Bounded Cost Algorithms for Multivalued Consensus Using Binary Consensus Instances

Jialin Zhang*
Tsinghua University
zhanggl02@mails.tsinghua.edu.cn

Wei Chen
Microsoft Research Asia
weic@microsoft.com

Abstract

In this paper, we present two bounded cost algorithms that solve multivalued consensus using binary consensus instances. Our first algorithm uses $\lceil \log_2 n \rceil$ number of binary consensus instances where n is the number of processes, while our second algorithm uses at most $2\widetilde{k}$ binary consensus instances, where $\widetilde{k}$ is the maximum length of the binary representation of all proposed values in the run. Both algorithms are significant improvements over the previous algorithm in [6], where the number of binary consensus instances needed to solve one multivalued consensus is unbounded.

Keywords: distributed computing, fault tolerance, binary consensus, multivalued consensus.

1 Introduction

Consensus is a fundamental problem to solve when building fault-tolerant distributed systems. Informally, consensus abstracts the basic agreement problem one often sees in distributed systems as follows: Each process in a system proposes some value, and through communication they eventually need to decide on one value proposed, and the decision is irrevocable. Many important distributed tasks, such as atomic broadcast, data replication, mutual exclusion, atomic commit, etc., can use consensus as one of the core components in their implementations. Therefore, consensus has been extensively studied from various angles in the distributed computing community.

One basic form of consensus is *binary consensus*, in which processes may only propose 0 or 1 and the decision is one of the two values. Binary consensus is a form used in studying both impossibility and lower bound results (e.g., [2, 4]) and consensus algorithms (e.g., [1, 3]). However, solving the general multivalued consensus using binary consensus is not trivial. In [7], Turpin and Coan provide an algorithm reducing multivalued consensus to binary consensus in synchronous systems with Byzantine failures. In [6], Mostefaoui et al. provide a reduction algorithm in asynchronous systems with crash failures. Our work is a direct improvement of the work in [6].

In the algorithm of [6], every process runs a series of binary consensus instances sequentially to solve multivalued consensus. However, the number of the binary consensus instances needed to solve one multivalued consensus instance is unbounded, and it depends on the message delay among the processes. As long as each proposed value is delayed in reaching at least one process, it is possible that all processes may keep

*This work was supported in part by the National Natural Science Foundation of China Grant 60553001, and the National Basic Research Program of China Grant 2007CB807900,2007CB807901.

running an arbitrarily many number of binary consensus instances without solving the multivalued consensus. Since the actual implementation of binary consensus is usually costly, it is certainly undesirable to invoke an unbounded number of binary consensus instances to solve one instance of multivalued consensus.

In this paper, we provide bounded cost algorithms that solve multivalued consensus using a bounded number of binary consensus instances. In our first algorithm, every process invokes exactly $\lceil \log_2 n \rceil$ number of binary consensus instances to solve one multivalued consensus instance, where n is the number of processes in the system. The idea is to use binary consensus to agree on a $\lceil \log_2 n \rceil$ bit long process identifier bit by bit, such that the decision value is the proposal value of this process. In our second algorithm, every process invokes at most $2\widetilde{k}$ binary consensus instances, where $\widetilde{k}$ is the maximum length of the binary representation of all proposed values in the run. The idea is to use binary consensus to agree on the decision value directly bit by bit, but we need to address the issue of when to terminate the binary consensus instances since processes do not know in advance the number of bits they need to agree on. The second algorithm is adaptive to the actual length of proposed values in a run, and it is more efficient than the first algorithm if we know in advance that the length of proposed values are at most $\lceil \log_2 n \rceil / 2$.

Our paper focuses on bounding the cost for asynchronous systems. In synchronous systems, the algorithm in [6] is better since it only needs one binary consensus instance while our algorithms still needs $\lceil \log_2 n \rceil$ or $2\widetilde{k}$ binary consensus instances.

2 The model and the problem

We consider a distributed system consisting of n processes $\{p_1, p_2, \ldots, p_n\}$. Processes may fail by crashing, i.e., stop taking any actions. We say that a process is *faulty* in a run if it crashes in the run and a process is *correct* if it is not faulty.

