Condition-Based Consensus in Synchronous Systems

Yoav Zibin

Technion—Israel Institute of Technology
zyoav@cs.technion.ac.il

Abstract. The *condition-based* approach for solving problems in distributed systems consists of identifying sets of input vectors, called *conditions*, for which it is possible to design more efficient protocols. Recent work suggested using the condition based approach in *asynchronous systems* suffering from crashes, for solving various agreement problems [5, 9, 1, 4, 6]. This paper designs a fast condition-based consensus protocol for *synchronous systems*.

1 Introduction

The consensus problem stands in the heart of distributed systems prone to failures [2]. In this problem each process proposes a value and all correct processes must agree on one of the proposed values. It is well known that consensus cannot be solved in *asynchronous systems* even in the presence of a single possible crash. Many ways have been suggested to circumvent this impossibility result, such as considering a weaker problem (approximate agreement, set agreement, and randomized algorithms) or a stronger environment (failure detectors, and partially asynchronous environment).

The *condition-based* approach, which is the focus of this paper, consists of identifying sets of input vectors, called *conditions*, for which a problem is solvable. Recent work used the condition based approach in asynchronous systems suffering from crashes, for solving consensus [5], set-agreement [9, 1], and interactive-consistency [6]. These problems were also studied in the presence of Byzantine errors [6, 4]. A correlation with error correcting codes was recently found [4], where crashes correspond to erasures, and Byzantine faults to corruption errors.

Consider, for example, the condition $C_d^{\max}$ which includes only input vectors in which the largest value appears at least d times. It is easy to solve consensus if the input always belongs to $C_{f+1}^{\max}$, where f is the number of processes that can crash. The difficulty is in designing a *strict consensus protocol*, i.e., a protocol which also handles inputs that do not belong to the condition. Mostefaoui et al. [5] presented a strict consensus protocol which always guarantee safety (i.e., agreement and validity), however the protocol may not terminate if the input does not belong to $C_{f+1}^{\max}$. More generally, Mostefaoui et al. defined the class of d-legal conditions and presented a generic protocol for any $f+1$-legal condition. A more efficient consensus protocol for stronger conditions was presented later [8, 7].

In *synchronous systems* it is well known that solving consensus requires at least $f+1$ rounds. This paper uses the condition-based approach in order to design a consensus protocol which requires a smaller number of rounds. Specifically, for d-legal conditions, $1 \leq d \leq f+1$, we present a protocol which solves consensus in $(f+1)-(d-1)$

F.E. Fich (Ed.): DISC 2003, LNCS 2848, pp. 239–248, 2003.

rounds if the input always belongs to the condition. We also present a strict consensus protocol which adds an additional validation round to ensure all processes reached agreement (we assume that $2f < n$). More precisely, if the input belongs to the condition then consensus is solved in $(f + 1) - (d - 1) + 1$ rounds, otherwise it is solved in $f + 1$ rounds.

Outline Sec. 2 presents our notation and defines the strict and non-strict variants of the condition-based approach. Sec. 3 defines the consensus problem and describes the well-known flood-set protocol for solving it. The non-strict consensus protocol for any d-legal condition is presented in Sec. 4, and its strict variant in Sec. 5. Finally, open problems and direction for future research are discussed in Sec. 6.

2 Notation

This paper assumes a standard *synchronous message passing* model. The number of processes is denoted by n, and f is a bound on the number of processes that can *crash* (without ever recovering). A process that did not crashed is called *correct*.

Definition 1 (Immediate crash). *A process which crashes in the first round is said to* immediately crash. *(Note that this process can still send messages in the first round to a subset of the other processes.) Similarly, an* immediate crash *is a crash that occurred in the first round.*

The finite set of input values is denoted by $\mathcal{V}$. We assume a default value $\perp$ not in $\mathcal{V}$. Let $\mathcal{V}^n$ be the set of all possible vectors (of size n) with entries from $\mathcal{V}$, and let V_f^n be the set of all possible vectors with entries from $\mathcal{V} \cup \perp$, with at most f entries equal to $\perp$. We typically denote by I a vector in $\mathcal{V}^n$ and by J a vector in $\mathcal{V}_f^n$.

