful to be able to reason about the past as well as the future. Formally, we start with primitive propositions and close off under Boolean combinations, $\Box$, and the epistemic operators K_p for each process p.

A formula is true or false relative to a tuple $(\mathcal{R}, r, m)$ consisting of a system $\mathcal{R}$, run $r \in \mathcal{R}$, and time m. We write $(\mathcal{R}, r, m) \models \varphi$ if the formula φ is true at the point (r, m) in system $\mathcal{R}$. Among the primitive propositions in the language are $\mathsf{send}_p(q, \mathsf{msg})$, $\mathsf{recv}_q(p, \mathsf{msg})$, $\mathsf{crash}(p)$, $\mathsf{do}_p(\alpha)$, and $\mathsf{init}_p(\alpha)$. The truth of these primitive propositions is determined by the cut in the obvious way; for example, $\mathsf{send}_p(q, \mathsf{msg})$ is true at point (r, m) precisely when $send_p(q, \mathsf{msg})$ is

[3] In the notation of Chandra and Toueg, impermanent-$\mathcal{S} \cong \mathcal{S} \cong \mathcal{P}$ for $t = n - 1$ failures.

an event in p's history component of (r, m). $\Box\varphi$ holds at a point if it holds from that point on in the run. Thus, $(\mathcal{R}, r, m) \models \Box\varphi$ if and only if $(\mathcal{R}, r, m') \models \varphi$ for all $m' \geq m$. As usual, we define $\Diamond\varphi = \neg\Box\neg\varphi$; thus, $\Diamond$ is the dual of $\Box$. It is easy to see that $(\mathcal{R}, r, m) \models \Diamond\varphi$ if $(\mathcal{R}, r, m') \models \varphi$ for some $m' \geq m$. Finally, $K_p\varphi$ is true if φ is true at all the points that p considers possible, given its current history. Formally, $(\mathcal{R}, r, m) \models K_p\varphi$ if and only if $(\mathcal{R}, r', m') \models \varphi$ for all points $(r', m') \sim_p (r, m)$. We say a formula φ is *valid in system* $\mathcal{R}$, denoted $\mathcal{R} \models \varphi$, if $(\mathcal{R}, r, m) \models \varphi$ for all points (r, m) in $\mathcal{R}$.

In our analysis, we make particular use of *local* and *stable* formulas. A formula φ is *local* to process p in system $\mathcal{R}$ if at every point p knows whether it is true, that is, if $K_p\varphi \vee K_p\neg\varphi$ is valid in $\mathcal{R}$. Thus, all formulas describing a process's local state, for example, $\mathsf{send}_p(q, \mathsf{msg})$, $\mathsf{recv}_q(p, \mathsf{msg})$, $\mathsf{crash}(p)$, and $\mathsf{init}_p(\alpha)$, are local to only that process. It follows from the properties of knowledge that formulas of the form $K_p\varphi$ are also local to p, since $K_p\big(K_p\varphi\big) \vee K_p(\neg K_p\varphi)$ is valid in every system. A *stable* formula is one that, once true, remains true; that is, φ is stable in system $\mathcal{R}$ if $\varphi \Rightarrow \Box\varphi$ is valid in $\mathcal{R}$. All of $\mathsf{send}_p(q, \mathsf{msg})$, $\mathsf{recv}_q(p, \mathsf{msg})$, $\mathsf{crash}(p)$, and $\mathsf{init}_p(\alpha)$ are stable.

2.4 Distributed Coordination

We are interested in modeling distributed coordination of certain actions among the processes in Proc. The actions may be allocating a resource, delivering multicast messages, or committing a transaction; we are not concerned with the specifics. We are also not concerned here with other requirements such as executing actions in a particular order (e.g., total-order multicast) or not executing conflicting actions (e.g., consensus). We are interested only in the eventual, distributed execution of these actions.

Formally, we assume that each process p has a set $\mathcal{A}_p$ of *coordination actions* it may want to perform. We further assume that the sets $\mathcal{A}_p$ and $\mathcal{A}_q$ are disjoint for $p \neq q$. (Think of the actions in $\mathcal{A}_p$ as somehow being tagged by p.) We assume that for each action in $\alpha \in \mathcal{A}_p$, there is a special action of *initiating* α. The corresponding event $init_p(\alpha)$ can appear only in p's history, and can appear at most once in a run. Intuitively, if $init_p(\alpha)$ appears in p's history, we would like all the processes in Proc to perform α. The only question is what requirements we should make of actions performed by processes that crash.

Uniform Distributed Coordination (UDC) of action α occurs if whenever any $p' \in \mathsf{Proc}$ executes $\alpha \in \mathcal{A}_p$, then so eventually does every correct $q \in \mathsf{Proc}$. As well, no process performs $\alpha \in \mathcal{A}_p$ unless p initiates it. Formally, UDC of $\alpha \in \mathcal{A}_p$ holds in a system $\mathcal{R}$ if the following three conditions hold:

DC1. $\mathcal{R} \models \mathsf{init}_p(\alpha) \Rightarrow \Diamond(\mathsf{do}_p(\alpha) \vee \mathsf{crash}(p))$;

DC2. $\mathcal{R} \models \bigwedge_{p',q \in \mathsf{Proc}}\Big(\mathsf{do}_{p'}(\alpha) \Rightarrow \Diamond(\mathsf{do}_q(\alpha) \vee \mathsf{crash}(q))\Big)$;

DC3. $\mathcal{R} \models \bigwedge_{q \in \mathsf{Proc}}\Big(\mathsf{do}_q(\alpha) \Rightarrow \mathsf{init}_p(\alpha)\Big)$.

