The Decentralized Non-Blocking Atomic Commitment Protocol *

Rachid Guerraoui André Schiper
Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
e-mail: [rachid,schiper]@lse.epfl.ch

Abstract

In a transactional system, an *atomic commitment* protocol ensures that for any transaction, all *data managers* agree on the same outcome: *commit* or *abort*. A *non-blocking atomic commitment* protocol enables an outcome to be decided at every correct data manager despite the failure of others. It is well known that the famous *Two Phase Commit* protocol *(2PC)* is a blocking protocol, whereas the *Three Phase Commit* protocol *(3PC)* is a non-blocking protocol. Both protocols are centralized. In this paper, we discuss a Decentralized Non-Blocking Atomic Commitment protocol, named *DNB-AC*. In the absence of failures, the DNB-AC protocol requires the same number of steps to *commit* than the 2PC protocol (three steps). Performance measures show that the DNB-AC protocol is significantly faster that the 3PC and, in the case of a broadcast network, is almost as efficient as the 2PC.

The DNB-AC protocol uses a uniform consensus protocol as a termination protocol. Using the recent results by Chandra and Toueg on solving consensus in an asynchronous system augmented with an unreliable failure detector, we give a precise characterization of the liveness of DNB-AC. We show that DNB-AC is non-blocking when a majority of data managers are correct and the failure detector is $\diamond\mathcal{S}$.

1 Introduction

A transaction is a set of invocations on data objects, characterized by the *concurrency atomicity* and *failure atomicity* properties [2]. Failure atomicity, also called *all-or-nothing* property, is ensured by an *atomic commitment* protocol, launched at the end of the transaction: the atomic commitment protocol returns either *commit* or *abort*. The simplest and best known atomic commitment protocol is the so called *Two Phase Commit protocol (2PC)* [4]. The 2PC is considered as a "cheap" protocol (as it only requires three communication steps to commit). Its main drawback is to be blocking: if the coordinator of the 2PC protocol fails while the participants are uncertain about the *commit/abort* outcome of the transaction, the participants must wait for the recovery of the coordinator. Blocking can be avoided by using the *Three Phase Commit Protocol (3PC)* [7]. In the absence of failures the 3PC has however

*Appears in the IEEE International Symposium on Parallel and Distributed Processing, 1995. Research supported by the "Fonds national suisse" and OFES under contract number 21-32210.91, as part of the ESPRIT Basic Research Project BROADCAST (number 6360), and by PPR-IF under contract number 5003-34344.

a higher latency (i.e. needs more time to commit) than the 2PC: this is due to two additional communication steps between the coordinator and the participants in the 3PC protocol. An interesting question is then the following: is there a non-blocking atomic commitment protocol that has a lower latency than the 3PC protocol?

The answer to this question is yes! We give in the paper a non-blocking atomic commitment protocol, noted DNB-AC (Decentralized Non-Blocking Atomic Commitment), that has a latency lower than the 3PC protocol: as the 2PC, the DNB-AC protocol requires only three communication steps to commit. The basic idea of the DNB-AC protocol, in the absence of failures, was given in 1981 by Skeen [7], but has since been completely ignored. This is probably due to the complexity of handling the failure scenarios in the protocol. Nevertheless, the recent work of Chandra and Toueg, showing how to solve consensus (and uniform consensus) in an asynchronous system augmented by a failure detector [3], defines a rigorous framework in which the DNB-AC protocol can be well and simply expressed.

The rest of the paper is organized as follows. In Section 2, we define the system model, and the non-blocking atomic commitment problem. In Section 3, we present the DNB-AC protocol and prove its correctness by intuitive arguments (the complete proofs are given in the Appendix). In Section 4, we compare the latency of the DNB-AC, 2PC and 3PC protocols, and also the cost in number of messages of the three protocols.

2 Model

2.1 System Model

We consider a distributed system composed of a finite set of processes $S = \{p_1, p_2, \dots, p_n\}$ completely connected through a set of channels. Communication is by message passing, *asynchronous* and *reliable*. Asynchrony means that there is no bound on communication delays. A reliable channel ensures that a message sent by a process p_i to a process p_j is eventually received by p_j, if p_i and p_j are correct (i.e. do not fail)[1]. Processes fail by crashing; we do not consider Byzantine failures.