The problem to solve is *multivalued consensus*, in which every process proposes a value from an arbitrary range of values, and makes an irrevocable decision on one value. It needs to satisfy the following three properties:

- *Validity*: If a process decides v, then v has been proposed by some process.
- *Uniform Agreement*: No two processes (correct or not) decide differently.
- *Termination*: If all correct processes propose, eventually all correct processes decide.[1]

In this paper, we show how to solve multivalued consensus using *binary consensus*, which is a special form of multivalued consensus in which processes may only propose 0 or 1. Note that in this paper we are interested in the uniform version of consensus, i.e., it satisfies the Uniform Agreement property. In our algorithms, we use *B-Consensus*(v) to represent a binary consensus instance, where v is the proposal value. When there are multiple binary consensus instances used in the algorithm, we use array notation like *B-Consensus*$[k]()$ to differentiate different consensus instances.

The communication primitive we use in this paper is *uniform reliable broadcast* [5], which is also used in [6]. In uniform reliable broadcast, a process broadcasts a value v, which is associated with an attribute *sender*(v) to denote the initiator of the broadcast of v, and eventually correct processes should deliver v. More precisely, it should satisfy the following properties:

[1] An alternative specification may not require that all correct processes propose, and only require that all correct processes that propose eventually decide. It is easy to check that our algorithm satisfies this alternative requirement if the underlying binary consensus satisfies the same requirement.

- *Uniform Integrity*: For any value v, any process (correct or faulty) delivers v at most once, and only if v was previously broadcast by *sender*(v).
- *Validity*: If a correct process broadcasts v, then it eventually delivers v.
- *Uniform Agreement*: If a process (correct or faulty) delivers a value v, then all correct processes eventually deliver value v.

In our algorithms, we use *UR-Broadcast*(v) to denote the uniform reliable broadcast of value v, and *UR-Deliver*(v) to denote the uniform reliable delivery of v.

We do not use any other communication primitives. Therefore, we do not need to further clarify if our model is based on the message-passing model or the shared-memory model. In fact, the uniform reliable broadcast abstraction can be implemented in both models. In the message-passing model with reliable links, it can be implemented using the algorithms in [5]. In the shared-memory model, it can be simply implemented as follows: When a process wants to broadcast v, it writes v into a shared single-writer multi-reader atomic register dedicated to this process, since once v is written, all other processes can read it and it will not be lost. Subsequent broadcasts by the same process can be implemented by piggybacking all values that have been broadcasted by the process together and write into one shared register.

3 Algorithms from binary consensus to multivalued consensus

In this section, we present two algorithms that solve multivalued consensus using binary consensus instances and uniform reliable broadcast. The number of binary consensus instances used in the first algorithm is linear to the length of process identifiers, while the number of binary consensus instances used in the second algorithm is linear to the lengths of proposed values in the run. Therefore, the algorithms can be used for different situations to achieve the best efficiency.

For many variables in our algorithms, we need to operate on both their integer representations and bit string representations. To keep the simplicity and the clarity of the presentation, we use the following conventions in both algorithms. All simple variables in the algorithms, such as i, j, l, and d, are treated as non-negative integers by default. For an integer variable j, we suppose the binary representation of the value of j is $j_m j_{m-1} \dots j_1 j_0$, such that $j_k \in \{0, 1\}, k = 0, 1, \dots, m$, and $j_m = 1$ if $j > 0$ (j_m is the most significant bit and thus is always 1 except when $j = 0$). For any non-negative integer k, we use $j[k]$ to represent the k-th bit of j: When reading $j[k]$, the return value is j_k if $0 \leq k \leq m$ and 0 if $k > m$; when writing a bit b to $j[k]$, the result is changing the integer value of j such that the k-th bit $j[k]$ becomes b and all other bits remain unchanged. We also use $j[k..0]$ to represent the bit string $j[k] \cdot j[k-1] \cdots j[1] \cdot j[0]$. By convention, if $k < 0$, $j[k..0]$ is the empty string. For example, if $j = 101$ (i.e., integer 5), then $j[1] = 0$, $j[2] = 1$, $j[5] = 0$, $j[1..0] = 01$, $j[2..0] = 101$, $j[5..0] = 000101$, and setting $j[3]$ to 1 changes j to 1101 (integer 13).

