Round-by-Round Fault Detectors:
Unifying Synchrony and Asynchrony*

(Extended Abstract)

Eli Gafni
(eli@cs.ucla.edu)

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
U.S.A.

Abstract

This paper presents a new family of models of distributed-computation which combines features from synchronous, asynchronous, and failure-detector-augmented systems. Like synchronous systems, computation in this family of models evolve in rounds, and communication missed at a round is lost. Unlike synchronous systems, information that is missed at a round does not necessarily imply a real process failure. The features of a specific model is captured in an abstract module called the round-by-round fault detector. The abstraction of system features into such a module facilitates the comparison of different systems, by contrasting their associated fault detectors. We show that this family of models unifies the study of synchrony, asynchrony, message-passing and shared memory. We believe that a study based on round-by-round fault-detectors unifies the study of distributed computing and will lead to new results and insights.

1 Introduction

For many years, researchers studying synchronous message-passing systems have considered algorithms composed of rounds of computation. In each round, a process sends a message to the others and then waits to receive messages from the other processes. The synchronous nature of the system ensures that, by the end of the round, each process receives all messages sent to it in that round by correct processes. In the parlance of Elrad and Frances [1] then, each round of a synchronous system is a communication-closed-layer.

Asynchronous message-passing systems do not provide this communication closedness. However, round-based algorithms for \( f \)-resilient asynchronous systems in which communication-closedness is enforced by discarding messages which are late, and buffering messages which are early, have been developed in the context of the asynchronous Byzantine-Agreement problem [2, 3]. In such algorithms, a process executing a round waits until it has received at least \( n - f \) messages for that round. The bound on the number of failures ensures that this will not block the algorithm. However, it was not clear to researchers whether round-based asynchronous systems are equivalent to the ones in which late messages are not discarded [3].

Researchers, belatedly, turned their attention
again to round-based asynchronous systems in [4]. There, studying the topological properties of system, they referred to round-based systems as iterated systems. The motivation for that name was the fact that the topological structure induced by round-based models, is an iteration of the topological structure induced by a single round. One of the contributions in [4] was the realization for the first time that there is a nicely structured iterated model that is equivalent to shared-memory.

The new idea in this paper is to study round-based models and abstract away the implementation of the communication exchange between processes, be it shared-memory, message-passing or any other mechanism. The properties of the communication mechanisms and system guarantees (such as resiliency), are captured by a module called, round-by-round fault-detect (RRFD).

In round $r$, a process emits its “message” for the round. Each process $p_i$, from the set of processes $S$, communicates with the RRFD and for each process $p_j \in S$ waits until it receives the message emitted by $p_i$ at round $r$, or the RRFD instructs it not to wait for such a message, by indicating that $p_j$ is faulty. When for each process $p_j$, process $p_i$ has either received $p_j$’s message, or it got an indication that $p_j$ is faulty, it can then proceed to the next round.

As with failure detectors considered elsewhere in the literature [5, 6, 7, 8], RRFD’s are unreliable – they may indicate $p_j$ at round $r$ as faulty to some processes and deliver $p_j$’s message to others, as well as indicate $p_j$ to be faulty in one round, only to deliver a message from it in the next round.

Thus, an RRFD system evolves in rounds. In each round each process emits data to all other processors. Each process $p_i$ receives from the system the data emitted by some subset $S(i, r)$ of the processes, and a set of processes $D(i, r)$. The system guarantees that $S(i, r) \cup D(i, r) = S$. Process $p_i$ proceeds to the next round and computes new messages to emit, based on the data it received, and the set $D(i, r)$. Different RRFD systems differ in the predicates over the sets $D(i, r)$, $i = 1, \ldots, n$, $r = 0, 1, \ldots$, that they guarantee. Although it is always the case that $D(i, r) \neq S$ (if one interprets $D(i, r)$ as a set of “late” processes, not all processes can be late), we do not preclude $p_i \in D(i, r)$, since $p_i$ may be late to round $r$ and “learn” that from the RRFD. Such a process, though, may “know” the message it sent through its local state at the beginning of the round.