Non-Uniform Distributed Coordination (nUDC) requires coordination only if the process that performs α is correct. Thus, nUDC of α holds in a system $\mathcal{R}$ if DC1, DC3, and the following hold:

DC4. $\mathcal{R} \models \bigwedge_{p',q \in \mathsf{Proc}}\Big(\mathsf{do}_{p'}(\alpha) \Rightarrow \Diamond(\mathsf{do}_q(\alpha) \vee \mathsf{crash}(q) \vee \mathsf{crash}(p'))\Big)$.

The next propositions show that, unlike UDC, nUDC is easy to attain, and that reliable communication is significant for UDC.

Proposition 2.1: *There is a protocol that attains nUDC, even without failure detectors, in a context where there is no bound on the number of failures.*

Proposition 2.2: *UDC is achievable in a context where communication is reliable and there is no bound on the number of failures.*

Propositions 2.1 and 2.2 highlight some of the differences between UDC, nUDC, and consensus. Unlike consensus, both UDC and nUDC are attainable in asynchronous systems with failures (although UDC needs reliable communication); indeed, they are attainable no matter how many processes may fail. However, as we shall see in the next two sections, things change when we consider UDC in a context with unreliable communication.

3 UDC With No Bound on Failures

We start by showing that UDC is achievable in a context with unreliable communication, provided we have impermanent-strong failure detectors.

Proposition 3.1: *There is a protocol that attains UDC in a context with impermanent-strong failure detectors.*

Chandra and Toueg [CT96] prove a result analogous to Proposition 3.1 for consensus. They show that consensus is achievable with strong failure detectors in a context with at most $n - 1$ failures where communication is reliable. Their algorithm works without change even if we have only impermanent-strong failure detectors and allow n failures. Moreover, their algorithm can be easily modified to deal with unreliable, but fair, communication. Thus, unlike UDC, the reliability of communication has no significant impact on the attainability of consensus.

We next prove what is essentially a converse to Proposition 3.1. Specifically, we show that if processes can perform UDC and there is no bound on the number of possible failures, then they can simulate perfect failure detectors (since under our assumptions, impermanent-strong and perfect failure detectors are equivalent). Thus, among other things, we need to make precise the notions of "simulating a perfect failure detector" and "no bound on process failures".

"Simulating a perfect failure detector" means that there is a function from process p's histories to events of the form $suspect_p(S)$ such that if these events are inserted into the history, the resulting run satisfies the properties of perfect failure detectors (with respect to these new failure-detector events). Formally, for each run $r \in \mathcal{R}$, we construct a run $f(r)$ such that for each process p, we have

- $f(r)_p(0) = (\langle\rangle, \ldots, \langle\rangle)$;
- if $r_p(m+1) = r_p(m)$, then $f(r)_p(2m+1) = f(r)_p(2m+2) = f(r)_p(2m)$;
- if $r_p(m+1) = r_p(m) \cdot e$, then $f(r)_p(2m+1) = f(r)_p(2m) \cdot suspect_p(S)$, where $S = \{q : (\mathcal{R}, r, m) \models K_p(\mathsf{crash}(q))\}$ and $f(r)_p(2m+2) = f(r)_p(2m+1) \cdot e'$, where $e' = e$ unless e is of the form $suspect_p(S)$, in which case e' is $suspect'_p(S)$.

In $f(r)$, process p's history is identical to its history in r except that, at each odd step, p's failure detector reports the processes that p knows have crashed at the corresponding point in $\mathcal{R}$. In addition, we relabel the failure detector events in r so that they are not counted as failure detector events in $f(r)$ (and thus do not need to be considered when we show that $\mathcal{R}'$ has perfect failure detectors). Now define system $\mathcal{R}' = \{f(r) : r \in \mathcal{R}\}$. Proposition 3.2 shows that the failure detector in $\mathcal{R}'$ is perfect, under suitable assumptions.

The notion of simulation implicitly underlying this specific definition for perfect failure detectors is more general that the notion of reduction used in [CT96, CHT96]. For example, in these papers, it is shown that if consensus can be solved by means of a failure detector (and there are at most $t < n/2$ failures), then that failure detector can be transformed to a particular failure detector called $\Diamond\mathcal{W}$ (for *eventually weak*), which satisfies *eventual weak accuracy* and *weak completeness*; see [CT96] for the precise definition. Since consensus can be solved with $\Diamond\mathcal{W}$ failure detectors, these failure detectors are viewed as the weakest failure detectors for consensus. It follows immediately from our simulation that if UDC can be attained using some failure detector, then that failure detector can be transformed in a trivial sense to a perfect failure detector. We just ignore the failure detector altogether and use the transformation above. However, our notion of simulation does not depend on using failure detectors to attain UDC. Thus, it applies in situations where the reductions of [CT96, CHT96] do not.

Consider the following four properties of $\mathcal{R}$, which formalize some standard assumptions.

A1. If there exists a run in $\mathcal{R}$ where all the processes in S crash, and (r, m) is a point in $\mathcal{R}$ such that no process in $\mathsf{Proc} - S$ has crashed, then there is a run r' extending (r, m) such that $F(r') = S$.

A2. If $F(r_1) = F(r_2)$ and $(r_1, m) \sim_q (r_2, m)$ for all $q \notin F(r_1)$, then there are extensions r_1' and r_2' of (r_1, m) and (r_2, m), respectively, such that $(r_1', m') \sim_q (r_2', m')$ for all $m' \geq m$ and all $q \notin F(r_1)$, and all the processes in $F(r_1)$ crash by time $m+1$ in r_1' and r_2'.

