Information Processing Letters 110 (2009) 13–19

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Implementing uniform reliable broadcast with binary consensus in systems with fair-lossy links

Jialin Zhang [a,*,1], Wei Chen [b]

[a] Tsinghua University, China
[b] Microsoft Research Asia, China

A R T I C L E   I N F O

Article history:
Received 31 October 2008
Received in revised form 5 June 2009
Accepted 29 September 2009
Available online 1 October 2009
Communicated by A.A. Bertossi

Keywords:
Distributed computing
Fault tolerance
Binary consensus
Uniform reliable broadcast

A B S T R A C T

When implementing multivalued consensus using binary consensus, previous algorithms assume the availability of uniform reliable broadcast, which is not implementable in systems with fair-lossy links. In this paper, we show that with binary consensus we can implement uniform reliable broadcast directly in systems with fair-lossy links, and thus the separate assumption of the availability of uniform reliable broadcast is not necessary. We further prove that, when binary consensus instances are available only as black boxes, any implementation of uniform reliable broadcast in the fair-lossy link model requires the invocation of an infinite number of binary consensus instances even if no process ever broadcasts any messages, and this is true even when multivalued consensus is used.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

Consensus is a fundamental problem to solve in building fault-tolerant distributed systems. In the consensus problem, each process in the distributed system proposes one value and eventually all processes decide on one of the proposed values, and the decision is irrevocable. The consensus problem characterizes the distributed agreement that is seen in many distributed coordinating tasks such as atomic broadcast, data replication, mutual exclusion, atomic commit, and thus it serves as the basic building block in achieving these tasks.

One basic form of consensus is *binary consensus*, in which the proposed values are either 0 or 1. Binary consensus is used in studying both impossibility and lower bound results (e.g., [2,8]) and consensus algorithms (e.g., [1,4]). However, solving the general multivalued consensus using binary consensus is not trivial. In [12], Turpin and Coan provide an algorithm reducing multivalued consensus to binary consensus in synchronous systems with Byzantine failures. In [11], Mostefaoui et al. provide a reduction algorithm in asynchronous systems with crash failures, and in [15] we provide new reduction algorithms that bound the number of binary consensus invocations.

In this paper we resolve one important issue left in [11, 15]. The algorithms in both papers rely on the availability of uniform reliable broadcast (URB) primitives in the system. While URB is implementable in shared-memory systems, or in message-passing systems with reliable links, or in message-passing systems with fair-lossy links and a majority of non-faulty processes, it is not implementable in message-passing systems with fair-lossy links and an arbitrary number of faulty processes. Informally, a fair-lossy link is one that may drop messages but if a message is sent infinitely often through the link, then the receiver eventually receives it. Fair-lossy link model is more realistic, especially in wide-area networks in which messages do get lost from time to time. In [3], Aguilera et al. show that the weakest failure detector solving URB in systems with fair-lossy links is Θ (which means that URB is not implementable). Thus, the question is whether URB is imple-

* Corresponding author.
E-mail addresses: zhanggl02@mails.tsinghua.edu.cn (J. Zhang), weic@microsoft.com (W. Chen).

[1] 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.

0020-0190/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.ipl.2009.09.013

mentable in systems with fair-lossy links and an arbitrary number of faulty processes, when binary consensus is also available. In [11,15], the authors simply assume that in the fair-lossy link model URB is available or equivalently failure detector Θ is available when solving multivalued consensus using binary consensus.

In this paper, we show that the assumption on the availability of URB or failure detector Θ is not necessary. Instead, we show that binary consensus can implement URB directly in the fair-lossy link model with an arbitrary number of crash failures. People familiar with failure detector researches may notice that solving consensus implies the availability of a quorum failure detector Σ [7], which is stronger than Θ [6]. However, we cannot use the above reasoning directly, because when showing that consensus implies failure detector Σ, the first step is to show that consensus can implement shared registers, but such implementations require multivalued consensus (to the best of our knowledge). Therefore, in this paper, we provide a direct implementation of URB from binary consensus in the fair-lossy link model. Moreover, we prove a necessity result on any such implementations, when binary consensus instances are available only as black boxes: any such implementation requires the invocation of an infinite number of binary consensus instances on all correct processes even if no process actually broadcasts any messages, and this result still holds even if we use multivalued consensus instead of binary consensus. In our full technical report [14], we further provide a more efficient algorithm that solves URB using multivalued consensus, and show that both algorithms actually implements a stronger form of broadcast called strong uniform atomic broadcast.