Using this notation, an abstract algorithm using an RRFD functions as follows (for $p_i$):

$$
\begin{align*}
  r &:= 1 \\
  \text{forever do} \\
  &\quad \text{compute messages } m_{i, r} \text{ for round } r \\
  &\quad \text{emit } m_{i, r} \\
  &\quad \text{(wait until)} \\
  &\quad \forall p_j \in S \quad \text{received } m_{j, r} \text{ or} \\
  &\quad \quad p_j \in D(i, r) \\
  r &:= r + 1 \\
  \text{end}
\end{align*}
$$

An RRFD system satisfying predicate $P$ solves a task $T$ if there exist an emit-receive format algorithm such that, for any $D(i, r)$ family of sets ($p_i \in S$, $r > 0$) that satisfies $P$, if processes start with inputs from $T$, then eventually, after enough rounds, processes commit to outputs that satisfy $T$’s input/output requirements.

This way of viewing a fault detector contrasts with earlier research with failure detectors [5, 6, 7, 8]. That research considered a fixed (asynchronous) system in which failures were “unannounced” and difficult to detect. A failure detector is used to augment these systems by “announcing” (perhaps unreliably) which processes were faulty. That work viewed a failure detector as a “helpful” entity. In contrast, this paper defines a system based on the round-by-round fault detector to which it corresponds. There is no notion of “augmenting” a system by a failure detector, but rather it is an integral part of the system. It is for this reason the fault-detector may be considered in fact to be an adversary. The more freedom the RRFD has to present different sets of faulty processes, the more power it has and the harder it will be to solve problems in the corresponding system.

The remainder of the paper is organized as follows. Section 2 presents many traditional systems in the RRFD framework. Section 3 shows how RRFD’s can be used to study traditional problems in distributed computing. It proposes an RRFD system that is equivalent to a system that has an access k-set agreement object. Section 4 shows how RRFD’s can be used to relate synchronous and asynchronous systems by proving a theorem.
that relates the solvability of problems in an asynchronous system to the existence of a bounded solution in a synchronous system (under certain failure bounds). A corollary of this result shows how asynchronous impossibility results [9, 10, 11, 12] can be used to give direct proofs of synchronous lower bounds [24, 13]. Section 5 shows how the semi-synchronous model of Dolev, Dwork, and Stockmeyer [14] can be understood in terms of RRFD's and uses this to solve an open problem in that model. Related work is considered in Section 6, and concluding remarks appear in Section 7.

2 Examples of RRFD systems

This section shows how a variety of traditional systems may be thought of in terms of round-by-round fault detectors.

Let $P_A$ be the predicate defining an RRFD system $A$, and let $P_B$ define an RRFD system $B$ over the same number of processes, then, we say that $A$ is a submodel of $B$ iff $P_A \Rightarrow P_B$. Obviously, if $A$ is a submodel of $B$ then $A$ implements $B$. The contrary does not hold. An RRFD system $A$ may implement an RRFD system $B$ and $A$ is not a submodel of $B$.

This paper proposes to investigate systems by finding their RRFD counterparts. The RRFD counterparts, being part of the same family, bring forth the commonality and the difference between the systems. There are many possibilities of choosing an RRFD counterpart of a given system. There are some questions one may ask about the counterpart of a non-RRFD system $N$:

1. Find an RRFD system $M$ such that $M$ and $N$ are equivalent, in the sense that they implement each other.

2. Given a system $N$ are there weakest and strongest RRFD systems $Q_w$ and $Q_s$, respectively, equivalent to $N$, such that any RRFD system $Q$ equivalent to $N$, $Q$ is a submodel of $Q_w$ and $Q_s$ is a submodel of $Q$.

3. Find an RRFD system $M$ that corresponds to $N$ in the sense that is equivalent to $N$, but in addition "resembles" $N$ the most. We did not yet find a definition of this notion that is satisfactory to us.

What follows is rather informal discussion of RRFD systems that “correspond” to well known non-RRFD models of interest.

1. System $N$ is a synchronous message-passing system with at most $f < n$ processes that may fail by send-omission.

The RRFD system $A$ we propose is:

$$\forall p_i, r, \exists S : |D(i, r)| \leq f.$$  (1)

System $N$ implements $A$ by process $p_i$ designating $D(i, r)$ as the set of processes from which it failed to receive an $r$-round message by the end of the $r$th round. System $A$ implements $N$, by $p_i$ simulating the reception of the clock-tick when it ready to move to the next round.