Definition 2 (Partial view). *For vectors $J_1, J_2 \in \mathcal{V}_f^n$, we say that J_1 is a* partial view *of J_2, denoted $J_1 \leq J_2$, if for $k = 1, \ldots, n$, $J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$, i.e., J_1 can be* extended *into J_2 by changing some of its $\perp$ entries.*

Definition 3 (Union of partial views). *When $J_1, J_2 \in \mathcal{V}_f^n$ are partial views of the same vector I, i.e., $J_1, J_2 \leq I$, we define their* union $J = J_1 \cup J_2$, *such that for $k = 1, \ldots, n$,*

$$J[k] = a \neq \perp \Leftrightarrow J_1[k] = a \textbf{ or } J_2[k] = a.$$

Observe that $J_i \leq J \leq I$ for $i = 1, 2$.

We define two functions:

- $\#_x(J) = |\{i \mid J[i] = x\}|$, i.e., the number of entries of J whose value is x.
- $\mathrm{dist}(J_1, J_2) = |\{i \mid J_1[i] \neq J_2[i]\}|$, i.e., the Hamming distance between J_1 and J_2.

We assume that there is some linear ordering of $\mathcal{V}$ and we can define the function $\max : \wp(\mathcal{V}) \mapsto \mathcal{V}$. Next, we assume that $\perp$ is the smallest value, and extend $\max$ to handle also $\perp$ values. We will sometimes treat a vector $J \in \mathcal{V}_f^n$ as a set and write $\max(J)$; Note that $\perp$ will never be picked by $\max$ unless $J = \{\perp\}^n$. By defining a lexicographic ordering over $\mathcal{V}^n$, we can extend this function to handle vectors, i.e., $\max : \wp(\mathcal{V}^n) \mapsto \mathcal{V}^n$. (No confusion will occur due to this overloading of the $\max$ function.)

Definition 4 (Conditions). *A* condition $C \subseteq \mathcal{V}^n$ *is a set of input vectors.*

A protocol which solves *non-strict* consensus [4] for a given condition C need to consider only input vectors in C, i.e., if the input vector does not belong to C then no guarantees are made on the output. In contrast the *strict* variant must consider every possible input vector in $\mathcal{V}^n$. In an *asynchronous* environment [5, 8, 9, 7, 4], a strict protocol must always guarantee safety (i.e., agreement and validity), even for input vectors that do not belong to C, but it may not terminate. (The consequences of making other properties more strict can be found in [5, 4], e.g., a correct process must always terminate but it may sometimes decide on $\perp$.) In a *synchronous* environment the protocol must always terminate, and it must terminate faster for input vectors in C.

Definition 5 (d-legal conditions). *A condition C is called* d-legal *if there exists a mapping $h : C \mapsto \mathcal{V}$ with the following properties:*

1. $\forall I \in C : \#_{h(I)}(I) \geq d$,
2. $\forall I_1, I_2 \in C : h(I_1) \neq h(I_2) \Rightarrow \mathrm{dist}(I_1, I_2) \geq d$.

It was shown [5, 4] that $(f+1)$-legal conditions are necessary and sufficient for solving both strict and non-strict consensus in asynchronous systems.

An example of a d-legal condition is $C_d^{\max}$,

$$C_d^{\max} = \{I \in \mathcal{V}^n \mid \#_{\max(I)}(I) \geq d\}, \text{i.e., } d \text{ occurrences of the maximum value.}$$

The mapping h associated with $C_d^{\max}$ returns the maximal value in the vector. It is straight forward to verify the two properties h must have according to Def. 5.

3 Consensus Problem

In the *consensus* problem each process has to *decide* on a value (the output value), such that the following properties holds:

Agreement: No two different values are decided by *correct* processes.
Termination: A correct process must decide.
Validity: The decided value is one of the proposed values.

In all the protocols in this paper, whenever a process decides it also halts. Therefore every process that decides (whether correct or *faulty*) chooses the same value; this property is called *uniform agreement*, and our protocols in fact solve *uniform consensus*. (If processes are allowed to halt after they decided then solving consensus requires at least $t+1$ rounds, whereas solving uniform consensus requires at least $t+2$ rounds [3], where $t < f - 1$ is the number of processes that actually crashed.)