A3. The formula $K_q \mathsf{init}_p(\alpha)$ is insensitive to failure by q, where a formula φ local to q is said to be *insensitive to failure by* q if for all runs $r, r' \in \mathcal{R}$, if $r_q'(m') = r_q(m) \cdot crash_q$, then $(\mathcal{R}, r, m) \models \varphi$ iff $(\mathcal{R}, r', m') \models \varphi$.

A4. If φ is a stable formula local to some process p in $\mathcal{R}$ that is insensitive to failure by p and there is some $S \subseteq \mathsf{Proc}$ such that $(\mathcal{R}, r, m) \models \bigwedge_{q \in S} \neg K_q \varphi$, then there exists a point (r', m) such that (a) $r_q'(m) = r_q(m)$ for $q \in S$, (b)for $q \notin S$, there is a prefix h of $r_q(m)$ (not necessarily strict) such that $r_q'(m)$ is either h or $h \cdot crash_q$, and $r_q'(m) = h \cdot crash_q$ only if $crash_q \in r_q(m)$, (c) $(\mathcal{R}, r', m) \models \neg\varphi$.

A5_t. For every $S \subseteq \mathsf{Proc}$ such that $|S| \leq t$, there exists a run $r \in \mathcal{R}$ such that $F(r) = S$.

A1 essentially says that process failures are independent of other events. If it is possible for the processes in S to crash, this may happen at any time in any run. A2 says that the actions of non-faulty processes are not affected by those of faulty processes from whom they hear nothing. Thus, if two points (r, m) and (r', m) are indistinguishable to the correct processes, then there are extensions of these runs that continue to be indistinguishable, in which they hear nothing further from the faulty processes. A3 says that a process cannot learn that p initiated α just by crashing. A4 says, among other things, that if each of the processes in S considers $\neg\varphi$ possible, where φ is a stable failure-insensitive formula local to some process, then there is a point where $\neg\varphi$ is true that all the processes in S simultaneously consider possible. We discuss A4 in more detail below. A5_t captures the assumption that at most t processes may fail. Thus, A5_n captures the assumption that there is no bound on process failures.

In the presence of A1, if there is no bound on the number of failures, strong failure detectors are perfect failure detectors.

Proposition 3.2: *If $\mathcal{R}$ satisfies A1 and A5_{n-1} (or A5_n), then $\mathcal{R}$ satisfies weak accuracy iff $\mathcal{R}$ satisfies strong accuracy.*

It follows from Proposition 3.2 that if $\mathcal{R}$ satisfies A1 and A5_{n-1}, then $\mathcal{R}$ has strong failure detectors iff $\mathcal{R}$ has perfect failure detectors.

A1 and A3 are properties we would expect to hold of all systems generated by protocols in the contexts of interest to us. A2 implicitly assumes that there is no information relevant to the system beyond what is in the processes' states. For example, suppose each process had a message buffer, such that once a message was in q's buffer, then as long as q did not crash, q would eventually receive the message. Consider two runs r_1 and r_2 such that $(r_1, m) \sim_q (r_2, m)$, $F(r_1) = F(r_2)$, and $F(r_1)$ consists of all processes other than q. Moreover, suppose that there is a message msg in q's buffer in (r_1, m), but not in (r_2, m). By A2, there are extensions of r_1' and r_2' of (r_1, m) and (r_2, m) such that all processes other than q crash in round $m+1$ in both r_1' and r_2' and $(r_1', m') \sim_q (r_2', m')$ for all $m' \geq m$. But this is impossible, since q receives msg in r_1' but not in r_2'.

Assumption A4 is perhaps the least standard assumption. A4 assumes that processes are essentially using a *full-information protocol* (FIP) [Coa86, FHMV95] to generate $\mathcal{R}$ and places some restrictions on the information they can get from failure detectors. With an FIP, when a process p sends a message to q, it sends complete information about its state. To see why we need FIPs for A4, suppose that neither p nor q have crashed at (r, m), and at some time q sends a message msg to p', which p' receives. Then p' sends p a message saying $\mathsf{crash}(q) \vee \mathsf{send}_q(p', \mathsf{msg})$, which p receives by time m. Thus, $(\mathcal{R}, r, m) \models K_p(\mathsf{crash}(q) \vee \mathsf{send}_q(p', \mathsf{msg})) \wedge \neg K_p(\mathsf{crash}(q)) \wedge \neg K_p(\mathsf{send}_q(p', \mathsf{msg}))$. Then there cannot exist a point (r', m') such that (a) $r_p'(m') = r_p(m)$, (b) $r_q'(m')$ is a prefix of $r_q(m)$, and (c) $(\mathcal{R}, r', m') \models \neg \mathsf{send}_q(p', \mathsf{msg})$. If that were the case, then necessarily $(\mathcal{R}, r', m') \models \mathsf{crash}(q)$, because $K_p(\mathsf{crash}(q) \vee \mathsf{send}_q(p', \mathsf{msg}))$ holds at $(\mathcal{R}, r, m)$, violating the assumption that $r_q'(m')$ is a prefix of $r_q(m)$.