We assume that our asynchronous system is augmented with a failure detector [3]. A failure detector can be viewed as a distributed oracle. Each process p_i has access to a local failure detector module FD_i. This module maintains a list of processes that it currently *suspects* to have failed. We do not make any assumption on how a failure detector is implemented. Failure detectors are abstractly characterized by their *completeness* and *accuracy* properties [3]. In the sequel, we consider the *Eventually Strong* failure detector, noted $\diamond\mathcal{S}$, which guarantees (1) *strong completeness*: eventually every process that fails is permanently suspected by every correct process, and (2) *eventual weak accuracy*: there is a time after which some correct process is never suspected by any correct process. Notice that the failure detector $\diamond\mathcal{S}$ can make mistakes, i.e false failure suspicions.

Chandra and Toueg have shown that $\diamond\mathcal{S}$ is sufficient to solve consensus and uniform consensus problems in a system, where a majority of processes are correct [3]. In the consensus problem (and in the uniform consensus problem), every process p_i starts with an initial value v_i, and the processes must agree on some decision value v. The uniform consensus problem is defined by the three conditions: (1) *C-Uniform-Agreement* [2]: No two processes (correct

[1] This assumption does not exclude link failures, if we require that any link failure is eventually repaired.

[2] Consensus is defined by the *C-Agreement* condition: No two correct processes can decide differently.

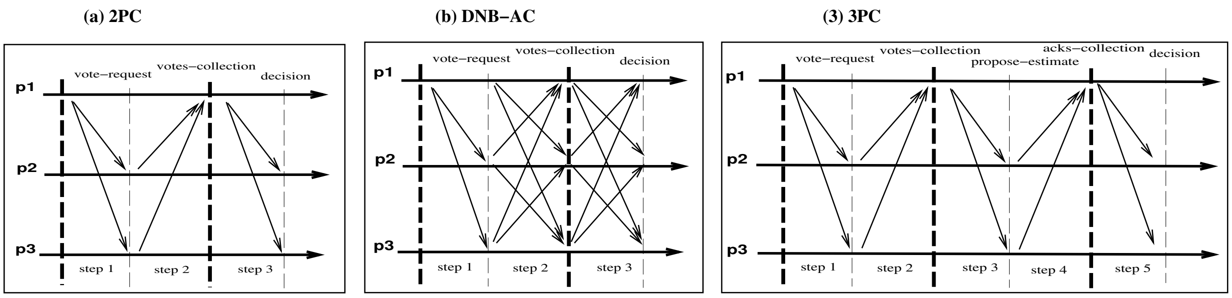

Figure 1: Overview of 2PC, DNB-AC and 3PC

or not) can decide differently; (2) *C-Validity*: the decision value must be the initial value of some process; (3) *C-Termination*: every correct participant eventually decides.

2.2 The Non-Blocking Atomic Commitment Problem

The aim of an atomic commitment protocol is to bring the participants in a transaction to agree on an outcome of the transaction (either *commit* or *abort*). The outcome of the transaction depends on *votes (yes* or *no)* provided by the participants[3]. The outcome can be *commit* only if the votes of all participants are *yes*. In order to exclude trivial situations where participants always decide *abort*, it is generally required that *commit* must be decided if (i) all votes are *yes* and (ii) *there is no failure* [1, 5, 6, 7, 8]. This *Non-Triviality* condition is too strong for an asynchronous system since it requires precise knowledge about failures. We consider a weaker *Non-Triviality* condition where *commit* must be decided if (i) all votes are *yes* and (ii) *no process has ever been suspected* [4]. We define the non-blocking atomic commitment problem by the following conditions: (1) *AC-Uniform-Agreement:* No two participants can decide differently; (2) *AC-Validity: Commit* is decided only if all participants have voted *yes*; (3) *AC-Termination:* Every correct participant eventually decides; (4) *AC-Non-Triviality:* If no participant votes *no*, and no participant has ever been suspected, then the decision must be *commit*.

3 The DNB-AC Protocol

3.1 Overview of DNB-AC protocol

The DNB-AC protocol has three steps (Fig 1 (b)). In step 1, similarly to the 2PC protocol (Fig 1 (a)), the coordinator requires votes from all participants. However, unlike the 2PC, there is no coordinator in the next steps of the *(decentralized)* DNB-AC protocol. In step 2 of the DNB-AC protocol, a participant that votes *yes* sends its vote to all the participants (rather than only to the coordinator in the 2PC and 3PC). In step 3, once a participant has received the *yes* votes from all, it sends a *preCommit* message to all. Finally, when a participant receives *preCommit* from all, it decides *commit*.

This describes the protocol in the "good" case, i.e. when all votes are *yes*, and when no process is suspected to have crashed. If, during step 1, a participant p_i either suspects the

[3] Each vote reflects the ability of the participant to ensure that its data object updates are permanent.