Protocol 1 describes the well-known *flood-set protocol* (see e.g., [2]) in which sets of input values are iteratively flooded during d rounds. Each process p_i represents its set as a partial view $J_i \leq I$ which is extended during d rounds. It is well known that after $f+1$ rounds all the partial views are equal and thus $\mathsf{FloodSet}_{f+1}$ solves consensus.

In order to prove our results in the next sections, we will need the following lemmas.

Protocol 1 $\mathsf{FloodSet}_d$

Input: a value $x_i \in \mathcal{V}$.
Output: a decided value $y_i \in \mathcal{V}$, and a partial view $J_i \in \mathcal{V}_f^n$.
Code for process p_i:

```
1: J_i ← {⊥}^n
2: J_i[i] ← x_i // we always maintain the invariant that J_i ≤ I
3: For r = 1, ..., d do // run for d rounds
4:    Send J_i to all processes (including yourself)
5:    V ← ∅ // the set of partial views received
6:    Upon receiving J_j from p_j do V ← V ∪ {J_j}
      // update our partial view
7:    J_i ← ⋃_{J_j ∈ V} J_j
8: end For
9: y_i ← max(J_i)
10: return ⟨y_i, J_i⟩
```

Lemma 1. *For every correct process p_i, $J_i \leq I$.*

Proof. Observe that J_i is updated only in lines 1,2 and 7. By induction, each $J_j \in V$ satisfies $J_j \leq I$, and therefore their union also satisfies $\bigcup_{J_j \in V} J_j \leq I$.

Lemma 2. *Let p_i be a correct process. If p_j did not* immediately crash *then $J_i[j] = I[j]$, and if $J_i[j] = \perp$ then p_j must have* immediately crashed*.*

Proof. If p_j did not immediately crash then it must have sent its input to p_i, i.e., $J_i[j] = I[j] \neq \perp$.

Lemma 3. *Let $J_i, J_j \leq I$ be two resulting partial views from running $\mathsf{FloodSet}_d$. Then, if there exist I'_i, I'_j, such that $J_i \leq I'_i$, $J_j \leq I'_j$, and $\mathrm{dist}(I'_i, I'_j) \geq k$, then there were at least k* immediate crashes*.*

Proof. Assume by contradiction that there were less than k immediate crashes. Let $S \subseteq [1, n]$, $|S| \geq k$, be the set of indices where I'_i and I'_j differ, i.e., for every $m \in S$, $I'_i[m] \neq I'_j[m]$. Then, there is some $m \in S$ such that p_m did not immediately crashed. From Lemma 2, if either $J_i[m] = \perp$ or $J_j[m] = \perp$ then p_m immediately crashed. Therefore, both $J_i[m] \neq \perp$ and $J_j[m] \neq \perp$. Since $J_i, J_j \leq I$, we have that $J_i[m] = J_j[m]$. Since $J_i \leq I'_i$ and $J_j \leq I'_j$, then $I'_i[m] = I'_j[m]$; contradiction.

Lemma 4. *If d* immediate crashes *occurred then after running $\mathsf{FloodSet}_{(f+1)-(d-1)}$ all the partial views are equal.*

Proof. Since in the first round there were at least d immediate crashes, there could be at most $(f - d)$ crashes in the remaining $(f - d) + 1$ rounds. Therefore, there must be a round in which no crashes occurred, after which all the partial views become equal.

In order to gain some intuition on the consensus protocols of the next sections, we first show how to solve non-strict consensus for the specific condition $C_d^{\max}$.

Lemma 5. *Running* $\mathsf{FloodSet}_{(f+1)-(d-1)}$ *solves non-strict consensus for the condition* $C_d^{\max}$.

Proof. It is straightforward to prove **Validity** and **Termination** in $(f+1)-(d-1)$ rounds. Next we prove **Agreement**. Let $I \in C_d^{\max}$ be the input vector. Then, there are at least d processes whose input is $\max(I)$. On the one hand, suppose that one of those d processes did not *immediately crash*. Then according to Lemma 2, for every correct process p_i, we have $\max(I) \in J_i$, and thus p_i decides on $\max(I)$. On the other hand, if those d processes did *immediately crash*, then according to Lemma 4, all the partial views are equal, i.e., $J_i = J_j$, and every correct process decides on the same $\max(J_i)$.