This example violates A4 because p' did not tell p all it knew, which is precisely what cannot happen with a full-information protocol. Assuming that $\mathcal{R}$ is generated by an FIP, under reasonable assumptions about failure detectors, $\mathcal{R}$ should satisfy A4. To see why, given (r, m), we can construct run r' as follows. First note that $(\mathcal{R}, r, 0) \models \neg\varphi$, for φ stable and local to p, for otherwise φ would be true at all points in $\mathcal{R}$ and so would $K_p\varphi$. So let m_p be the first time in r where φ becomes true. Let $S \subseteq \mathsf{Proc}$ be the processes that do not know φ at (r, m). If processes are following a full-information protocol, there can be no chain of messages from p to a process $q \in S$ between times m_p and m in r, for if there were, q would know φ at (r, m).[4] For each process $q \in \mathsf{Proc}$, let m_q be the least time at or before m at which there is a message chain from p to q in r between m_p and m_q, if there is such a time; otherwise, we take $m_q = m + 1$. Note that for $q \in S$, we have $m_q = m + 1$. We then construct r' so that $r_q'(m') = r_q(m')$ for $m' \leq m_q - 1$; if q does not crash in r between times m_q and m inclusive, then $r_q(m') = r_q(m_q - 1)$ for $m' \geq m_q$; otherwise, $r_q(m') = r_q(m_q - 1) \cdot \mathit{crash}_q$ for $m' \geq m_q$. By construction, we have $r_q(m) = r_q'(m)$ for $q \in S$. For $q' \notin S$, we have that $r_{q'}(m)$ is either $r_{q'}(m_q - 1)$ or $r_{q'}(m_q - 1) \cdot \mathit{crash}_{q'}$. The reason we need to add $\mathit{crash}_{q'}$ is that the failure detector of some process $q \in S$ might report that q' fails in r. Since $r_q(m) = r_q'(m)$, if q's failure detector is accurate, it must be the case that q' also fails in r'. Assuming the failure detectors are "reasonable" (in particular, cannnot say something like "either p has crashed or p received the message"), then r' should indeed be a run in $\mathcal{R}$. Note that if $(\mathcal{R}, r, m) \models \neg\mathsf{crash}(p)$ then $r_p'(m) = r_p(m_p)$; otherwise, either $r_p'(m) = r_p(m_p - 1)$ or $r_p'(m) = r_p(m_p - 1) \cdot \mathit{crash}_p$. Since φ is insenstive to failure by p, we have that $(\mathcal{R}, r', m) \models \neg\varphi$. Thus, (r', m) satisfies the requirements of A4.

Theorem 3.4 shows that if $\mathcal{R}$ attains UDC and satisfies A1–A4 and A5_n (or A5_{n-1}) then the system $\mathcal{R}'$ has perfect failure detectors. In light of this discussion, our result can be viewed as saying that under some relatively innocuous assumptions (A1–A3), if there is no bound on the number of failures (A5_n) and the processes are telling each other as much as they can (A4), then attaining UDC is tantamount to having perfect failure detectors. Before proving that, we provide a characterization of the facts that must be known by a process before it can perform a coordination action α. Specifically, a process must know that if there are any correct processes at all, then one of these knows that α has been initiated. Although the proposition is phrased in terms of the knowledge of the initiating process p (and this is all we need for our proofs), a straightforward extension of the argument shows that it holds for every process p' that performs α.

Proposition 3.3: *If $\mathcal{R}$ satisfies A1, A2, and A4, then*
$$\mathcal{R} \models \bigwedge_{p \in \mathsf{Proc}} \bigwedge_{\alpha \in \mathcal{A}_p}$$
$$K_p\Big(\mathsf{init}_p(\alpha) \wedge \bigwedge_{q \in \mathsf{Proc}} \Diamond(K_q \mathsf{init}_p(\alpha) \vee \mathsf{crash}(q))\Big)$$
$$\Rightarrow K_p\Bigg(\bigvee_{q \in \mathsf{Proc}} \Box\neg\mathsf{crash}(q) \Rightarrow \bigvee_{q \in \mathsf{Proc}}\Big(K_q \mathsf{init}_p(\alpha) \wedge \Box\neg\mathsf{crash}(q)\Big)\Bigg).$$

Note that the clause $\bigvee_{q \in \mathsf{Proc}} \Box\neg\mathsf{crash}(q)$ in the antecedent of the right-hand side of the the formula in Proposition 3.3 is trivially true if we assume A5_t for $t \leq n - 1$. We are now ready to state our theorem.

[4] There is a message chain from p to q between m_p and $m > m_p$ if there is a sequence of messages $\mathrm{msg}_1, \ldots, \mathrm{msg}_k$ and processes $p_1, \ldots, p_{k+1}$ such that (a) msg_i is sent by p_i to p_{i+1} and is received, (b) p_{i+1} sends msg_{k+1} after receiving msg_k, (c) $p = p_1$, (d) $q = p_{k+1}$, (e) p sends msg_1 at or after m_p, and (f) q receives msg_{k+1} at or before m. If the processes follow a full-information protocol, then p_{i+1} knows everything that p_i knows when it receives msg_i.

Theorem 3.4: *Suppose $\mathcal{R}$ is the system generated by a protocol that attains UDC, $\mathcal{R}$ satisfies A1–A4 and $A5_n$ (or $A5_{n-1}$) and in each run $r \in \mathcal{R}$, if p is correct in r, then p initiates actions infinitely often in r (i.e., infinitely many events of the form $init_p(\alpha)$ appear in p's history in r). Then the system $\mathcal{R}'$ generated as above is one with perfect failure detectors.*

4 Generalized Failure Detectors

Theorem 3.4 shows that if there is no bound on the number of failures (or a bound of $n-1$), then UDC essentially requires perfect failure detectors. On the other hand, as Gopal and Toueg [GT89] show, if there are fewer than $n/2$ failures, then UDC is attainable without failure detectors. We now generalize both of these results, characterizing the type of failure detector needed to attain UDC if there is a bound of t on the number of possible failures, for all values of t.