2. The model and the problem

We consider a message-passing system consisting of n processes $\Pi = \{p_1, p_2, \ldots, p_n\}$. We assume that global time takes non-negative integer values, but it is not accessible to processes. Processes may fail by crashing, i.e., stop taking any actions. A failure pattern F is a function from a global time t to a subset of processes that have crashed by time t. A crashed process does not recover, i.e., $F(t) \subseteq F(t')$ for all $t \leqslant t'$. We say that a process p is *faulty* (in a failure pattern F) if it crashes in F (i.e., there exists a time t such that $p \in F(t)$), and p is *correct* if it is not faulty. In this paper, we only consider the failure patterns that contain at least one correct process. There is a fair-lossy link between each pair of the processes so that processes can use the links to communicate with each other. We say a link is *fair-lossy* if it satisfies the following properties [3]:

- *Fairness*: If a correct process p sends a message m to a correct process q an infinite number of times, then q eventually receives m from p.
- *Uniform Integrity*: If q receives a message m from p, then p previously sent m to q; and if q receives m infinitely often from p, then p sends m infinitely often to q.

A distributed algorithm $\mathcal{A}$ consists of n deterministic automata, one for each process. Processes run the algorithm step by step. In one step, one process p may receive a message m (or not receiving a message), make a local state transition according to its automaton, its current state, and the message received, and it may send a message to one process. A step is taken at one time point (not accessible to processes), and one process can take at most one step at any time point. A *partial run* of algorithm $\mathcal{A}$ is a finite sequence of steps that is well-formed, i.e., if p receives m from q in a step, q must have sent m to p in an earlier step. A partial run ρ is *compatible* with a failure pattern F if for each step t, the process that does the operation in this step is not contained in $F(t)$. A *run* of the algorithm is a sequence of an infinite number of steps together with a failure pattern such that (a) all correct processes take an infinite number of steps; (b) a crashed process does not take any more step after it crashes (according to the failure pattern); and (c) the send and receive primitives on all links satisfy the fair-lossy link properties.

The problem to solve is *uniform reliable broadcast* (URB) in which a process can 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. We assume that all values broadcast by all processes are different (e.g., they can be differentiated by process identifiers and local sequence numbers). More precisely, uniform reliable broadcast should satisfy the following properties [9]:

- *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 this paper, we show how to solve uniform reliable broadcast using fair-lossy link and *binary consensus*. The binary consensus is a special form of general consensus in which processes can only propose 0 or 1, and make 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 a correct process proposes, it will decide eventually.

In our algorithms, we use *FL-Send*() and *FL-Receive*() to represent the send and receive primitives on fair-lossy links, *UR-Broadcast*() and *UR-Deliver*() to represent uniform reliable broadcast and delivery primitives, and *B-Con*() to represent a binary consensus instance such that the parameter of *B-Con*() is the proposal while the return value is the decision value. We use array notation *B-Con*[ ] to differentiate different binary consensus instances.

```
Local variables on process $p$:
1  $M$, set of known broadcast values, initially empty
2  $D$, set of delivered values, initially empty
3  $\ell$, non-negative integer, initially 0
4  $r$, 0 or 1, representing the result of current binary consensus instance

Code for process $p$:
5  **Task 1:** To execute *UR-Broadcast*$(v)$:
6    $M \leftarrow M \cup v$
7  **Task 2:** Upon *FL-Receive*$(v)$:
8    $M \leftarrow M \cup v$
9  **Task 3:** Repeat periodically:
10   **for** any value $v \in M \setminus D$, *FL-Send*$(v)$ to all processes
11 **Task 4:** Repeat periodically:
12   **for** $i \leftarrow 0$ to $\ell$ **do**
13     **if** $i \notin D$ **then**
14       **if** $i \in M \setminus D$ **then** $r \leftarrow$ *B-Con*$[\ell][i](1)$
15         **else** $r \leftarrow$ *B-Con*$[\ell][i](0)$
16       **if** $r = 1$ **then**
17         $D \leftarrow D \cup \{i\}$
18         *UR-Deliver*$(i)$
19   **endfor**
20   $\ell \leftarrow \ell + 1$
```

Fig. 1. Implementation of uniform reliable broadcast using binary consensus.

3. Implementation of URB using binary consensus

We present an algorithm in Fig. 1 that solves uniform reliable broadcast using binary consensus instances and fair-lossy links. In the algorithm, the values broadcast by processes are non-negative integers. Any finite-length values (including metadata fields such as sender identifiers) can be encoded by a non-negative integer, so using non-negative integers does not lose the generality of the solution.