Lemma 6. *It is not possible to solve non-strict consensus for* $C_d^{\max}$ *in less than* $(f+1)-(d-1)$ *rounds.*

Proof. Observe that, when $d = 1$, $C_1^{\max} = \mathcal{V}^n$, i.e., we need to solve the problem for any possible input and therefore the known lower bound of $f+1$ rounds holds. When $d > 1$, by causing $d-1$ of the processes with the maximal input to crash even before they start, we returned to the general form of the problem. Specifically, afterwards we need to solve consensus with up to $f-(d-1)$ crashes for *any possible input*, which cannot be done in less than $(f-(d-1))+1$ rounds.

4 Non-strict Consensus Protocol

Given a d-legal condition C, and its associated mapping $h : C \mapsto \mathcal{V}$ (see Def. 5), the protocol for solving *non-strict consensus* is presented in Protocol 2. Remember that when the input does not belong to C we make no guarantees on the protocol.

Protocol 2 Non-strict consensus protocol for a d-legal condition C with the mapping $h : C \mapsto \mathcal{V}$

Input: a value $x_i \in \mathcal{V}$.
Output: a decided value $y_i \in \mathcal{V}$, and a partial view $J_i \in \mathcal{V}_f^n$.
Code for process p_i:

1: $J_i \leftarrow \mathsf{FloodSet}_{(f+1)-(d-1)}(x_i)$ // *Run the protocol for* $(f+1)-(d-1)$ *rounds*
2: $\mathcal{E}_i \leftarrow \{I' \in C \mid h(I') \in J_i$ **and** $J_i \leq I'\}$ // *Determining the set of candidate vectors*
3: **If** $\mathcal{E}_i = \emptyset$ **then**
   // *It must be that* $\#_\perp(J_i) \geq d$, *and thus* $J_i = J_j$ *for all correct processes* p_i, p_j.
4: $y_i \leftarrow \max(J_i)$
5: **else**
6: $I'_i \leftarrow \max(\mathcal{E}_i)$
7: $y_i \leftarrow h(I'_i)$
8: **return** $\langle y_i, J_i \rangle$

In line 1 process i calculates its partial view J_i by running $\mathsf{FloodSet}_{(f+1)-(d-1)}$. The essence of the protocol is that J_i is either "close enough" to the input vector or that all the partial are equal. Lines 2–7 are a deterministic procedure to select the agreed

value y_i. In line 2 we determine a set of candidate vectors from C. Each candidate I' must satisfy two requirements: (i) I' is an extension of J_i, $J_i \leq I'$, and (ii) $h(I') \in J_i$. The second requirement ensures the validity of our protocol, however because of it, the input vector I might not be a candidate. If the set of candidates is empty we will prove that all the partial views are equal and the processes can agree on $\max(J_i)$ (lines 3–4). Otherwise, we can deterministically select any candidate vector and agree on $h(I'_i)$ (lines 6–7). We note that the use of the $\max$ function in lines 4 and 6 is arbitrary; Any other deterministic decision procedure will maintain the correctness of the protocol.

Consider, for example, the condition $C_2^{\max}$, where the maximal value must appear at least twice, and h returns the maximal value in a vector. Assume that $f = 3$, $n = 5$, and $\mathcal{V} = \{1, 2, 3, 4\}$. Suppose that we run the protocol with the input $I = \langle 1, 4, 4, 3, 2\rangle$, and process i finishes with the partial view $J_i = \langle \perp, \perp, \perp, 3, 2\rangle$. There are many extensions of J_i which belong to C, e.g.,

$$I = \langle 1, 4, 4, 3, 2\rangle$$
$$I' = \langle 1, 3, 3, 3, 2\rangle$$

The set $\mathcal{E}_i$ includes I' since $J_i \leq I'$ and $h(I') = 3 \in J_i$, but

$$I \notin \mathcal{E}_i,$$

since $h(I) = 4 \notin J_i$.