2. System $N$ is a synchronous message-passing system with at most $f$ faulty processes that fail by crashing.

The RRFD system $A$ we propose satisfies predicate 1 and in addition:

$$(\forall r > 0)(\forall p \in S)(D(i, r) \subseteq D(k, r + 1)).$$ (2)

It is thus explicit in the model definition that the crash-fault model is a submodel of the send-omission-fault model.

3. System $N$ is an asynchronous message-passing system with at most $f$ crash-failures.

The RRFD system $A$ we propose satisfies:

$$(\forall r > 0)(\forall p \in S)(|D(i, r)| \leq f).$$ (3)

System $N$ implements $A$, by simulating rounds, discarding messages that have been missed, and buffering messages which are too early. Each round a process waits until it receives $n - f$ messages of the round. To see that system $A$ implements $N$, run $A$ in full information mode. When process $p_i$ receives a round $r$ message at round $r$ from $p_j$ it can recreate all the simulated messages it missed from $p_j$ since the last round it received a message from $p_j$. It can thus simulate their FIFO reception at that moment. Thus this simulation maps the runs (and views) of the RRFD
system $A$, to (a subset of) runs of system $N$. Consequently $A$ implements $N$.

To appreciate the difficulties one may encounter in explicitly identifying a weakest RRFD for certain systems, notice that contrary to intuition $A$ is not weakest for $N$. If $f < t$ and $2t < n$, we can allow $t$ processes to miss $t$ other processes. Formally, consider an RRFD system $B$ that satisfies:

$$
\exists Q \subseteq S, |Q| \leq t, (\forall p_i \in S - Q, \ D(i, r) \leq f), (\forall p_i \in Q, \ D(i, r) \leq t).$$

Two rounds of $B$ implement a round of $A$. Obviously, $A$ is a strict submodel of $B$.

4. System $N$ is an asynchronous SWMR shared memory system with at most $f$ crash-faulty processes.

In this system we have an array of registers $C_1, \ldots, C_n$. Process $p_i$, repeatedly, writes into $C_i$ and then reads all the other variables in some arbitrary order, at least once, until it reads at least $n - f$ values it did not read before. It is surprisingly difficult to find a natural explicit predicate (rather than specifying the predicate by a state machine) that captures $N$. This indicates to us that our understanding of this ubiquitous system is operational rather than declarative. Among many choices we settled for an RRFD system $A$ that satisfies predicate 3 and in addition:

$$
\forall r > 0, \ |\cup_{p_j \in S} D(i, r)| < n. \quad (4)
$$

This predicate says that in any round there is at least one process that is declared faulty to no process. Being a submodel of the RRFD system in item 3 it has at least as much power. But, we avoid the “network-partition” problem that message-passing with $2f \geq n$ encounters. For instance, to emulate $p_i$'s write operation of a value $v$ to its register, run the $A$ in a full information mode where $p_i$ indicates it is writing $v$. At the round that all messages received in $A$ by $p_i$ contain the knowledge of $v$ being written in the simulated system $N$, $p_i$ may terminate the simulated writing operation. In the subsequent round any process will know of $v$. To see that $N$ implements $A$, notice that the first process to write will be read by all.

To see the implementation of shared-memory by message-passing [22] in the context of RRFDs, notice that if $2f < n$ then two rounds of the RRFD system in item 3 implement a round of $A$. (In the second round each process emits the set of correct processes it heard from in the first round. The simulated set $D(i, r)$ is the set of processes $p_k$ has not heard of by the end of the second round. The reason predicate 4 is satisfied is that since in the first round all heard from a majority, there must be at least on process that was heard by majority. Such a process will be known to all at the end of the second round.)

As far as being a weakest RRFD, notice that shared-memory also satisfies: $\forall p_k, p_j, p_j \in D(i, r) \Rightarrow p_i \not\in D(j, r)$. But notice that this predicate does not imply predicate 4 since we can have $p_1$ miss $p_2$ that misses $p_3$, etc. until $p_n$ missing $p_1$. Thus, at the least we need to take the conjunction of the two predicates.