Each process p maintains two sets M and D, where M contains all values submitted for uniform reliable broadcast that p is aware of, and D contains all values that p has *UR-Deliver*ed. When a process p wants to uniform reliable broadcast a value v, it puts v into a set of values M (Task 1). If process p *FL-Receive*s a value v from other processes, p also puts this value v into M (Task 2). Process p periodically *FL-Send*s all of the values in the set $M \setminus D$ to other processes (Task 3). Process p also periodically runs Task 4 with a counter ℓ that is incremented every time the task runs. In Task 4, for every $i = 0, 1, \ldots, \ell$ such that i has not been *UR-Deliver*ed by p yet ($i \notin D$), p does the following. Process p invokes a binary consensus instance *B-Con*$[\ell][i]$ either with proposal 1 if $i \in M \setminus D$, which means that some process has broadcast i but p has not delivered it yet (line 14), or with proposal 0 (line 15). If the result of instance *B-Con*$[\ell][i]$ is 1, p *UR-Deliver*s i and add i into set D (lines 17–18). The following theorem shows that the implementation in Fig. 1 satisfies all the properties of uniform reliable broadcast.

Theorem 1. *The algorithm in Fig.* 1 *implements uniform reliable broadcast using binary consensus instances in a system with fair-lossy links.*

Proof. We first notice that no process will be blocked forever in the algorithm. The only place where a process p may be blocked is an invocation of a binary consensus instance *B-Con*$[\ell][i]$ for some ℓ and i. By the Termination property of consensus, if p is correct, then eventually the consensus instance will return the decision value.[2] Next we show that the algorithm satisfies the Uniform Agreement, Uniform Integrity, and Validity properties of uniform reliable broadcast.

Uniform Agreement: If a process p *UR-Deliver*s value v, it must have a binary consensus instance *B-Con*$[\ell][v]$ that returns 1 for some ℓ. For any correct process q, when it runs procedure from line 12–19 with ℓ, if q has not delivered v, then q must invoke consensus instance *B-Con*$[\ell][v]$, and the return value must be 1 according to the Uniform Agreement property of binary consensus. Thus, q will *UR-Deliver* it in line 18. So Uniform Agreement property holds.

Uniform Integrity: When process p *UR-Deliver*s value v, we have $v \in D$ by line 17. Then, by the condition in line 13, process p will never deliver it again. So any process can *UR-Deliver* any value at most once. If process p *UR-Deliver*s value v in line 18, then the binary consensus instance *B-Con*$[\ell][v]$ returns 1 for some ℓ. So some process q proposes 1 to this instance. The only case to propose 1 is in line 14 which means $v \in M \setminus D$ in process q at that time. By the Uniform Integrity property of fair-lossy links, we know that all values in M are previously broadcast by some process. This proves the Uniform Integrity property.

Validity: We prove it by contradiction. Suppose value v is *UR-Broadcast* by some correct process p but never *UR-Deliver*ed by p. Since we have proven the Uniform Agreement property of the uniform reliable broadcast algorithm, we know that no process ever *UR-Deliver* v in the run. Since v is never *UR-Deliver*ed by process p, v is permanently in $M \setminus D$ on process p after p *UR-Broadcast*s it. Thus, process p will *FL-Send* v to all other processes infinitely often in Task 3. By the Fairness property of fair-lossy links, every correct process q will eventually *FL-Receive* value v and add v into its set M. Since no process will ever *UR-Deliver*ed v, v will never be in set D on any process. So there exists a time point after which all correct processes have $v \in M \setminus D$ and all faulty processes have crashed. At that time point, suppose $l \geqslant v$ is the smallest integer that binary consensus *B-Con*$[\ell][v]$ has never been called by any process. Then, for any correct process p, it will invoke consensus instance *B-Con*$[\ell][v]$ because $v \in M \setminus D$ and no process is blocked anywhere in the algorithm. When a correct process p invokes the binary consensus instances *B-Con*$[\ell][v]$, it will propose 1 by line 14. Thus, *B-Con*$[\ell][v]$ must return 1 by the Validity property of binary consensus. Therefore, p will *UR-Deliver* v by line 16 and 18. This contradicts that v is never *UR-Deliver*ed by any process. So Validity holds. □

Theorem 1 shows that the algorithm in Fig. 1 implements uniform reliable broadcast. This result together with the results in [11,15] is enough to show that using binary

[2] Even if we use a weaker termination property that only requires decision when all correct processes propose, it is still true that no process will be blocked at any binary consensus instance. This can be proven by an induction on the sequence of consensus instances invoked.

consensus instances alone can solve multivalued consensus in systems with fair-lossy links.