In general, the fact that $I \notin \mathcal{E}_i$ is not surprising. Recall that we demanded that for every $I \in C$, $\#_{h(I)}(I) \geq d$. It might be the case that all those d processes crashed, and you cannot choose $h(I)$ because you have not seen $h(I)$, i.e., $h(I) \notin J_i$. However, we will prove that in such a case all the partial views J_i are equal, and thus you can safely choose $\max(J_i)$.

Theorem 1. *Protocol 2 solves non-strict* consensus *for any d-legal condition C, i.e., for any input vector $I \in C$ it satisfies:*

Agreement *No two different values are decided.*
Termination *A correct process decides in $(f + 1) - (d - 1)$ rounds.*
Validity *The decided value is one of the proposed values, i.e., $y_i \in I$.*

Proof. It is straightforward to see that every correct process decides after exactly $(f + 1) - (d - 1)$ rounds.

Next we prove **Validity**, i.e., $y_i \in I$. From Lemma 1, we have that $J_i \leq I$. On the one hand, if $\mathcal{E}_i = \emptyset$ then $y_i \in J_i$, and since $y_i \neq \perp$ we have that $y_i \in I$. On the other hand, every $I' \in \mathcal{E}_i$ satisfies $h(I') \in J_i$, and again since $y_i = h(I') \neq \perp$ we have that $y_i \in I$. (Note that validity holds even when $I \notin C$.)

Finally we prove **Agreement**. Assume by contradiction that two processes p_i and p_j decided on different values $y_i \neq y_j$. Note that lines 2–8 are deterministic, i.e., if $J_i = J_j$ then $y_i = y_j$ (which will be a contradiction). We wish to show that d processes *immediately crashed*, since by applying Lemma 4, after running $\mathsf{FloodSet}_{(f+1)-(d-1)}$, we will have the contradiction that $J_i = J_j$.

Suppose that one of p_i or p_j reached line 4. W.l.o.g., assume it is p_i, i.e., $\mathcal{E}_i = \emptyset$. Since $J_i \leq I$ and $I \notin \mathcal{E}_i$ we have that $h(I) \notin J_i$. From the first property of h (see

Def. 5), $\#_{h(I)}(I) \geq d$. Thus those d entries must be $\perp$ in J_i, and we have d processes which *immediately crashed*.

Now suppose that both p_i and p_j reached line 6, i.e., $y_i = h(I'_i)$ and $y_j = h(I'_j)$. Since the condition is d-legal we have that

$$h(I'_i) \neq h(I'_j) \Rightarrow \mathrm{dist}(I'_i, I'_j) \geq d.$$

Since $J_i, J_j \leq I$, $J_i \leq I'_i$, $J_j \leq I'_j$, and $\mathrm{dist}(I'_i, I'_j) \geq d$, from Lemma 3 we know that d processes *immediately crashed*. □

5 Strict Consensus Protocol

It was proven [5, Theorem 7.1] that in order to solve strict consensus (for a non-trivial condition) in an *asynchronous* system, you must assume that $2f < n$. Although this is not mandatory in *synchronous* systems, we also assumed that $2f < n$ in our strict consensus protocol which is presented in Protocol 3.

This protocol amends the non-strict consensus protocol of the previous section by adding an additional validation round. If the processes reached an agreement then the protocol stops after running for only $(f+1)-(d-1)+1$ rounds. Otherwise the input does not belong to the condition, and the protocol runs for $f+1$ rounds.

The idea is to check if we reached consensus using Protocol 2, and in parallel to continue running the basic flood-set protocol. If we reached consensus after running Protocol 2 in line 1, then all the suggestions y_i will be equal, and all processes will reach line 19 and terminate. However, it is possible that consensus have not been reached and still one process will terminate in line 19 and return the value y, and another process p_j will not.

If we removed our assumption that $2f < n$ then it is possible that p_j will decide on a different value in line 19. But if $2f < n$, we can be sure that y is the majority value among the suggestions p_j received, and p_j will reach line 21 and eventually decide on y.