A *generalized failure detector* reports that (it suspects that) at least k processes in a set S are faulty. As we discussed earlier, such generalized failure detectors are appropriate when processes can observe faulty behavior in some component(s) without being able to tell which processes in the component are actually faulty. We model such generalized suspicions by using events of the form $suspect_p(S, k)$. We are interested in generalized failure detectors that give useful information. Of course, what is "useful" may depend on the application. Given a system $\mathcal{R}$ and an upper bound of t on the number of failures that may occur in a run of $\mathcal{R}$, we say that $suspect_p(S,k)$ is a *t-useful failure-detector event for r* if (a) $F(r) \subseteq S$ and (b) $n - |S| > \min(t, n-1) - k$ (or, equivalently, $k > |S| - n + \min(t, n-1)$). Note that if p learns at the point (r, m) that there are k faulty processes in S and $n - |S| > \min(t, n-1) - k$, then p can conclude that, if there are any correct processes at all in r, then one of the processes in $\mathsf{Proc} - S$ is correct at (r, m) (although it may not know which one). Just knowing that some processes in a set are correct is not in general useful. For example, if $t < n$, then all processes know that at least $n-t$ processes in Proc are correct. As we shall see, what makes this fact useful is that $F(r) \subseteq S$, even though p may not know this.

A generalized failure detector in $\mathcal{R}$ is *t-useful* if for all $r \in \mathcal{R}$ and processes p, we have:

Generalized Strong Accuracy: if $suspect_p(S,k)$ is in $r_p(m)$ then there is a subset $S' \subseteq S$ such that $|S'| = k$ and for all $q \in S'$, we have that $crash_q$ is in $r_q(m)$.

Generalized Impermanent Strong Completeness: if p is correct, then there is a t-useful failure-detector event for r in $r_p(m)$, for some m.

Note that it is trivial to construct a t-useful failure detector in a context with at most t failures if $t < n/2$: repeatedly output $(S, 0)$ for every $S \subset \mathsf{Proc}$ with $|S| = t$. Suspecting no processes in any subset S trivially satisfies generalized strong accuracy, and in every run r at least one t-sized subset of Proc must contain $F(r)$. Whenever $F(r) \subseteq S$, then $(S, 0)$ is a t-useful failure-detector event.

Also note that if $suspect_p(S,k)$ is an $(n-1)$-useful or n-useful failure-detector event, then we must have $|S| = k$, since the only way to have $k > |S| - 1$ is to have $k = |S|$. Thus, we can easily convert an n-useful or $(n-1)$-useful generalized failure detector to a perfect failure detector, by just reporting events of the form $suspect_p(S)$ every time the generalized failure detector reports $suspect_p(S,k)$ with $|S| = k$. Conversely, we can easily convert a perfect failure detector to an n-useful (and hence $(n-1)$-useful) failure detector. Given a history for process p, we simply replace each event $suspect_p(S)$ by the event $suspect_p(S', k)$, where S' is the union of S together with all the sets that appeared in prior failure detector events in the history, and $k = |S'|$. It is easy to see that this gives us a useful failure detector. Thus, the following result generalizes Proposition 3.1 and Gopal and Toueg's result.

Proposition 4.1: *There is a protocol that attains UDC in a context with a bound of t on the number of failures and with t-useful generalized failure detectors.*

We want a converse to Proposition 4.1 that generalizes Theorem 3.4. We show that if processes can perform UDC in a context with a bound t on the number of failures, then t-useful generalized failure detectors can be simulated in that context.

Given system $\mathcal{R}$, construct system $\mathcal{R}'$ as follows. Fix an order $S_0, \ldots, S_{2^n - 1}$ of the subsets of Proc. For each run $r \in \mathcal{R}$, let $\mathcal{R}' = \{f(r) : r \in \mathcal{R}\}$ where $f(r)$ is constructed exactly as before, with just one minor change:

- if $r_p(m+1) = r_p(m) \cdot e$, then $f(r)_p(2m+1) = f(r)_p(2m) \cdot suspect_p(S_l, k)$, where l is the length of the history $r_p(m+1) \bmod 2^n$ and $k = \max |\{k' : (\mathcal{R}, r, m) \models K_p(k' \text{ processes in } S \text{ have crashed})\}|$ and $f(r)_p(2m+2) = f(r)_p(2m+1) \cdot e'$, where e' is as before.

In the full paper we prove the following.

Theorem 4.2: *Suppose $\mathcal{R}$ is the system generated by a protocol that attains UDC in a context with at most t failures, $\mathcal{R}$ satisfies A1–A4 and $A5_t$, and in each run $r \in \mathcal{R}$, if p is correct in r then p initiates actions infinitely often in r. Then the system $\mathcal{R}'$ generated as above is one with t-useful generalized failure detectors.*

5 Conclusions

We have shown that the problem of Uniform Distributed Coordination in asynchronous systems varies in its complexity both with communication guarantees and with the number of failures that must be

		$0 < t < n/2$	$n/2 \leq t < n-1$	$n-1 \leq t \leq n$
Reliable channels	UDC	no FD	no FD	no FD
	consensus	$\Diamond\mathcal{W}$ †	$\mathcal{S}$trong	$\mathcal{P}$erfect †
Unreliable channels	UDC	no FD	t-useful †	$\mathcal{P}$erfect †
	consensus	$\Diamond\mathcal{W}$ †	$\mathcal{S}$trong	$\mathcal{P}$erfect †

Table 1: UDC versus consensus by failure-detector type; † indicates optimality.

tolerated (see Table 1). Unlike consensus UDC is sensitive to communication guarantees. This is significant since UDC is likely the only acceptable reliability guarantee for many wide-area applications, precisely where reliable communication cannot be assumed.