4. Necessity of infinite number of invocations of consensus instances

In the previous section, we give an algorithm that uses binary consensus instances to implement uniform reliable broadcast. One issue of the algorithm is that each process needs to call an infinite number of binary consensus instances, even if the number of the values broadcast by all processes is finite (or even zero). In this section, we show that this is necessary, provided that binary consensus instances are black boxes to the algorithm that uses them. This is still true even if we use multivalued consensus instead of binary consensus. The intuition is that if a faulty process delivers a value, uniform reliable broadcast requires that all correct processes deliver the same value, but the fair-lossy links may drop all messages sent by the faulty process. Therefore, the only "reliable way" for the algorithm to transfer information from faulty processes to correct processes is through (uniform) consensus instances, and correct processes have to continuously invoking consensus instances even though they do not broadcast any values themselves.

Formalizing the above intuition into a proof is nontrivial, however, because we need to model carefully what a consensus black box can or cannot do. We now provide additional model details needed in the proof of the necessary condition. We use shared memory objects to model consensus instances as black boxes. In a step of the algorithm, a process does *one* of the following actions: (a) receives a message (possibly a null message) and sends a message, or (b) invokes a one-shot multivalued consensus object (MC-object for short), i.e. proposes a value to the MC-object, or (c) receives the decision value from an MC-object. After the invocation, the MC-object will return a decision value, which is handled by a process in a later step. Processes are allowed to execute other steps (i.e. parallel tasks) between the invocation and the return of an MC-object. Every MC-object is one-shot, meaning that each process can invoke an MC-object at most once. MC-objects satisfy the specification of multivalued consensus.

Each step of the algorithm is thus fully determined by parameters (p, Σ_p, m, d, O), where p denotes which process to take the step, Σ_p denotes the current local state of p, m denotes the message received in the step, and d denotes the decision value received, and O is the MC-object from which d is received. Parameter m could be $\bot$, meaning that no message is received in the step, and (d, O) could also be $(\bot, \bot)$, meaning that no consensus decision is received in the step. If $(m, d, O) = (\bot, \bot, \bot)$, it denotes a local step. A process only invokes an MC-object O with some proposed value v in a local step, in which case O and v are specified in the local state Σ_p and thus no need to be specified separately. Parameters m and d cannot be non-$\bot$ in the same step, meaning that receiving a message and receiving a decision do not occur in the same step.

In this model, MC-objects are black boxes, which means that uniform reliable broadcast algorithms can only access these instances through their interfaces (proposing a value and receiving a decision), and algorithms cannot access or modify the implementations of consensus (e.g., reading the internal states of the implementation or piggybacking messages onto messages in the implementation). For our proof, we need to determine whether the system environment (also referred to as the adversary or the scheduler in the literature) can select the decision value of an MC-object among its proposals and whether the environment can entirely determine the real time at which the decision is returned. It would make our proof easier if the environment can do both freely in all runs. However, real implementations of consensus may result in different behaviors in different runs. For example, an implementation may use a failure detector, which generates different output in different failure patterns, resulting in different decision values and computation time in different runs. To consider such realistic situations and make our result stronger, we do not assume that the environment can control the decision value or the computation time of an MC-object. However, the environment can further delay the delivery of the decision value of an MC-object to a process, which is consistent to the asynchrony assumption that allows the environment to delay message delivery to a process or delay a step of a process.

We only consider MC-objects whose behaviors are not affected by potential failures in the future. More precisely, we say that MC-objects are *realistic* if in any run R of these objects with failure pattern F, for any time t and any failure pattern F' satisfying $F(t') = F'(t')$ for all $t' \leqslant t$, there exists a run R' with F' such that all steps of R' by time t' are the same as the steps in R and they occur at the same real time points as the steps in R. All MC-objects considered in the following theorem and its proof are realistic.

In a partial run ρ, an MC-object O is *pending* in ρ if some process p invokes O in ρ but no process receives decision value from O in ρ. A partial run ρ is said to be *unambiguous* if no MC-object is pending in ρ. Let p be the process that executes the last step in partial run ρ. Let $\rho = \rho' \cdot \rho_p$, where ρ_p is the sequence of steps all executed by p and ρ' is either empty or its last step is executed by some process $q \neq p$. We call it *last-step decomposition* of partial run ρ. The partial run ρ is said to be *strongly unambiguous* if both ρ and ρ' are unambiguous. The empty partial run is both unambiguous and strongly unambiguous. In a failure pattern F, if there exists only one correct process p in F, let t be the time such that for any $t' \leqslant t$, $\Pi \setminus F(t')$ contains at least two processes and for $t' > t$, $F(t') = \Pi \setminus \{p\}$. We call t the *last crash time* and any process q in $\Pi \setminus F(t)$ different from p the *last crash process*.

The following theorem proves that infinite MC-objects are necessary to implement uniform reliable broadcast.