Another problematic scenario is the possibility that one process p_i decides on a majority value m_i in line 26, and another process p_j decides on the maximal value in J_j in line 28. To avoid this possibility, the majority value m_i is flooded to all other processes using the $\mathrm{Majority}$ message. Since we ran for $f+1$ rounds there must be a round in which no crashes occurred. Therefore, either (i) consensus was reached in line 1 or (ii) a $\mathrm{Majority}$ message will reach p_j. In both cases p_j will not decide in line 28.

Theorem 2. *When $2f < n$, Protocol 3 solves strict consensus for any d-legal condition C, i.e., it satisfies:*

Agreement *No two different values are decided.*
Termination *A correct process decides in at most $f+1$ rounds.*
Improved_Termination *When the input belongs to the condition, a correct process decides in $(f+1)-(d-1)+1$ rounds.*
Validity *The decided value is one of the proposed values.*

Protocol 3 Strict consensus protocol, when $2f < n$, for a d-legal condition C

Input: a value $x_i \in \mathcal{V}$.
Output: a value in $\mathcal{V}$.
Code for process p_i:

1: $\langle J_i, y_i \rangle \leftarrow$ run Protocol 2 on x_i // *Runs for* $(f+1)-(d-1)$ *rounds // y_i is the* suggestion *of process i*
2: $m_i \leftarrow \perp$ // *The majority value among the suggestions*
3: **For** $r = 1, \ldots, d-1$ **do** // *run for $d-1$ more rounds*
4: **If** $m_i = \perp$ **then**
5: **Send** $\langle \text{Suggestion}, J_i, y_i \rangle$ **to** all processes (including yourself)
6: **else**
7: **Send** $\langle \text{Majority}, m_i \rangle$ **to** all other processes
8: $V \leftarrow \emptyset$ // *the set of partial views received*
9: $S \leftarrow \{\perp\}^n$ // *the vector of suggestions received*
10: **Upon receiving** msg_j **from** p_j **do**
11: **If** $\langle \text{Majority}, m_j \rangle = \text{msg}_j$ **then**
12: $m_i \leftarrow m_j$
13: **else**
14: Let $\langle \text{Suggestion}, J_j, y_j \rangle = \text{msg}_j$
15: $V \leftarrow V \cup \{J_j\}$ // *Collect the partial views*
16: $S[j] = y_j$ // *Collect the suggestions*

17: **If** $m_i = \perp$ **then**
18: **If** $\exists y$ such that: $\#_y(S) = n - \#_\perp(S)$ **then** // *all the values in S are equal to y*
19: **return** y
20: **else if** $\exists y$ such that: $\#_y(S) > \frac{n}{2}$ **then** // *y is the majority value*
21: $m_i \leftarrow y$
22: **else**
23: $J_i \leftarrow \bigcup_{J_j \in V} J_j$ // *update our partial view*
24: **end For**

25: **If** $m_i \neq \perp$ **then**
26: **return** m_i
27: **else**
28: **return** $\max(J_i)$

Proof. **Termination** in $f+1$ rounds is straight forward: we run Protocol 2 for $(f+1)-(d-1)$ and then we run for $d-1$ more rounds. **Improved_Termination** is also easy: when the input belongs to the condition, all the values y_i returned from Protocol 2 are equal, and in the next round all processes will reach line 19 and terminate.

Next we prove **Validity**. Note that the validity proof of the non-strict version (see Thm 1) did not use the fact that the input belongs to the condition, i.e., $I \in C$. Therefore, after line 1, we have that $y_i \in J_i \leq I$. Since we decide either on some suggestion y_i or on $\max(J_i)$, the decided value is one of the proposed values.

Finally we prove **Agreement**. Assume by contradiction that two processes p_i and p_j decided on different values. It cannot be that both p_i and p_j decided in line 19 or 26, because it cannot be that two different suggestions are in majority. Therefore, at least one of p_i or p_j decided in line 28, w.l.o.g., assume it is p_i. Note that $m_i = \perp$, i.e., no $\mathrm{Majority}$ message reached p_i.

Suppose that some process decided in line 19 on the value y, i.e., p_j only saw suggestions for the value y. Then, in the same round it must be that all the other processes saw at least $\lceil \frac{n}{2} \rceil$ suggestions for y. Otherwise, the number of correct processes is less than $\lceil \frac{n}{2} \rceil$. But that cannot be for p_i, since p_i decided in line 28 and $m_i = \perp$.