[4] We say that a process p_i is suspected *(on a cut c)*, if it is suspected by at least one process *(on c)*.

coordinator, or votes *no*, then p_i decides *abort* in a unilateral way. The handling of the other cases remains simple, as they can be reduced to a *uniform consensus*:

- during step 2, a participant p_i that suspects any other participant, starts a uniform consensus with *abort* as its initial value (p_i does not know the vote of all participants);
- during step 3, a participant p_i that suspects any other participant, starts a uniform consensus with *commit* as its initial value (p_i is in step 3 and thus knows that all votes are *yes*).

3.2 Description of DNB-AC protocol

The DNB-AC protocol is described by the function *atomicCommitment()* in Figure 2. This function is called by the transaction manager p_1, and by the data managers $p_2, \ldots, p_n$ ($p_1, \ldots, p_n$ are the *participants*). The transaction manager p_1 is the coordinator for the first step of the protocol (p_1 sends the *voteRequest* message to all). We assume that all the participants know that p_1 is the coordinator for step 1. During the DNB-AC protocol, every process p_i is informed by its local failure detector of crash suspicions: the notation $p_j \in \Diamond\mathcal{S}_i$ (e.g line 3) indicates that process p_i suspects process p_j. The function *atomicCommitment()* triggers two concurrent tasks, noted *Task 1* (lines 1 to 24) and *Task 2* (lines 26 to 28), and terminates by the execution of a "**return** v" statement, where v is either *commit* or *abort* (lines 6, 9, 16, 22 and 24). When p_i executes **return** v, it decides on the outcome v of the transaction.

Task 2 waits for a decision message $(p_j, outcome, decide)$, and forwards the *outcome* to the other participants (line 27). This ensures that, if one correct participant decides on an outcome, then all the correct participants also decide on an outcome.

Task 1 executes the main protocol. At line 1, the coordinator sends the *voteRequest* message to all. From there on, the protocol is decentralized: each participant p_i behaves in the same way, and performs the following three steps:

Step 1 (lines 3-10): p_i waits until (i) it receives the message *(voteRequest,step1)* from the coordinator, or (ii) it suspects the coordinator (line 3). In case (ii), p_i sends the message $(p_i, abort, decide)$ to all (line 5), and decides *abort* (line 6). In case (i), p_i defines its vote, and if the vote is *no* (line 7), p_i sends the message $(p_i, abort, decide)$ to all, and decides *abort.* If p_i votes *yes*, it sends the message $(p_i, yes, step2)$ to all (line 10), and proceeds to step 2.

Step 2 (lines 11-17): p_i waits until (i) it receives *yes* votes from all participants, or (ii) it suspects another participant (line 12). In case (ii), p_i asks all the participants to start the uniform consensus by sending the message $(p_i, startCons)$ (line 14), and p_i starts a uniform consensus with *abort* as its initial value (line 15). The decision of the uniform consensus protocol becomes the outcome of p_i (line 16). In case (i), p_i sends the message $(p_i, preCommit, step3)$ to all (line 17), and proceeds to step 3.

Step 3 (lines 18-21): p_i waits until (i) it receives *(-,preCommit,step3)* messages from all, or (ii) it receives a *startCons* message, or (iii) it suspects another participant. In case (ii) and (iii), p_i starts a uniform consensus with *commit* as its initial value (line 21), and the decision of the uniform consensus protocol becomes the outcome of p_i (line 22). In case (i), p_i sends the message $(p_i, commit, decide)$ to all (line 23), and decides *commit* (line 24).

```
function atomicCommitment()

cobegin
|| /* Task 1 */
    if p_i = p_1 then  /* Only the coordinator executes: */
        send (voteRequest,step1) to all ;
    wait until [received (voteRequest,step1) from coord or p_1 ∈ ◇S_i] ;        /* Step 1 */
        if p_1 ∈ ◇S_i then
            send (p_i, abort, decide) to all ;
            return abort ;  /* Abort decision */
        if (unable to make updates permanent) then  /* vote = no */
            send (p_i, abort, decide) to all ;
            return abort ;  /* Abort decision */
    send (p_i, yes, step2) to all participants ;
    for each participant p_j :                                                   /* Step 2 */
        wait until [received (p_j, yes, step2) from p_j or p_j ∈ ◇S_i] ;
            if p_j ∈ ◇S_i then
                send (p_i, startCons) to all participants ;
                outcome := uniform-consensus(abort) ;  /* Termination by consensus */
                return outcome ;
    send (p_i,preCommit, step3) to all ;
    for each participant p_j :                                                   /* Step 3 */
        wait until [received ((p_j,preCommit, step3) or (p_j, startCons)) from p_j or p_j ∈ ◇S_i] ;
            if received (p_j, startCons) from p_j or p_j ∈ ◇S_i then
                outcome := uniform-consensus(commit) ;  /* Termination by consensus */
                return(outcome) ;
    send (p_i, commit, decide) to all participants ;
    return commit ;  /* Commit decision */

|| /* Task 2 */
    wait until [received (p_j, outcome, decide) from p_j] ;
        send (p_i, outcome, decide) to all participants ;
        return outcome ;
coend
```

Figure 2: The DNB-AC protocol of process p_i

3.3 Correctness

The complete proof of the DNB-AC protocol is given in the Appendix. We give here some informal correctness arguments, that should help the reader in understanding the protocol.