Note that we have completely characterized the type of failure detector required to attain UDC for all values of t. For consensus, it is known that $\Diamond\mathcal{W}$ is necessary and sufficient if $t < n/2$. (Recall that in this case no failure detectors at all are necessary to attain UDC.) While strong (actually, impermanent-strong) failure detectors suffice for consensus for $n/2 \leq t < n$, there is no characterization of exactly the type of failure detector that is required. The notion of t-useful failure detectors defined here may prove useful in that regard. We leave exploring this issue for future work.

Acknowledgments: We thank Marcos Aguilera, Boris Deianov, and Sam Toueg for their perceptive coments, particularly with regard to assumption A4.

References

[ACT97] M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. In *Proceedings of the 11th International Workshop on Distributed Algorithms*, pages 126–140. Springer-Verlag, 1997. A full version is also available as Technical Report 97-1631, Department of Computer Science, Cornell University, 1997.

[BJ87] K. Birman and T. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *11th Symposium on Operating System Principles*, pages 123–138, 1987.

[CHT96] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 46:685–722, 1996.

[Coa86] B. Coan. A communication-efficient canonical form for fault-tolerant distributed protocols. In *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pages 63–72, 1986.

[CT96] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225–267, 1996.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, Mass., 1995.

[FHMV97] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, 1985.

[GT89] A. Gopal and S. Toueg. Reliable Broadcast in Synchronous and Asynchronous Environments. In *3rd WDAG. Springer Verlag (LNCS 392)*, pages 110–123, 1989.

[SS93] A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *Proceedings of the IEEE 13th ICDCS*, 1993.

Appendix – Proofs of Propositions and Theorems

Proposition 2.1: *There is a protocol that attains nUDC, even without failure detectors, in a context where there is no bound on the number of failures.*

Proof: We just sketch the protocol here, since it is so simple. Whenever a process p wants to attain nUDC of action α (i.e., if $init_p(\alpha)$ is in p's history) p goes into a special $\mathsf{nUDC}(\alpha)$ state. If a process is in an $\mathsf{nUDC}(\alpha)$ state, it performs α and sends an α-message repeatedly to all other processes (which, intuitively, tells them to perform α). If a process receives an α-message telling it to perform α, it goes into an $\mathsf{nUDC}(\alpha)$ state, if it has not already done so. It is easy to see that this protocol attains nUDC. [5] ∎

[5] This protocol, like most of the others we present in this paper, does not have any mechanism for termination. Processes keep sending messages forever. Since message communication is unreliable, it is not hard to show that there is in fact no protocol that attains nUDC and terminates. We can deal with this problem by adding a *heartbeat* mechanism [ACT97], but this issue is beyond the scope of this paper.

Proposition 2.2: *UDC is achievable in a context where communication is reliable and there is no bound on the number of failures.*

Proof: We proceed just as in the proof of Proposition 2.1, except that before performing the action α, a process simply sends a message to all other processes telling them to perform α and inform all other processes if they have not already done so. More precisely, if $init_p(\alpha)$ is in p's history, p goes into a special $\mathsf{UDC}(\alpha)$ state. If a process is in a $\mathsf{UDC}(\alpha)$ state, it sends an α-message to all processes and then performs α. If a process receives an α message, it goes into a UDC-state if it has not already done so. Since a process q performs α only after sending out an α-message to all processes, if communication is reliable, q knows that all other correct processes will receive the message, and thus also perform α, even if q crashes. ∎

Proposition 3.1: *There is a protocol that attains UDC in a context with impermanent-strong failure detectors.*

Proof: The proof is similar in spirit to that of Proposition 2.1. Whenever a process wants to attain UDC of action α, it goes into a special $\mathsf{UDC}(\alpha)$ state. If a process p is in a $\mathsf{UDC}(\alpha)$ state, it sends an α-message repeatedly to all other processes telling them to perform α and reads its failure detector repeatedly. Process p performs α if, for every process q, p receives an acknowledgment from q to its α-message or p's failure detector says that q is faulty. However, p continues to send α-messages even after performing α, until it has received an acknowledgment for *all* processes (which may never happen).[6] Every time a process q receives an α-message from p, q sends an acknowledgment to p; it also goes into a $\mathsf{UDC}(\alpha)$ state if it has not already done so.

To show that this protocol attains UDC, it suffices to show that, in every run, (1) if a process p is in a $\mathsf{UDC}(\alpha)$ state, then p will eventually perform α or crash and (2) if p performs α then every other correct process performs α. To see that (1) holds, suppose that p is in a $\mathsf{UDC}(\alpha)$ state in run r. Then p repeatedly sends an α-message in r, so if p is correct, then eventually every correct process q will get p's α-message and acknowledge it. Since a correct process sends an acknowledgment for each α-message it receives, R5 ensures that p will eventually get an acknowledgment from every correct process. Since p has a impermanent-strong failure detector, it will also eventually suspect every faulty process. Thus, it will perform α, according to the protocol above.

To see that (2) holds, first observe that since p has a weakly accurate failure detector, there is some correct process, say q^*, that p never suspects. Thus, if p performs α, it must receive an acknowledgment from q^* to its α-message. Hence, q^* goes into a $\mathsf{UDC}(\alpha)$ state and, by the previous argument, also performs α. Since q^* is correct, all correct processes eventually receive an α-message from q^* and so perform α. ∎

[6] If p has a strongly accurate failure detector rather than just a weakly accurate failure detector, it can actually stop sending messages after performing α. This follows from the proof of Proposition 3.1.