Theorem 2. *In the fair-lossy model, for any algorithm $\mathcal{A}$ that implements uniform reliable broadcast using MC-objects, for any failure pattern F, for any strongly unambiguous partial run ρ compatible with F of $\mathcal{A}$, there exists a run R with failure pattern F such that*

(1) *ρ is the initial sequence of R;*
(2) *no process broadcasts any value after ρ; and*

(3) *every correct process invokes an infinite number of MC-objects. Moreover, if there are at least two correct processes in F, we only require ρ to be unambiguous instead of being strongly unambiguous.*

In the special case when ρ is the empty sequence, the theorem shows that for any failure pattern F, there exists a run with F in which no process ever broadcasts any value but every correct process invokes infinitely many MC-objects. In order to prove this theorem, we first prove the following two lemmata. We say that a partial run ρ' is an extension of a partial run ρ if ρ (as a sequence of steps) is a prefix of ρ', and we write $\rho' = \rho \cdot \rho''$ where ρ'' is the sequence of additional steps after ρ.

Lemma 1. *In the fair-lossy model, consider the failure pattern F that has only one correct process p. Let q be any last crash process. For any algorithm $\mathcal{A}$ that implements uniform reliable broadcast using MC-objects, consider a strongly unambiguous partial run ρ of $\mathcal{A}$ compatible with F. Consider another failure pattern F' which is the same as F except that q is correct in F'. We prove that there exists a run R with failure pattern F' where the partial run ρ is a prefix of R.*[3]

Proof. Let run R_1 be a run beginning with partial run ρ and the failure pattern is F. Let t be the last crash time. We construct run R with failure pattern F' as follows.

First, since the failure patterns F and F' before time t are exactly the same and all consensus objects are realistic, we can schedule the steps in R by time t to be exactly the same as those in R_1. If the last step of ρ is executed by time t, we have already finished the proof. Otherwise, the last step of ρ is executed by process p since only p is alive after time t. Let the last-step decomposition of ρ to be $\rho = \rho' \cdot \rho_p$. We know that ρ' is unambiguous because ρ is strongly unambiguous. Since the last step of ρ' is executed before time t, it can be scheduled in R exactly the same as in R_1. Then, after time t, we want process p in R to take the same sequence of steps in ρ_p as in R_1. We cannot simply schedule all steps in ρ_p in R exactly as those in R_1, since the environment does not control the decision values or the computation times of the MC-objects. We now provide detailed explanation on how to schedule ρ_p in R.

Let $\rho_p = s_1 \cdot s_2 \cdots s_k$, where s_i is the i-th step in ρ_p for $i = 1, 2, \ldots, k$. Suppose that $s_1, \ldots, s_{i-1}$ have been scheduled in R and we now need to schedule s_i. Thus we know that after s_{i-1} process p is in the same state as in R_1. Let $s_i = (p, \Sigma_{p,i}, m_i, d_i, O_i)$. We consider the following cases.

- *Case 1*: $(m_i, d_i, O_i) = (\perp, \perp, \perp)$. Step s_i is a local step. Since p has the same state $\Sigma_{p,i}$ as in R_1, we can schedule the same step s_i in R.
- *Case 2*: $m_i \neq \perp$. In s_i, p receives a message m_i from some process q'. Since q' must have sent m_i to p in an earlier step in R_1, and all steps in R prior to s_i are the same as in R_1, p may also receive the same message m_i and execute the same step s_i in R.
- *Case 3*: $(d_i, O_i) \neq (\perp, \perp)$. In s_i, p receives consensus decision d_i from MC-object O_i. This implies that p must have proposed to O_i in an earlier step. By the Termination property of consensus, p should eventually receive a decision from O_i. We first argue that p will receive the same decision d_i in R. There are two cases to consider. If O_i has been invoked by some process in ρ', then since ρ' is unambiguous, some process q' must have received decision d'_i from O_i in ρ' of R_1. By the Uniform Agreement property of consensus, $d'_i = d_i$. Then since ρ' is also the partial run of R, the decision value of O_i must also be $d'_i = d_i$, and thus p will receive d_i as the decision from O_i. If O_i is not invoked by any process in ρ', then p must have proposed some value v_i to O_i in some step s_j of R_1 with $1 \leqslant j < i$. Since only p takes steps in ρ_p, by the Validity property of consensus, $d_i = v_i$. Therefore, in R, p also proposes v_i in s_j and p must receive $v_i = d_i$ as the decision from O_i. We now argue that we can schedule s_i before scheduling any other later steps of p in ρ_p. Even though O_i may take longer real time to compute in R than in R_1, we can still schedule step s_i by delaying all later steps in ρ'', according to the asynchrony assumption of the system. Hence, s_i can be scheduled next in R when $s_i = (p, \Sigma_{p,i}, \perp, d_i, O_i)$.