To see that the RRFD with this predicate is an alternative to predicate 4, notice that if after $k$ rounds no processor is known by all, then the “does not know” relation must contain a cycle, since we have a directed graph in which each process has at least one outgoing edge, corresponding to the “does not know” relation. Information is passed on the cycle in the reverse direction in all $k$ rounds. Thus a cycle must be of length at least $k + 1$. Consequently, after $n$ rounds there cannot be a cycle. (We conjecture that two rounds suffice.)

5. System $N$ is an asynchronous Atomic-Snapshot shared-memory system with at most $f$ processes that may fail by crashing.

This system has a natural RRFD corresponding system $A$. Its predicate satisfies predicate 3, and in addition:

$$
\forall r, \ p_k \not\in D(i, r) \land \\forall r, \ p_j \not\in D(i, r) \land \ D(i, r) \subseteq D(j, r) \lor (D(j, r) \subseteq D(i, r))
$$

Showing that this RRFD implements $f$-resilient Atomic-Snapshot shared-memory is a simple corollary of [4].

6. System $N$ is an asynchronous message-passing system with traditional failure detector $S$ [6].
In this asynchronous system all but one (a-priori unknown) process may fail. But the system is augmented with a failure detector that eventually announces any “real” crash, and never announces as faulty, one process that never fails. Other processes may or may not be announced as faulty.

The RRFD system that will naturally correspond to $S$ is the one that satisfies:

$$(\exists p_j) (p_j \notin \cup_{r>0} \cup_{r \in S} D(i,r)).$$

(Processes use the failure detector $S$ to advance from one round to the next. Thus, $D(i,r)$ is the value that allows $p_k$ to complete round $r$.) This specification needn’t include the fact that every faulty process must be announced by $S$ eventually. This comes “for free” when using an RRFD system. If a process really crashes and is not announced, the system will block, and thus vacuously implement the asynchronous system with $S$.

We next notice that an equivalent predicate is:

$$|\cup_{r>0} \cup_{p \in S} D(i,r)| < n,$$

which corresponds item 1 (with $f = n-1$). Thus we have reduced the existence of a wait-free algorithm for $S$ to the existence of algorithm for consensus in item 1, just by predicate manipulation.

3 RRFD’s and k-Set Agreement

RRFD’s can also be studied in relation to classical problems in the field of distributed computing. This section focuses on the problem of $k$-set agreement [23]. This problem requires each of a collection of $n > k$ processes to choose a value that is the initial value of one of the processes; at most $k$ different values can be chosen. Note that, for $k = 1$, this is the traditional consensus problem.

The following fault-detector

$$(\forall r > 0) \left( \cup_{p \in S} D(i,r) - \cap_{p \in S} D(i,r) < k \right).$$

is the one we propose to capture set consensus. Theorem 3.1 shows that this RRFD system can implement $k$-set consensus. Theorem 3.3 shows that any system that can implement $k$-set consensus (and shared memory) can implement this RRFD.

Before proving this, consider the nature of this fault detector. In each round, the number of processes detected by some process but not by all is less than $k$. Thus, this imposes a bound on the “uncertainty” exhibited by the fault detector. For $k = 1$, this means that the fault detectors at different processes cannot disagree. The results of this section clearly quantify the intuition that, the weaker the problem to be solved (i.e., the larger $k$ is), the weaker the system can be (as measured by its RRFD).

Theorem 3.1 The problem of $k$-set agreement can be solved with a detector supporting the following:

$$(\forall r > 0) \left( \cup_{p \in S} D(i,r) - \cap_{p \in S} D(i,r) < k \right).$$

Proof Using this detector, $k$-set consensus can be solved in one round. A process $p_i$ emits its value and chooses the value of the process in $S - D(i,1)$ with the lowest process identifier. If $v_1, v_2$ are two chosen values corresponding to $p_1 < p_2$ then it implies that $p_1$ is in the union of the faulty sets (some process chose $p_1$) but not in the intersection (some process chose $p_2$). Since the size of the union minus the intersection is less then $k$, it implies that at most $k$ distinct values can be chosen.

An immediate corollary of Theorem 3.1 is the following result of Chaudhuri [23]. It follows since $k-1$-resilient shared-memory implements $k-1$-resilient Atomic-Snapshot, and the corresponding RRFD predicate of item 5, implies the predicate of Theorem 3.1.