Proposition 3.2: *If $\mathcal{R}$ satisfies A1 and A5$_{n-1}$ (or A5$_n$), then $\mathcal{R}$ satisfies weak accuracy iff $\mathcal{R}$ satisfies strong accuracy.*

Proof: Let $\mathcal{R}$ satisfy A1, A5$_{n-1}$, and weak accuracy. If $\mathcal{R}$ does not satisfy strong accuracy, then there is a point (r,m) such that $suspect_p(S,k) \in r_p(m)$, $q \in S$, and q has not failed in S. Let $S' = \mathsf{Proc} - \{q\}$. By A5$_{n-1}$, there is a run r' where all the processes in S' fail. Thus, by A1, there is a run r'' extending (r,m) such that all the processes in S' fail in r''. Thus, q is the only correct process in r''. By weak accuracy, we must have that q is never suspected as faulty in r'', contradicting the assumption that it is in fact suspected by p. ∎

Proposition 3.3: *If $\mathcal{R}$ satisfies A1, A2, and A4, then*
$\mathcal{R} \models \bigwedge_{p\in\mathsf{Proc}} \bigwedge_{\alpha\in\mathcal{A}_p}$

$$K_p\Big(\mathsf{init}_p(\alpha) \wedge \bigwedge_{q\in\mathsf{Proc}} \Diamond(K_q\mathsf{init}_p(\alpha) \vee \mathsf{crash}(q))\Big)$$
$$\Rightarrow K_p\Bigg(\bigvee_{q\in\mathsf{Proc}} \Box\neg\mathsf{crash}(q) \Rightarrow \bigvee_{q\in\mathsf{Proc}}\Big(K_q\mathsf{init}_p(\alpha) \wedge \Box\neg\mathsf{crash}(q)\Big)\Bigg).$$

Proof: Suppose, by way of contradiction, that

$$\begin{aligned}&(\mathcal{R},r,m) \models \\ &K_p\Big(\mathsf{init}_p(\alpha) \wedge \bigwedge_{q\in\mathsf{Proc}} \Diamond(K_q\mathsf{init}_p(\alpha) \vee \mathsf{crash}(q))\Big) \\ &\wedge \neg K_p\Bigg(\bigvee_{q\in\mathsf{Proc}} \Box\neg\mathsf{crash}(q) \\ &\qquad \Rightarrow \bigvee_{q\in\mathsf{Proc}}\Big(K_q\mathsf{init}_p(\alpha) \wedge \Box\neg\mathsf{crash}(q)\Big)\Bigg).\end{aligned} \tag{1}$$

Then there must be a point $(r^1,m') \sim_p (r,m)$ such that

$$(\mathcal{R},r^1,m') \models \mathsf{init}_p(\alpha) \wedge \bigvee_{q\in\mathsf{Proc}} \Box\neg\mathsf{crash}(q) \wedge \bigwedge_{q\in\mathsf{Proc}}\Big(\Box\neg\mathsf{crash}(q) \Rightarrow \neg K_q\mathsf{init}_p(\alpha)\Big).$$

We have $(\mathcal{R},r^1,m') \models \bigwedge_{q\notin F(r^1)} \neg K_q\mathsf{init}_p(\alpha)$. Since p knows that it initiated α at (r^1,m'), we must have $p \in F(r^1)$. Moreover, $F(r^1) \neq \mathsf{Proc}$.

By A4, there exists a point (r^2,m') such that $(r^2,m') \sim_q (r^1,m')$ for $q \in \mathsf{Proc}-F(r^1)$ and $(\mathcal{R},r^2,m') \models \neg\mathsf{init}_p(\alpha)$. By A1, there exists a run r^3 extending (r^2,m') such that $F(r^3) = F(r^1)$. Since r^3 extends

(r^2, m'), we must have $r^1_q(m') = r^3_q(m')$ for all $q \in \mathsf{Proc} - F(r^1)$. By A2, there exist runs r^4 and r^5 extending r^1 and r^3, respectively, such that $r^4_q(m'') = r^5_q(m'')$ for $m'' \geq m'$. Moreover, all the processes in $F(r^1)$ (and, in particular, p) crash by time $m' + 1$. Thus, $\mathsf{init}_p(\alpha)$ is not in p's history in r^5, which means that $(\mathcal{R}, r^5, m') \models \bigwedge_{q \in \mathsf{Proc}-F(r^1)} \Box\neg K_q \mathsf{init}_p(\alpha)$. Since r^4 and r^5 are indistinguishable to such q from m' onward, we have $(\mathcal{R}, r^4, m') \models \bigwedge_{q \in \mathsf{Proc}-F(r^1)} \Box\neg K_q \mathsf{init}_p(\alpha)$. Since $(r, m) \sim_p (r^1, m')$ and r^4 extends (r^1, m'), we must have $(r, m) \sim_p (r^4, m')$. Hence, we have $(\mathcal{R}, r, m) \models \neg K_p \Big(\Diamond(K_q \mathsf{init}_p(\alpha) \vee \mathsf{crash}(q))\Big)$ for $q \in \mathsf{Proc} - F(r^1)$. This gives the desired contradiction to (1). ∎

Theorem 3.4: *Suppose $\mathcal{R}$ is the system generated by a protocol that attains UDC, $\mathcal{R}$ satisfies A1–A4 and $A5_n$ (or $A5_{n-1}$) and in each run $r \in \mathcal{R}$, if p is correct in r, then p initiates actions infinitely often in r (i.e., infinitely many events of the form $init_p(\alpha)$ appear in p's history in r). Then the system $\mathcal{R}'$ generated as above is one with perfect failure detectors.*