Therefore, we know that in R we can schedule steps in ρ_p in the same order as in R_1. Suppose ρ_p ends at time t' in R. After time t', we schedule the steps of p and q to run in a round-robin way and let them receive all messages and all consensus decisions (in particular, q receives all messages p sent in ρ_p).

Thus, we construct a run R compatible with failure pattern F' and the partial run ρ is a prefix of R. □

The above lemma proves an important property of the model: any strongly unambiguous partial run obtained in a run with a single correct process can also be obtained in another run with an additional correct process, despite that the two runs have different failure patterns and thus the MC-objects may not behave exactly the same in two runs. The condition of partial run being strongly unambiguous is crucial for this property, because it enforces the MC-objects to have the same decision values in the two runs.

Lemma 2. *In the fair-lossy model, for any algorithm $\mathcal{A}$ that implements uniform reliable broadcast using MC-objects, for any failure pattern F, for any strongly unambiguous partial run ρ compatible with F, and for any correct process p in F, there exists a partial run ρ' such that*

(1) *$\rho' = \rho \cdot \rho''$ is an extension of ρ (compatible with F);*
(2) *only process p takes steps in ρ'';*
(3) *p does not broadcast any value in ρ''; and*
(4) *p invokes at least one MC-object in ρ''.*

Proof. Suppose, for a contradiction, that there exists a failure pattern F, a strongly unambiguous partial run ρ com-

[3] Note that in R the exact real time to execute each step in ρ may be different from the real time in the original run that generates ρ.

patible with F, a correct process p in F, such that for all the partial run extensions $\rho' = \rho \cdot \rho''$ of ρ in which only p takes steps in ρ'' and p does not broadcast any value in ρ'', p does not invoke any MC-object in ρ''. We first extend ρ to obtain $\rho_0 = \rho \cdot \rho'_0$, such that (a) in ρ_0 p receives all decision values from all objects that p proposes to in ρ, and (b) all steps in ρ'_0 are decision-receiving steps of p. Item (a) above can be achieved due to the Termination property of consensus, while item (b) can be achieved since by asynchrony assumption we can delay all other steps in the system. Let t_1 be the real time at which the last step of ρ_0 is executed.

Consider a full run R_1 constructed as follows. First, the failure pattern F_1 of run R_1 is such that (a) $F_1(t) = F(t)$ for all $t \leqslant t_1$, (b) p is correct in F_1, and (c) all other processes not crashed yet by time t_1 crash at time $t_1 + 1$. Second, the sequence of steps by time t_1 is exactly ρ_0, which is possible because all MC-objects are realistic. Third, after time t_1, p does not broadcast any values. Let $R_1 = \rho_0 \cdot \rho_p$, where ρ_p is the sequence of steps of p after time t_1. By the selection of ρ, it is easy to check that p does not invoke any MC-object in ρ_p. Moreover, by the construction of ρ_0 p does not receive any decision from any MC-object in ρ_p either.

Let $q \neq p$ be any last crash process in F_1. By Lemma 1, we can construct a run R_2 such that the failure pattern F_2 of run R_2 is the same as F_1 except that q is correct in F_2 and ρ_0 is a prefix of R_2. Suppose that the last step of ρ_0 in R_2 is executed at time t_2, which may be different from t_1.

We then construct a run R_3 with failure pattern F_3. (1) The failure pattern F_3 is the same as F_2 except that p crashes at time $t_2 + 1$. (2) The sequence of steps by time t_2 is exactly ρ_0, which can be done since the failure pattern by time t_2 is exactly the same in both run R_2 and R_3. (3) At time $t_2 + 1$, process q broadcasts a value v (by our definition v is different from any values broadcast in partial run ρ_0). Since q is a correct process, by the Validity property of uniform reliable broadcast, q eventually delivers v in R_3. Let the partial run until q delivers v be $\rho_0 \cdot \rho'_q$, where all steps in ρ'_q are executed by process q. Sequence ρ'_q may include invocations and returns of a finite number of MC-objects. We extend the partial run $\rho_0 \cdot \rho'_q$ so that q receives the decision values from all objects invoked in ρ'_q, but we delay all other steps so that all steps in the extension after $\rho_0 \cdot \rho'_q$ are decision-receiving steps. Let $\rho_0 \cdot \rho_q$ be this extension. Then since ρ_0 is unambiguous and in ρ_q q receives all decision values from objects that q proposes to, we know that $\rho_0 \cdot \rho_q$ is strongly unambiguous. Suppose that the last step in ρ_q is executed at time $t_3 > t_2$.