Corollary 3.2 $k$-set agreement can be solved in an asynchronous shared-memory system with at most $k-1$ failures.

Section 5 shows how a partially synchronous system defined by Dolev, Dwork, and Stockmeyer [14] can be used to implement the failure detector of Theorem 3.1 with $k = 1$, thus giving a solution to consensus.

The following theorem shows that a system that can solve $k$-set consensus can implement the RRFD above.

Theorem 3.3 Suppose that a system allows a solution to the problem of $k$-set consensus and also
that the system can implement SWMR shared-memory. Then the system supports a detector with the following property:

\[(\forall r > 0) \left( \bigcup_{p_i \in S} D(i, r) - \cap_{p_i \in S} D(i, r) \right) < k \).

**Proof** To emit a value at round \( r \) a process appends it value (together with sequence-number \( r \)) to its cell. To compute the values \( D(i, r) \), the processes do the following. They run a \( k \)-set consensus algorithm, each using its identifier as input. Suppose that a process \( p_i \) receives \( j \) as its output in round \( r \). It then writes \( j \) to its cell and reads the rest of the cells. It reads the set \( Q \) of process identifiers (as well as \( \perp \) for the processes that have yet to write). The process then uses \( S - Q \) as its value of \( D(i, r) \). Obviously, it can read emitted values for \( Q \) at round \( r \), since these process participated already in the round.

Notice that two sets \( D(i, r) \) and \( D(j, r) \) can differ only on processors IDs that were chosen through the \( k \)-set agreement algorithm (all other processes are in both sets). Thus, the difference in the union minus the intersection is bounded by \( k \). But since all of them will exclude the chosen identifier that was written first to a SWMR variable, if follows that the difference is bounded by \( k - 1 \).

\[ \square \]

## 4 Relating Synchronous and Asynchronous Systems

This section shows how RRFD’s can be used to relate synchronous and asynchronous systems. Specifically, it shows that an Atomic-Snapshot asynchronous system with at most \( k \) crash-failures can implement the first \( \lfloor f/k \rfloor \) rounds of synchronous system with \( f \) omission or crash faults. This can be used to prove the result of Chaudhuri et al. [24] that there is no \( \lfloor f/k \rfloor \)-round algorithm for \( k \)-set agreement for the latter (synchronous) system. The existence of such an algorithm combined with the results of this section would imply the existence of an asynchronous algorithm for \( k \)-set agreement that tolerates \( k \) failures, which is known to be impossible [9, 11, 12]. (For the special case of \( k = 1 \), this means that the impossibility result of Fischer, Lynch, and Paterson [10] implies the lower bound of Fischer and Lynch [13].)

The result is given in two sections. Section 4.1 proves the result for send-omission failures in the synchronous systems. The proofs is by simple reduction in the context of RRFD systems. Section 4.2 strengthens the results to systems with crash failures.

The simple reduction of the omission-fault lower bound to the asynchronous impossibility result was not observed earlier since researchers tend to think of synchronous and asynchronous systems as living in different domains. Once observed, it suggests that the reduction can be extended to crash faults via the omission-crash transformers suggested in [15]. Thus the value of the RRFD framework is in its ability to suggest connections, previously overlooked. Once such a connection is made, the technical details do not require high level of ingenuity.

### 4.1 Send-Omission Failures

An RRFD systems a system \( A \) implements \( B \) if by combining some rounds of \( A \) to simulate a round of \( B \) we can simulate the messages emitted at the round and implement a predicate that implies \( B \)'s RRFD predicate.

**Theorem 4.1** Consider integers \( f \) and \( k \) such that \( f \geq k > 0 \). Asynchronous RRFD Atomic-Snapshot shared-memory system (item 5) with at most \( k \) failures, can implement the first \( \lfloor f/k \rfloor \) rounds of an RRFD message passing system with at most \( f \) omission failures (i.e., with the RRFD given in item 1).

**Proof** Consider an RRFD asynchronous Atomic-Snapshot shared-memory system with at most \( k \) failures. Consider an execution of this system for \( \lfloor f/k \rfloor \) rounds. We will map a fault in the RRFD asynchronous system to a fault in the synchronous one.