Proof: It is immediate from the construction that p crashes in (r, m) iff p crashes in $(f(r), 2m)$. It easily follows that p's failure detector satisfies strong accuracy. To show that it satisfies strong completeness, suppose that p is correct and q fails in run $f(r) \in \mathcal{R}'$ and hence also in run $r \in \mathcal{R}$. Since p initiates actions infinitely often in $\mathcal{R}$, there must be some action α initiated by p in $f(r)$ after q has failed. Since $\mathcal{R}$ satisfies UDC, by DC1, p must eventually perform α in run r, say at time m. Moreover, by DC2, p knows that, for each process q' (and, in particular, q), q' must eventually either crash or must perform α. Using DC3, it easily follows that we must have $(\mathcal{R}, r, m) \models K_p\Big(\Diamond\Big(\bigwedge_{q' \in \mathsf{Proc}} K_{q'} \mathsf{init}_p(\alpha) \vee \mathsf{crash}(q)\Big)\Big)$. Since $\mathcal{R}$ satisfies A1, A2, and A4 by assumption, it follows from Proposition 3.3 that

$$(\mathcal{R}, r, m) \models K_p\Big(\bigvee_{q' \in \mathsf{Proc}} \Box\neg\mathsf{crash}(q') \Rightarrow \bigvee_{q' \in \mathsf{Proc}}\Big(K_{q'} \mathsf{init}_p(\alpha) \wedge \Box\neg\mathsf{crash}(q')\Big)\Big)\ . \tag{2}$$

Suppose that $(\mathcal{R}, r, m) \models \Box\neg K_p \mathsf{crash}(q)$. Since q crashes in r before p initiates α, we also have $(\mathcal{R}, r, m) \models \neg K_q \mathsf{init}_p(\alpha)$. Thus, there must exist a point $(r^1, m') \sim_p (r, m)$ such that $(\mathcal{R}, r^1, m') \models \neg\mathsf{crash}(q) \wedge \neg K_p K_q \mathsf{init}_p(\alpha)$. Since $K_q \mathsf{init}_p(\alpha)$ is stable, local to q, and (by A3) insensitive to failures by q, by A4, there must exist a point $(r^2, m') \sim_p (r^1, m')$ such that $r^2_q(m')$ is a prefix of $r^1_q(m')$ and $(\mathcal{R}, r^2, m') \models \neg K_q \mathsf{init}_p(\alpha)$. Since $r^2_q(m')$ is a prefix of $r^1_q(m')$, we also have $(\mathcal{R}, r^2, m') \models \neg\mathsf{crash}(q)$. This means $(r^2, m') \sim_p (r, m)$ and $(\mathcal{R}, r^2, m') \models \neg\mathsf{crash}(q) \wedge \neg K_q \mathsf{init}_p(\alpha)$.

By $A5_{n-1}$ (or the stronger $A5_n$) and A1, there is a run r^3 extending (r^2, m') such that all processes except q fail in r^3. Of course, $(r^3, m') \sim_p (r, m)$ and

$$(\mathcal{R}, r^3, m') \models \bigwedge_{q' \in \mathsf{Proc}-\{q\}} \Diamond\mathsf{crash}(q') \wedge \neg K_q \mathsf{init}_p(\alpha) \wedge \Box\neg\mathsf{crash}(q)\ .$$

But this contradicts (2). ∎

Proposition 4.1: *There is a protocol that attains UDC in a context with a bound of t on the number of failures and with t-useful generalized failure detectors.*

Proof: Whenever a process wants to attain UDC of action α, it goes into a special $\mathsf{UDC}(\alpha)$ state. If a process p is in a $\mathsf{UDC}(\alpha)$ state, it sends an α-message repeatedly to all other processes telling them to perform α and reads its failure detector repeatedly. Process p performs α at time m if, by time m, (a) its failure detector reports $suspect_p(S, k)$, (b) it has received messages from all the processes in $\mathsf{Proc} - S$ acknowledging α, and (c) $n - |S| > \min(t, n-1) - k$. Process p continues to send α-messages to $q \in S$ until it either receives an acknowledgment from q or knows q to be faulty. A process that receives an α-message from p sends an acknowledgment to p and goes into a $\mathsf{UDC}(\alpha)$ state if it has not already done so.

To show that this protocol attains UDC, again it suffices to show that, in every run, (1) if a process p is in a $\mathsf{UDC}(\alpha)$ state, then p will eventually perform α or crash and (2) if p performs α then every other correct process performs α. For (1), suppose that p is in a $\mathsf{UDC}(\alpha)$ state in run r. Then p repeatedly sends an α-message in r, so if p is correct, then eventually every correct process q will get it and acknowledge it. Since a process q acknowledges p's α-message each time it gets it, by R5, p will eventually get an acknowledgment from every correct process. Since p has a t-useful failure detector, if it is correct in r, there will be a t-useful failure-detector event say $suspect_p(S, k)$, in $r_p(m)$ for some $m > m_r$. Since p eventually gets acknowledgments from all the processes in $\mathsf{Proc} - S$ (since these, at least, are correct in r), it will eventually perform α, according to the algorithm.

To see that (2) holds, the arguments for (1) show that if p performs α as a result of the failure-detector event $suspect_p(S, k)$, all the processes in $\mathsf{Proc} - S$ have received an α message (and hence are in a $\mathsf{UDC}(\alpha)$ state) and $\mathsf{Proc} - S$ contains at least one correct process, say q, if there are any correct processes in r. Since q continues to send α-messages to all processes, all the correct processes in r will eventually be in a $\mathsf{UDC}(\alpha)$ state. It then follows from (1) that all the correct processes will perform α. ∎