Proposition 1 (AC-Validity). *A participant can decide commit only if every participant sends a yes vote to all participants (at line 10).*

PROOF (SKETCH). *Commit* can be decided either at line 24, or at line 21 if some participant starts the uniform consensus with *commit* as the initial value. In both cases, at least some participant has started step 3, and has thus received a *yes* vote from all (line 12). □

Proposition 2 (AC-Uniform-Agreement). *No two participants can decide differently.*

PROOF (SKETCH). If some participant does not send a *yes* vote to all, then by Proposition 1, no participant decides *commit*. Assume that every participant sends a *yes* vote to all (line 10), and proceeds to step 2. There are two cases to consider: *(a)* no participant suspects another participant in step 2 (line 12), or *(b)* some participant suspects another participant in step 2.
Case(a). No uniform consensus is started with *abort* as the initial value. By the C-Validity condition (Sect. 2.1), no participant can decide *abort*.
Case(b). Every participant which decides, decides through the uniform consensus protocol. By the C-Uniform-Agreement condition (Sect. 2.1), no two participants decide differently. □

Proposition 3 (AC-Termination). *Assume that a majority of participants are correct, and consider the properties of the $\diamond\mathcal{S}$ failure detector. Then every correct participant eventually decides.*

PROOF (SKETCH). There are three cases to consider: *(a)* at least one correct participant does not start step 2, or *(b)* all correct participants start step 2, and at least one correct participant does not start step 3, or *(c)* all correct participants start step 3. In all three cases, the *strong completeness* property of $\diamond\mathcal{S}$ ensures that a crashed process is eventually suspected by every correct process.
Case (a). If a correct participant p_i does not start *step 2*, then either it has voted *no* (line 7), or it has suspected p_1 (line 4). In both situations, p_i sends the decision message $(p_i, abort, decide)$ to all (at line 5), and every correct participant eventually decides.
Case (b). If one correct participant p_i does not start *step 3*, then p_i must have suspected another participant. In this case, p_i sends a *startCons* message to all (line 14), and starts uniform consensus (line 15). Thus p_i does not send *preCommit* (line 17), and every correct participant that starts step 3, also starts uniform consensus (line 21). As a majority of participants start uniform consensus, then by the C-Termination condition (Sect. 2.1), every correct participant eventually decides.
Case (c). If one correct participant p_i receives *preCommit* estimates from all, then p_i sends the decision message $(p_i, commit, decide)$ to all (line 23), and every correct participant eventually decides. Otherwise (no correct participant receives *preCommit* from all), p_i receives either *startCons*, or suspects another participant: in both situations, p_i starts uniform consensus (line 21). As a majority of correct participants starts uniform consensus, then by the C-Termination condition, every correct participant eventually decides. □

Proposition 4 (AC-Non-Triviality) *If no participant votes no, and no participant is ever suspected, then the decision is commit.*

Protocol	Number of steps	Number of messages without a broadcast network	Number of messages with a broadcast network
2PC	3	3 (n−1)	n+1
DNB−AC	3	(n−1)(2n+1)	2n+1
3PC	5	5 (n−1)	2n+1

Figure 3: Number of steps (see Fig. 1) and messages in the "good" case

PROOF (SKETCH). We prove the result by showing that *abort* can be decided only if some participant is suspected, or some vote is *no*. At lines 6, 9, and 16, a participant decides *abort* only if it has suspected another participant, or if it has voted *no*. At line 22, a participant decides *abort* only if *abort* has been returned by the uniform consensus. In this case, by the C-Validity condition (Sect 2.1), some participant p_i has started uniform consensus at line 15 with *abort* as its initial value, which means that p_i has suspected some participant. □

4 Evaluation of the DNB-AC protocol

In this section we compare the DNB-AC protocol with 2PC and 3PC in the "good" case, i.e. when there is no suspicion and no participant votes *no* [5].