In failure pattern F_3, the only correct process is process q, and p is a last crash process. Thus by Lemma 1, we can construct a run R_4 such that the failure pattern F_4 of run R_4 is the same as F_3 except that p is correct in F_4 and $\rho_0 \cdot \rho_q$ is a prefix of R_4. Suppose that the last step of $\rho_0 \cdot \rho_q$ in R_4 is executed at time t_4, which may be different from t_3.

Finally, we construct a run R_5 as follows. The failure pattern F_5 of R_5 is the same as F_4 except that q crashes at time $t_4 + 1$. The sequence of steps in R_5 by time t_4 is $\rho_0 \cdot \rho_q$, exactly as in R_4, which is possible because all MC-objects are realistic. After time $t_4 + 1$, we schedule all steps in ρ_p in the same order as in run R_1 (but perhaps at different real time points). We can do so because (a) all messages sent by q in ρ_q can be dropped since q is faulty in R_5, (b) the state of p in R_5 at time $t_4 + 1$ is the same as the state of p in R_1 at time $t_1 + 1$, and (c) ρ_p contains only message-receiving steps (or local steps when the message is null).

However, in $R_5 = \rho_0 \cdot \rho_q \cdot \rho_p$, p does not deliver value v since v is a new value only broadcast in ρ_q by q, but q delivers v in ρ_q. This violates the Uniform Agreement property of uniform reliable broadcast — a contradiction. Therefore, the lemma holds. □

We now prove Theorem 2 by repeatedly applying Lemma 2.

Proof of Theorem 2. We first consider a strongly unambiguous partial run ρ compatible with an arbitrary failure pattern F. Let $p_1, p_2, \dots, p_k$ be the correct processes in F. We construct the run R as follows starting from partial run $\rho_0 = \rho$. First, we schedule p_1 to take enough message-receiving steps to receive all messages sent to p_1 in ρ_0. Let ρ'_0 be the resulting extension of ρ_0. Since ρ_0 is strongly unambiguous and p_1 does not invoke MC-objects in message-receiving steps, ρ'_0 is still strongly unambiguous. Then by Lemma 2, there exists a partial run extension $\rho''_0 = \rho'_0 \cdot \rho'''_0$ such that only p_1 takes steps in ρ'''_0, p_1 does not broadcast any value in ρ'''_0, and p_1 invokes at least one MC-object in ρ'''_0. By the Termination property of consensus, all MC-objects invoked by p_1 in ρ'''_0 eventually return decision values to p_1. We delay all other steps of any process except the steps in which p_1 receives the decision values from all objects invoked in ρ'''. Let ρ_1 be the extension of ρ''_0 when p has received the decision values from all objects invoked in ρ'''. Since ρ_0 is strongly unambiguous and p_1 does not invoke MC-objects in decision receiving steps, we know that ρ_1 is also strongly unambiguous.

We now repeat the same procedure as above on ρ_1 and process p_2. Namely, we first let p_2 receive all messages sent to it in ρ_1, and then apply Lemma 2 to find an extension in which p_2 does not broadcast any value but invokes at least one MC-object, and finally schedule decision receiving steps of p_2 to find an extension ρ_2 that is still strongly unambiguous.

In general, we repeat the above procedure in a round-robin way among all correct processes $p_1, p_2, \dots, p_k$ to construct the infinite run R. In this run, all correct processes execute an infinite number of steps and receive all messages sent to them. After ρ, only correct process take steps, they do not broadcast any more values, but every one of them invokes an infinite number of MC-objects. Therefore, the theorem holds for the arbitrary failure pattern case.

Finally we consider the case where failure pattern F includes at least two correct processes. Let ρ be the unambiguous partial run compatible with F. Let p be the process executing the last step of ρ. Since F includes at least two correct processes, there must be a correct process $q \neq p$. We extend ρ by scheduling q to take the next step and q does not broadcast a value in this step. If q

invokes an MC-object in the next step, we also need to schedule one more step for q to receive the decision value from this MC-object. In other cases, we only need one step from q. Let $\rho' = \rho \cdot \rho''$ be the extension. Then we know that $\rho \cdot \rho''$ is the last-step decomposition of ρ', and ρ' is unambiguous. Since ρ is unambiguous, ρ' is strongly unambiguous. The rest of the proof is the same as above. □

Several remarks are now in order on the subtlety of the theorem. First, the assumption that MC-objects are realistic is necessary. If an MC-object is not realistic, e.g. its implementation uses a failure detector that predicts future failures, then the implementation can guarantee that only correct processes return from consensus and all faulty processes will not receive decision values. We only need one such object to implement uniform reliable broadcast as follows. Each process p invokes the consensus object with an arbitrary proposal. Whenever p broadcasts a value v, it repeatedly sends v to all processes. Only after p receives a decision value from the object (indicating that p is correct in this run), p delivers all values it receives from any process (including itself). Note that agreement in decision values is not needed here.