3.1 Algorithm linear to the length of process identifiers

We first give an algorithm (Figure 1) that uses $\lceil \log_2 n \rceil$ sequential calls to binary consensus instances to solve multivalued consensus. The idea is for the processes to agree on the identifier of a process, and the proposed value of that process would be used as the decision value. To do so, processes use binary consensus instances to agree on the binary representation of the process identifier bit by bit, starting from

Local variables on process p_i:
1 v_i, input proposal value of process p_i
2 *prop*$[0..n-1]$, array storing the proposed values of processes, initially $\perp$ for all entries
3 l, non-negative integer storing the process identifier to be decided, initally 0

Code for process p_i:
4 *UR-Broadcast*(v_i)
5 **wait until** [*prop*$[i] \neq \perp$]
6 $j \leftarrow i$
7 **for** $k = 0$ **to** $\lceil \log_2 n \rceil - 1$
8 $l[k] \leftarrow$ *B-Consensus*$[k](j[k])$
9 **repeat** $j \leftarrow (j+1) \bmod n$
10 **until** [*prop*$[j] \neq \perp$ **and** $l[k..0] = j[k..0]$]
11 **endfor**
12 **return** *prop*$[l]$ /* decide on *prop*$[l]$ */
13 Upon *UR-Deliver*(v) with *sender*$(v) = p_j$:
14 *prop*$[j] \leftarrow v$

Figure 1: Algorithm linear to the length of process identifiers

the least significant bit. The key mechanism in the algorithm is to guarantee that after deciding on a process identifier l, every process also know the proposed value of process p_l.

In the algorithm, v_i is the proposal of process p_i. Each process p_i first uniformly broadcasts its own proposed value (line 4). Whenever p_i uniformly delivers a proposed value from p_j, it stores it into *prop*$[j]$ (lines 13–14). Process p_i waits until it delivers its own proposed value (line 5), then sets variable j to i (line 6). Variable j is the process identifier such that at the beginning of the k-th iteration of the for-loop (lines 7–11), *prop*$[j] \neq \perp$ and $j[k-1..0] = l[k-1..0]$, where variable l stores the process identifier on which all processes should eventually agree. In the for-loop (lines 7–11), processes fill in l as a bit string bit by bit starting from the least significant bit. In the k-th round (i.e., the k-th iterations of the for-loop with $k = 0, 1, \ldots, \lceil \log_2 n \rceil - 1$), process p_i first proposes to the k-th binary consensus instance *B-Consensus*$[k]()$ with the k-th bit of j (line 8). The decision of this instance is used to fill the k-th bit of l. Then p_i waits for a proposed value from a process p_j such that the current identifier l (with bits up to the k-th bit filled) is the same as j, that is, $l[k..0] = j[k..0]$ (line 9– 10). The loop ends when it has gone through all bits of process identifiers. At this point l is the final identifier, and p_i then decide on p_l's proposed value (line 12).

Theorem 1 *The algorithm in Figure 1 solves multivalued consensus problem based on the binary consensus.*

Proof. We show that the algorithm satisfies the Validity, Uniform Agreement, and Termination properties of consensus.

Validity: Suppose process p decides on *prop*$[l]$. First, in the last round of process p, p has already checked in line 10 that *prop*$[j] \neq \perp$ and $l[\lceil \log n \rceil - 1..0] = j[\lceil \log n \rceil - 1..0]$, which means $l = j$ and *prop*$[l] \neq \perp$. Thus p decides on *prop*$[l] = v \neq \perp$. The value of *prop*$[l]$ becomes v only after p uniformly delivers value v with *sender*$(v) = p_l$. By the Uniform Integrity of the uniform reliable broadcast, p_l broadcasts v. According to the algorithm, $v = v_l$ is the proposal value of p_l.

Uniform Agreement: Suppose process p and q (either correct or not) both return some decision value in line 12. Thus, both of them complete all $\lceil \log_2 n \rceil$ rounds without crashing in between. In each round $k = 0, 1, \ldots, \lceil \log_2 n \rceil - 1$, they run the same binary consensus instance *B-Consensus*$[k]()$ to determine the k-th bit of l. By the Uniform Agreement of the binary consensus instances, we know that they have the same

value l after they complete all rounds. Moreover, on both p and q, *prop*$[l]$ can only contain the proposed value v_l from process p_l. Therefore, p and q can only decide on the same value v_l.

Termination: First, for every correct process p_i, by the Validity of the uniform reliable broadcast, after p_i broadcasts v_i in line 4, p_i eventually delivers v_i and sets *prop*$[i]$ to v_i (line 14). Thus, p_i will not be blocked at line 5 forever. Suppose, for a contradiction, that some correct process p does not decide. Then p is blocked forever in some round k.