From item 5, this system supports a detector with the following property:

\[(\forall r > 0) \left( \bigcup_{p_i \in S} D(i, r) \right) < k \).

Thus, the RRFD has the following property:

\[ \left| \bigcup_{0 < r \leq \lfloor f/k \rfloor} \bigcup_{p_i \in S} D(i, r) \right| < k \lfloor f/k \rfloor \],

which implies

\[ \left| \bigcup_{0 < r \leq \lfloor f/k \rfloor} \bigcup_{p_i \in S} D(i, r) \right| \leq f. \]
This matches the property of the RRFD for a synchronous system with at most $f$ omission-failures over $\lceil f/k \rceil$ rounds. □

**Corollary 4.2** Any solution to $k$-set agreement in a synchronous system with at most $f$ omission-failures requires at least $\lceil f/k \rceil + 1$ rounds in the worst case.

Since this section is more of an exposition, we leave the proof of the Corollary to the next subsection where the result is extended to the more benign crash faults.

### 4.2 Crash Faults

This subsection adds some simple machinery needed to extend the results of the previous subsection from omission failures to crash failures. The techniques are similar to those developed by Neiger and Toueg [15] to convert algorithms tolerant of crash failures into ones tolerant of omission failures. Since we failed to find a ready-made off-the-shelf transformation that we can borrow from [15], and in order to make the paper self-contained, we will represent and utilize a simplified adopt-commit protocol that is introduced in [16].

The machinery we add is an *adopt-commit* protocol. Process $p_i$ inputs to the protocol a value $v_i$ it proposes. The output of $p_i$ in the protocol is to commit or adopt some input value $v$. The relation between the output of processes should satisfy:

1. If $v_j = v$ for all $j = 1, \ldots, n$, then all processes commit to $v$.
2. If any process commits to $v$ then all processes commit or adopt $v$.

The following protocol solves the adopt-commit problem wait-free (i.e. $f = n - 1$-resilient) in the SWMR shared memory system. The protocol for $p_k$ follows. We have two arrays $C_{1,1}, \ldots, C_{n,1}$ and $C_{1,2}, \ldots, C_{n,2}$ of SWMR registers initialized to $\bot$:

```plaintext
begin
  write $v_i$ to $C_{i,1}$
  $S := \cup_{j=1}^{n} \text{read } C_{j,1}$
  if $S - \{\bot\} = \{v\}$
      then $C_{i,2} := \text{"commit }v\text{"}$
  else $C_{i,2} := \text{"adopt }v\text{"}$
end
```

else $C_{i,2} := \text{"adopt }v\text{"}$

$S := \cup_{j=1}^{n} \text{read } C_{j,2}$
if $S - \{\bot\} = \{\text{commit }v\}$
      then return commit $v$
else if $\text{commit }v' \in S$
      then return adopt $v$
else return adopt $v_i$

The correctness of the protocol follows from the fact that if $p_i$ writes “commit $a$” in the second round, then no other process will write “commit $b$”, $b \neq a$, since the value $p_i$ proposes to commit to in round 2 has to be the same as the value written first in round 1. In round 2, if a processor commits to $v$ it must be that “commit $v$” was the value written first, and thus all processors will either commit or adopt $v$.

**Theorem 4.3** Theorem 4.1 holds when the RRFD for omission faults is replaced by the RRFD for crash faults.

**Proof** We show first how to implement a round of the synchronous system $B$ using three rounds of the asynchronous one $A$.

To simulate round $r$ of system $B$, inductively, $p_i$ has computed a simulated value to write for simulated round $r$. It has also a set of processes $F_i$ that it proposes to have crashed, which is empty at the beginning of round 1. Round $r$ of system $B$ will be simulated by 3 rounds of system $A$.

In the first round of $A$, $p_i$ writes its simulated value $v_{r,i}$ for round $r$. It then reads in a snapshot until the number of values it misses is less or equal to $k$. Let $M_i$ be set of processes whose values $p_i$ missed. Process $p_i$ sets $F_i := F_i \cup M_i$. In round 2 and 3 processors run in parallel $n$ adopt-commit protocols, one for each process $p_j \in S$. If $p_j$ in $F_i$ the input of $p_i$ in the adopt-commit protocol for $p_j$ is “$p_j$-faulty”, else “$p_j$-alive” (if it uses “$p_j$-alive”, then it includes the value it received from $p_j$).