Second, the theorem relies on the Termination property of consensus that requires a proposed correct process to decide no matter if other correct processes have proposed or not. This requirement is reasonable and is satisfied by many existing consensus algorithms in both the message-passing model and the share-memory model (e.g. [10,13]). If we consider a different Termination property as given below, then the implementation of uniform reliable broadcast does not need repeated invocations of consensus.

- *Termination'*: If all correct processes propose, then eventually they all decide. Conversely, if some process decides, all correct processes must have already proposed.

It is possible to implement consensus with the above property using a perfect failure detector [5], such that every process waits for every other process to either propose or to be deemed as having crashed by the failure detector (which must be true) before proceeding to select the decision value. With objects satisfying Termination', we can implement uniform reliable broadcast as follows. Whenever a process wants to broadcast a value v, it repeatedly sends v to all other processes and invokes an object with an arbitrary proposal and waits for the decision. When other processes receive v, they also invoke the same object with an arbitrary proposal. If a process obtains a decision value from the object, it guarantees that all correct processes receives value v and will eventually obtain the decision value from the object. So it can deliver v safely. The implementation only needs one object for every broadcast value, and agreement among decision values is not needed.

5. Conclusion

In this paper, we show that uniform reliable broadcast can be implemented in systems with fair-lossy links when binary consensus is available, and thus the separate assumption on the availability of uniform reliable broadcast or an equivalent failure detector Θ is unnecessary when implementing multivalued consensus from binary consensus. In our algorithm, every process needs to invoke binary consensus periodically even if there is no message being broadcast, and we prove that this behavior is inevitable.

With this work, we can finally claim that binary consensus indeed has the same power in terms of solvability as multivalued consensus in systems with fair-lossy links. We can then apply results obtained with multivalued consensus case to binary consensus. For example, the weakest failure detector for binary consensus is the same as the weakest failure detector for multivalued consensus in this model.

Acknowledgements

We thank the anonymous reviewers whose insightful comments help improving the paper.

References

[1] M.K. Aguilera, S. Toueg, Failure detection and randomization: A hybrid approach to solve consensus, SIAM Journal on Computing 28 (3) (1998) 890–903.
[2] M.K. Aguilera, S. Toueg, A simple bivalency proof that t-resilient consensus requires $t+1$ rounds, Information Processing Letters 71 (3–4) (1999) 155–158.
[3] M.K. Aguilera, S. Toueg, B. Deianov, Revisiting the weakest failure detector for uniform reliable broadcast, in: Proceedings of the 13th International Symposium on Distributed Computing, September 1999, pp. 19–33.
[4] M. Ben-Or, Another advantage of free choice: Completely asynchronous agreement protocols, in: Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing, August 1983, pp. 27–30.
[5] T.D. Chandra, V. Hadzilacos, S. Toueg, The weakest failure detector for solving consensus, Journal of the ACM 43 (4) (1996) 685–722.
[6] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, Shared memory vs. message passing, Technical Report IC/2003/77, EPFL, December 2003.
[7] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, V. Hadzilacos, P. Kouznetsov, S. Toueg, The weakest failure detectors to solve certain fundamental problems in distributed computing, in: Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing, July 2004, pp. 338–346.
[8] M.J. Fischer, N.A. Lynch, M.S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (1985) 374–382.
[9] V. Hadzilacos, 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.
[10] L. Lamport, The part-time parliament, ACM Transactions on Computer Systems 16 (2) (1998) 133–169.
[11] A. Mostefaoui, M. Raynal, F. Tronel, From binary consensus to multivalued consensus in asynchronous message-passing systems, Information Processing Letters 73 (5–6) (2000) 207–212.
[12] R. Turpin, B.A. Coan, Extending binary byzantine agreement to multivalued byzantine agreement, Information Processing Letters 18 (2) (1984) 73–76.
[13] J. Yang, G. Neiger, E. Gafni, Structured derivations of consensus algorithms for failure detectors, in: Proceedings of the 17th ACM Symposium on Principles of Distributed Computing, June 1998, pp. 297–306.
[14] J. Zhang, W. Chen, Implementing uniform reliable broadcast with binary consensus in systems with fair-lossy links, Technical Report MSR-TR-2008-162, Microsoft Research, October 2008.
[15] J. Zhang, W. Chen, Bounded cost algorithms for multivalued consensus using binary consensus instances, Information Processing Letters 109 (17) (2009) 1005–1009.