Let $k \in \{0, 1, \ldots, \lceil \log_2 n \rceil - 1\}$ be the earliest round number in which some correct process is blocked forever. We first show that no correct process can be blocked at the binary consensus instance *B-Consensus*$[k]()$ in line 8. Since no correct process is blocked in the previous round by the definition of k, all correct processes eventually propose to *B-Consensus*$[k]()$. By the Termination property of binary consensus, all correct processes eventually decide in *B-Consensus*$[k]()$. We then show that no correct process can be blocked forever in the repeat-until loop in lines 9–10 in round k. For every correct process p_i, p_i has already run the binary consensus instance *B-Consensus*$[k]()$ to determine bit $l[k]$. By the Validity of binary consensus, some process q must have proposed $l[k]$. If $k > 0$, we consider the $(k-1)$-th round on process q. According to line 10, on process q there exists j_q such that *prop*$[j_q] \neq \perp$ and $l[k-1..0] = j_q[k-1..0]$ and then q proposes $j_q[k]$ to binary consensus instance *B-Consensus*$[k]()$. So $l[k] = j_q[k]$ means $l[k..0] = j_q[k..0]$. If $k = 0$, suppose the identifier of q is j_q. Then according to lines 5–6, process q proposes $j_q[0] = l[0]$, and *prop*$[j_q] \neq \perp$ on process q. So, in both cases, we can find j_q such that *prop*$[j_q] \neq \perp$ on some process q and $l[k..0] = j_q[k..0]$. By the Uniform Agreement of uniform reliable broadcast, eventually on p_i *prop*$[j_q]$ is also non-$\perp$. After this time point, the repeat-until loop in lines 9–10 of round k on process p_i will end since p_i can at least find $j = j_q$ that matches the condition in line 10. So, p_i will not be blocked in this repeat-until loop. Therefore, no correct process is blocked forever in any round k, which means eventually all correct processes must decide. □

It is clear that in the algorithm, every process runs exactly $\lceil \log_2 n \rceil$ binary consensus instances sequentially to decide for the multivalued consensus. It is an improvement comparing to the algorithm in [6], in which the number of sequential binary consensus instances needed is unbounded and is affected by the synchrony and the speed of the uniform reliable broadcast algorithm.

3.2 Algorithm linear to the length of the proposed values

The algorithm in Figure 1 uses binary consensus instances to determine the process identifier bit by bit. Using the similar structure, we can also determine the decision value directly bit by bit. The advantage is that when the bit representations of the proposed values are much shorter than $\lceil \log_2 n \rceil$ (the length of the bit representations of process identifiers), multivalued consensus terminates with less number of invocations to binary consensus instances. The difficulty, however, is that we do not know in advance what are the values to be proposed, and thus it is difficult to set an upper bound on the number of binary consensus instances needed to determine all bits. If we use a large fixed bound based on the number of all possible proposed values, it could be very large or even infinite. To deal with this problem, we use another binary consensus instance to determine if the algorithm should terminate after choosing each bit of the decision value. The resulting algorithm is adaptive in the sense that the number of binary consensus instances needed is linear to the maximum length of proposed values in the run, and it is not related to the total number of possible proposed values in all runs.

Figure 2 shows the algorithm that determines the decision value bit by bit, starting from the least significant bit. In the algorithm, the proposal value v_i of process p_i is a non-negative integer value. Any finite-length string can be encoded by a non-negative integer, so using non-negative integers does not lose

```
Local variables on process p_i:
1  v_i, non-negative integer storing the input proposal value of process p_i
2  prop[0..n − 1], array storing the proposed values of processes, initially ⊥ for all entries
3  d, non-negative integer storing the decision value, initially 0

Code for process p_i:
4  UR-Broadcast(v_i)
5  wait until prop[i] ≠ ⊥
6  k ← 0; j ← i
7  repeat
8     d[k] ← B-Consensus[0][k](prop[j][k])
9     repeat j ← (j + 1) mod n
10    until [d[k..0] = prop[j][k..0]]
11    if d = prop[j] then finish = 1 else finish = 0
12    r ← B-Consensus[1][k](finish)
13    k ← k + 1
14 until r = 1
15 return d   /* decide on d */

16 Upon UR-Deliver(v) with sender(v) = p_j:
17    prop[j] ← v
```