At the end of the adopt-commit protocol for $p_j$, if $p_i$ either commits or adopts “$p_j$-faulty”, then it adds $p_j$ to $F_i$. If $p_i$ commits to “$p_j$-faulty,” it returns $\bot$ for the value from $p_j$ in simulated round $r$. Otherwise, it must have read, in the adopt-commit protocol, another process that proposed “$p_j$-alive”. This proposal includes $p_j$’s value for the round, and this value is used by $p_k$ as $p_j$’s value for round $r$. 

The correctness of the simulation follows. A process $p_j$ will appear to fail in a round $r$ only if some process commits it as faulty. Since all processes consequently will adopt or commit "$p_j$-faulty" at round $r$, $p_j$ will belong to all $F_i$’s at the beginning of round $r + 1$. They will all propose to fail it at round $r + 1$ and thus all will commit to its faultiness at round $r + 1$ and subsequent rounds.

Since each simulated round introduces at most $k$ new processes to $\bigcup_i F_i$, by the end of simulated round $[f/k]$, at most $f$ processes failed in the simulated synchronous system.

$\square$

The corollary to this theorem is the result of Chaudhuri et al. [24].

**Corollary 4.4** Corollary 4.2 holds with crash-faults RRFD replacing the send-omission RRFD.

**Proof** Use the simulation of Theorem 4.3 to simulate an algorithm. A process $p_i$ that ends with a view after simulating round $[f/k]$ in which it is committed to $p_i$ faulty, does not end up with a simulated view that allows it to chose a value. But if $p_i$ did not propose $p_i$ faulty at the beginning of round $[f/k]$, although it ended up committed to its failure, the process is still compatible with an alive process, since no process failed it in previous rounds. Thus we have only at most $k([f/k] - 1)$ processes that have not chosen a value. But since $n > f$, we have $n - k([f/k] - 1) > k$.

Suppose $k$-set consensus is solvable in $A$ in $[f/k]$ rounds, then after the simulation, processes whose output is "I crashed" can adopt a value from a process with a real output value, since the system is $k$ resilient and at least $k + 1$ processes have real outputs. The result is $k$-set consensus in an asynchronous system that is $k$-resilient, in contradiction to [9, 11, 12]. $\square$

The results apply verbatim to Atomic-Snapshot shared-memory substituted for the message passing.

## 5 Semi-Synchronous Systems

Dolev, Dwork, and Stockmeyer [14] considered systems that varied five different parameters and explored their ability to solve consensus. This section focuses on one of their models, specifically one with the following properties:

- There are no bounds on the relative speeds of processes (i.e., the processes are asynchronous).
- Processes fail by crashing.
- Processes perform a sequence of steps. Each step consists of receiving all messages that have been buffered by the communication subsystem since the last step and then broadcasting a message. In this particular model, such a step is atomic.
- The communication system supports broadcast: if process $q$ receives message $m$ from $p$, then all correct processes do so.
- Every message sent is delivered before any process can take $\Delta$ steps.

They showed that consensus is possible in this system by giving an algorithm that runs in $2n\Delta$ steps. They left as an open problem whether or not there was an $O(\Delta)$-time algorithm.

This section proves that this system admits a solution that runs in $2\Delta$ steps. It is done by showing that this system supports the RRFD given in Theorem 3.1 with $k = 1$ and $2\Delta$ steps per round (thus, allowing a consensus algorithm). This RRFD is identified by the following property:

$$(\forall r > 0)(\forall p_i, p_j \in S)(D(i, r) = D(j, r)).$$

Since the proof of Theorem 3.1 gives a one-round algorithm, there is an algorithm that runs in $2\Delta$ steps.