Therefore p_j decided in line 26 or 28. Since both p_i and p_j ran for $f+1$ rounds, and there are at most f crashes, there must be a round r in which no crashes occurred. If $r \leq f - d + 2$, then after running Protocol 2 in line 1 all processes would reach consensus and terminate in line 19. Therefore, $r \geq f - d + 3$. In round r, all processes will have the same set V, and therefore starting from that round we always have that $J_i = J_j$. Process p_j cannot decided in line 28 since $J_i = J_j \Rightarrow \max(J_i) = \max(J_j)$. Thus, p_j decided in line 26.

It must be that p_j received its $\mathrm{Majority}$ message exactly in *the last round*, since otherwise p_j would send a $\mathrm{Majority}$ message to p_i the next round (but p_i decided in line 28 so it did not receive any $\mathrm{Majority}$ message). Note that a process p_k can reach line 21 only in round $f - d + 3$ because the number of suggestions with the majority value can only decrease in time due to crashes. By examining Protocol 3 we can see that there must be a series of $\mathrm{Majority}$ messages starting from a process p_k that reached line 21 in round $f - d + 3$ and ending with process p_j in the last round (without ever reaching p_i). But this is a contradiction since in some round $r \geq f - d + 3$ there were no crashes. □

6 Open Problems

This paper presented both a strict and a non-strict consensus protocol for any d-legal condition in synchronous systems. When the input belongs to the condition, the non-strict protocol saves $d - 1$ rounds, and the strict protocol saves $d - 2$ rounds. It is easy to modify the protocols described here to solve the interactive-consistency problem in which the processes need to agree on an equal vector (see the author website[1]).

We could neither prove the optimality of our algorithms nor the necessity of assuming $2f < n$ in the strict variant.

The most important problem this paper leaves open is finding classes of conditions for which k-set agreement can be solved more efficiently. Even in asynchronous systems, except for the wait-free case [1], no necessary conditions were found. Finally, generalizing the results for other kinds of crashes, e.g., Byzantine faults, also seems like a worthwhile task.

[1] `http://www.cs.technion.ac.il/~zyoav`

References

1. H. Attiya and Z. Avidor. Wait-free n-set consensus when inputs are restricted. In *Proceedings of the 16th International Symposium on DIStributed Computing (DISC 2002)*, pages 118–132. Springer LNCS 2508, 2002.
2. H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill, 1998.
3. B. Charron-Bost and A. Schiper. Uniform consensus harder than consensus. Technical Report DSC/2000/028, École Polytechnique Fédérale de Lausanne, Switzerland, May 2000.
4. R. Friedman, A. Mostefaoui, S. Rajsbaum, and M. Raynal. Distributed agreement and its relation with error-correcting codes. In *Proceedings of the 16th International Symposium on DIStributed Computing (DISC 2002)*, pages 63–87. Springer LNCS 2508, 2002.
5. A. Mostefaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. In *Proceedings of the 33rd annual ACM symposium on Theory of computing*, pages 153–162. ACM Press, 2001.
6. A. Mostefaoui, S. Rajsbaum, and M. Raynal. Asynchronous interactive consistency and its relation with error-correcting codes. In *Proceedings of the 21st annual symposium on Principles of distributed computing*, pages 253–253. ACM Press, 2002.
7. A. Mostefaoui, S. Rajsbaum, M. Raynal, and M. Roy. Efficient condition-based consensus. In *Proceedings of the 8th International Colloquium on Structural Information and Communication Complexity (SIROCCO'01)*, pages 275–291. Carleton Univ. Press, 2001.
8. A. Mostefaoui, S. Rajsbaum, M. Raynal, and M. Roy. A hierarchy of conditions for consensus solvability. In *Proceedings of the 20th annual ACM symposium on Principles of distributed computing*, pages 151–160. ACM Press, 2001.
9. A. Mostefaoui, S. Rajsbaum, M. Raynal, and M. Roy. Condition-based protocols for set agreement problems. In *Proceedings of the 16th International Symposium on DIStributed Computing (DISC 2002)*, pages 48–62. Springer LNCS 2508, 2002.