Figure 2: Algorithm linear to the length of the proposed values

the generality of the solution. Similar to the first algorithm, each process first uniformly broadcasts its own proposed value and stores all received proposed values in array *prop*[ ]. Variable d is an integer variable used to store the final decision value. Processes decide the decision value d bit by bit from $d[0]$, $d[1]$, and so on. In round k, i.e., the k-th iteration of the repeat-until loop in lines 7–14 with $k \geq 0$, processes decide on $d[k]$, the k-th bit of d. Process p_i first uses the k-th bit of *prop*$[j]$ to be the proposal of the binary consensus *B-Consensus*$[0][k]()$ (line 8) where j is chosen in the previous round (initially, j is set to p_i's own identifier i). Then process p_i finds a non-empty proposal *prop*$[j]$ such that $d[k..0] =$ *prop*$[j][k..0]$ (lines 9–10), This is to ensure that the final decision value is from one of the proposed values, i.e., to ensure the Validity property of consensus. Finally, process p_i uses another binary consensus instance *B-Consensus*$[1][k]()$ to decide whether or not the procedure should terminate (line 12). The input to this binary consensus instance is 1 if p_i finds that the integer d is the same as *prop*$[j]$, or 0 otherwise (line 11). If the decision of this instance is 1, then the multivalued consensus terminates with the decision d.

Theorem 2 *The algorithm in Figure 2 solves multivalued consensus problem based on the binary consensus.*

Proof. We show that the algorithm satisfies the Validity, Uniform Agreement, and Termination properties of consensus.

Validity: Suppose d is the decision value returned by process p. By line 14, p breaks from the repeat-until loop with $r = 1$ in some round k. This means at least one process q proposes *finish* $= 1$ in the binary consensus instance *B-Consensus*$[1][k]()$ in line 12. So $d =$ *prop*$[j]$ for some j when process q runs line 11. By the Uniform Integrity of the uniform reliable broadcast, some p_j broadcasts *prop*$[j]$, and thus p_j proposes d for its multivalued consensus instance. The Validity property holds.

Uniform Agreement: Suppose process p and q (either correct or not) both return some decision value in line 15. Thus, neither of them crashes or is blocked before making a decision. We first prove that for any $k \geq 0$, if process p runs the binary consensus instance *B-Consensus*$[0][k]()$, process q also runs the binary consensus instance *B-Consensus*$[0][k]()$. If not, there must exist $k' < k$, process q breaks from repeat-until

loop in line 14 after running the binary consensus instance *B-Consensus*$[1][k']()$. Thus, *B-Consensus*$[1][k']()$ returns 1 in line 12. But process p also runs the binary consensus instance *B-Consensus*$[1][k']()$. By the Uniform Agreement property of the binary consensus, process p should get the return value 1 from instance *B-Consensus*$[1][k']()$, which means it should break from the repeat-until loop in line 14 in round $k' < k$, which contradicts to the fact that p runs the binary consensus instance *B-Consensus*$[0][k]()$. So processes p and q run the same set of binary consensus instances *B-Consensus*$[0][k]()$. The k-th bit of the decision value d returned by p and q are decided by the binary consensus instance *B-Consensus*$[0][k]()$, which means that the k-th bit decided by p and q are the same. So decision value returned by process p is the same as the decision value returned by process q. This proves the Uniform agreement property.

Termination: First, by the same argument as in Theorem 1, no correct process p is blocked in line 5. We then prove that no correct process is blocked inside the repeat-until loop (line 7– 14). If not, let k be the earliest round number in which some correct process is blocked forever. We first show that no correct process p_i is blocked at the binary consensus instance *B-Consensus*$[0][k]()$ in line 8. If $k = 0$, then $j = i$ for process p_i. So $prop[j] \neq \perp$ by line 5. If $k > 0$, since process p_i is not blocked in the previous round, $prop[j] \neq \perp$ by line 10. Therefore, all correct processes eventually propose to *B-Consensus*$[0][k]()$. By the Termination property of binary consensus, all correct processes eventually decide in *B-Consensus*$[0][k]()$. We then show that no correct process p_i is blocked forever in the repeat-until loop in lines 9–10 in round k. By the Validity of binary consensus, the decision value $d[k]$ in round k is proposed by some process q. So $d[k] = prop[j_q][k]$ for some j_q on q. If $k > 0$, in the $(k-1)$-th round of process q, according to line 10, we have $d[k-1..0] = prop[j_q][k-1..0]$. So $d[k..0] = prop[j_q][k..0]$ and $prop[j_q] \neq \perp$ on process q. If $k = 0$, suppose the identifier of q is j_q. Then we have $prop[j_q] \neq \perp$ and $d[0..0] = prop[j_q][0..0]$ on process q. In both cases, we have $prop[j_q] \neq \perp$ on process q and $d[k..0] = prop[j_q][k..0]$. By the Uniform Agreement of uniform reliable broadcast, eventually on p_i, $prop[j_q]$ is also non-$\perp$. After this time point, the repeat-until loop in lines 9–10 of round k on process p_i will end since p_i can at least find $j = j_q$ that matches the condition in line 10. So, p_i will not be blocked in this repeat-until loop. Therefore, all correct processes eventually propose to *B-Consensus*$[1][k]()$. By the Termination property of binary consensus, all correct processes eventually decide in *B-Consensus*$[1][k]()$. Therefore, no correct process is blocked forever in any round k.