The following describes how the RRFD given in equation 5 can be implemented in $2\Delta$ steps. A process's execution occurs in blocks of $2\Delta$ steps. If a process receives a round-$r$ message before sending its own, then it sends no further messages (acting as if it has omitted to broadcast), although it continues to receive message from the others. Otherwise, it broadcasts its round-$r$ message, tagging it with the round number. Notice that, in a sense, we use the first receive-send in a round as an atomic read-modify-write. If the “receive” returns empty set of messages then a message is broadcasted, otherwise it is not. At the end of a round $r$, process $p_i$ takes $D(i, r)$ to be the set of processes from which it does not receive round-$r$ messages.
Theorem 5.1 The RRFD described for the semi-synchronous system supports equation 5.

Proof Let $p_i$ be the first process to execute a receive/send at round $r$. Let $p_j$ be a process that broadcasts a round $r$ message. We will show, that any process $p_k$ will receive a message from $p_j$ in round $r$. For all processes $p_k$ let $T_k$ denote the time that $p_k$ executes the first receive/send at round $r$. By definition $T_j \geq T_i$, and $T_i \geq T_j$. Since $p_j$ did not receive $p_i$'s message of round $r$ before $p_j$ started round $r$, we conclude $T_j \leq T_i + \Delta(k)$, where $\Delta(k)$ denotes the time of any consecutive $\Delta$ steps by $p_k$. Thus, $T_j \leq T_k + \Delta(k)$. Since receive/send is atomic we conclude that $p_k$ by the end of round $r$ at time $T_k + 2\Delta(k)$ would have receive the message sent at $T_j$. In particular, it will receive the message sent by $p_j$. Thus the semi-synchronous RRFD system satisfies equation 5. □

6 Related Work

Unification of synchrony and asynchrony in limited domain has in the past been proposed by Awerbuch with his celebrated synchronizer [17]. Awerbuch was able to show that if no faults are expected synchrony and asynchrony are the same. The two systems implement each other. A decade later in 1993 Chaudhuri, Herlihy, Lynch and Tuttle [24] showed that with a combination of techniques most of which resemble the topological arguments that established impossibility of set-consensus in the asynchronous domain, one can establish a lower bound in the synchronous case.

Recently, [4] introduced the idea of taking a model and defining its iterated version, forcing communication-closedness. This gave rise to the ideas in this paper. The similarity between the iterated version of asynchronous model and synchronous one suggested the possibility of deriving a lower bound in the synchronous case by reduction from impossibility result in the asynchronous case.

Herlihy, Rajsbaum, and Tuttle, in a paper in this proceedings [18], were influenced in another direction. They considered, in the words of [4], the model of iterated message passing. They characterized the structure of one-shot message-passing is a pseudo-sphere, a simple structure whose iteration is the structure itself. Using explicit topological argument they were able to derive the synchronous lower bound, and extend it beyond our work to the semi-synchronous model in [19]. Their result though apply solely to the message passing-system.

7 Conclusions and Future Work

This paper presented a framework, RRFD, that unifies the most seemingly unrelated notions in distributed computing - synchrony and asynchrony. It has established the case for the framework by showing that it is a bridge that draws attention to the similarities between the models, and how results in one model may be transferred to the other. Moreover, these model are not only of theoretical interest. We advocate using them. We propose them as a setting to develop real algorithms. As with programming languages, restriction imposed on the programmer in the form of adhering to communication-closed-layers may be a blessing. It force the programmer to a line of design that will result, we hope, simple structured algorithms [20].

Future immediate work which we hope to add to the full version of the paper, is the extension of the reduction to the semi-synchronous model in [19]. Essentially, our contention by using reduction rather than direct topological arguments, is that there are very few instances of problems that one needs explicit topology. The use of explicit topology is analogous to proving a problem NP-complete by a direct reduction to Turing-Machine computation as done with Satisfiability, rather then by reduction to a similar problem. In fact, there may exist a small family of impossibility results, so that any impossibility or lower bound result in the distributed domain may be derived from them by reduction.

Finally, it will be interesting to show that in a precise sense RRFD generalizes the earlier notion of fault-detector [5, 6, 7, 8], and rederive the associated results. Such an investigation will be an instance of the general investigation as to what kind of systems can be implemented given a restricted language in which an RRFD predicate may be stated.
Acknowledgments

Gil Neiger has been my RRFD through many rounds of conceiving and writing the paper. The enormous amount of effort he dedicated to this paper is gratefully acknowledged. I would also like to thank DEC-SRC for providing me an office away from my office.

References