Finally, let $|v_i| = 1$ if $v_i = 0$, and $|v_i| = \max\{j \mid v_i[j] = 1\} + 1$, i.e. $|v_i|$ is the length of the binary representation of v_i. Let $\widetilde{k} = \max\{|v_i| \mid i = 1, 2, \ldots, n\}$, i.e., $\widetilde{k}$ is the maximum length of the binary representations of all proposed values. We prove that for any process p that finishes round $\widetilde{k} - 1$, p must decide at the end of this round and terminate the multivalued consensus. When p executes line 11 in round $\widetilde{k}-1$, we have $d[\widetilde{k}-1..0] = prop[j][\widetilde{k}-1..0]$ for some j according to line 10. By the definition of $\widetilde{k}$, $prop[j]$ has no bit higher than the $(\widetilde{k}-1)$-th bit that is 1, so $d = prop[j]$. Therefore, process p can only propose $finish = 1$ to the binary consensus instance *B-Consensus*$[1][\widetilde{k}-1]()$ according to line 11. Since no process can propose 0 to *B-Consensus*$[1][\widetilde{k}-1]()$, by the Validity of binary consensus the decision of this instance can only be 1. Therefore p will decide at the end of round $\widetilde{k}-1$ and terminates. Since we know that all correct processes will reach the end of round $\widetilde{k}-1$ if they have not decided earlier, we know that all correct processes eventually decide by the end of round $\widetilde{k}-1$. □

From the proof of Termination, it is clear that the algorithm in Figures 2 solves multivalued consensus with at most $2\widetilde{k}$ sequential calls to binary consensus instances, where $\widetilde{k}$ is the maximum length of the binary representation of all proposed values in the run. Therefore, when $\widetilde{k} < \lceil \log_2 n \rceil / 2$, this algorithm is preferred over the algorithm in the previous section.

4 Conclusion

In this paper we present two bounded cost algorithms that solve multivalued consensus problem using binary consensus instances and uniform reliable broadcast primitives. In the first algorithm every processes invokes $\lceil \log_2 n \rceil$ number of binary consensus instances where n is the number of processes, while in the second algorithm every processes invokes $2\widetilde{k}$ binary consensus instances, where $\widetilde{k}$ is the maximum length of the binary representation of all proposed values in the run. Both algorithms significantly improve the worst-case cost of the previous algorithm in [6] in which processes may invoke an unbounded number of binary consensus instances. Finally, we remark that, even though our definition of consensus is for deterministic algorithms, our results can be certainly extended to include randomized algorithms. Therefore, we can directly use the randomized binary consensus algorithms in [1, 3] to solve randomized multivalued consensus.

References

[1] M. K. Aguilera and S. Toueg. Failure detection and randomization: A hybrid approach to solve consensus. *SIAM Journal on Computing*, 28(3):890–903, 1998.

[2] M. K. Aguilera and S. Toueg. A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. *Information Processing Letters*, 71(3-4):155–158, 1999.

[3] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, Aug. 1983.

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

[5] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report 94-1425, Department of Computer Science, Cornell University, Ithaca, New York, May 1994.

[6] A. Mostefaoui, M. Raynal, and F. Tronel. From binary consensus to multivalued consensus in asynchronous message-passing systems. *Information Processing Letters*, 73(5-6):207–212, 2000.

[7] R. Turpin and B. A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.