As we have already pointed out, the DNB-AC and the 2PC [4] protocols require a minimum of three steps to reach the *commit* decision, whereas the 3PC [7] protocol requires a minimum of five steps (Fig 3). The minimum number of messages needed to decide *commit* are given in Figure 3: we distinguish two cases, (a) without a *broadcast* network, and (b) with a *broadcast* network.

(a) Without a broadcast network. In the 2PC, the coordinator sends $n-1$ messages to the participants, then $n-1$ messages are sent by the participants to the coordinator (step 2), and finally the coordinator sends $n-1$ messages (step 3). This makes a total of $3(n-1)$ messages. The 3PC requires $n-1$ additional messages in step 4, and $n-1$ additional messages in step 5, hence a total of $5(n-1)$ messages. In the DNB-AC protocol, $n-1$ messages are initially sent by the coordinator. Then, both in step 2 and in step 3, every participant sends $n-1$ messages to all. This makes a total of $(n-1)+n(n-1)+n(n-1) = (n-1)(2n+1)$ messages.

(b) With a broadcast network. In the 2PC, the coordinator broadcasts 1 message to all, then $n-1$ messages are sent by the participants, and finally the coordinator broadcasts 1 message to all: $n+1$ messages are thus sent. The 3PC generates $n-1$ additional messages in step 4 (participant to coordinator), and 1 additional broadcast in step 5 (coordinator to participants): a total of $2n+1$ messages. In the DNB-AC protocol, the coordinator first broadcasts 1 message, then every participant broadcasts 1 message to all (step 2), and finally every participant broadcasts 1 message to all (step 3): this makes a total of $2n+1$ messages[6].

Figure 4 (a and b) compares the performance on the 2PC, 3PC and DNB-AC protocols. Theses figures have been obtained on Ethernet (10Mb/s), with a set of SPARC20 machines (running SunOS 5.3), using the UDP transport protocol. Different participants are located on different machines, and a transaction simply consists in updating an integer at each par-

[5] This corresponds to the most frequent case.

[6] This is exactly the number of messages generated by the 3PC!

(a) without a broadcast network

Protocol	2 participants	4 participants	6 participants	9 participants
2PC	37	45	54	68
DNB−AC	38	48	61	80
3PC	50	60	74	93

(b) with a broadcast network

Protocol	2 participants	4 participants	6 participants	9 participants
2PC	38	42	51	62
DNB−AC	38	43	54	68
3PC	50	58	70	85

Figure 4: Performances (average time, in msec, to commit a transaction)

ticipant. Before returning its vote, a participant performs a disk access (to store the new updated value). The performances show that, the DNB-AC protocol is significantly faster than the 3PC protocol, and in the case of a broadcast network (especially if the number of participants is low), is almost as efficient as the 2PC protocol.

5 Concluding remarks

In this paper we have discussed an atomic commitment protocol, named DNB-AC (Decentralized Non-Blocking Atomic Commitment) which provides the non-blocking property while requiring the same number of steps to commit than the 2PC protocol (three steps). We have shown that the DNB-AC protocol provides significant better performances than the classical non-blocking 3PC protocol [7], and using a broadcast network, is almost as efficient as the 2PC protocol [4].

Failure scenarios are handled using a uniform consensus protocol as a termination protocol [3]. This modular approach shows the relationship between atomic commitment and uniform consensus problems, and enables to have a rigorous characterization of the liveness of DNB-AC in an asynchronous system augmented with an unreliable failure detector.

References

[1] O. Babaoglu and S. Toueg. Non-Blocking Atomic Commitment. In Sape Mullender, editor, *Distributed Systems*. ACM Press, 1993.

[2] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley. 1987.

[3] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Technical Report 93-1374, Department of Computer Science, Cornell Univ, 1993. A preliminary version appeared in the *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 325-340. ACM Press, August 1991.

[4] J. Gray. Notes on Database Operating Systems. In *Operating Systems: An Advanced Course*. Lecture Notes in Computer Science (60), Springer Verlag. 1978.

[5] V. Hadzilacos. On the relationship between the atomic commitment and consensus problems. In *Fault-Tolerant Distributed Computing*. B. Simons and A. Spector, editors, Lecture Notes in Computer Science (448), Springer Verlag. 1990.

[6] I. Keidar and D. Dolev. Increasing the Resilience of Atomic Commit at No Additional Cost. Technical Report CS94-18. Institute of Computer Science, The Hebrew University of Jerusalem, October 1994.

[7] D. Skeen. NonBlocking Commit Protocols. In Proceedings of the *ACM SIGMOD International Conference on Management of Data*, pages 133-142. ACM Press, 1981.

[8] D. Skeen. A Quorum-Based Commit Protocol. In Proceedings of the *Berkeley Workshop on Distributed Data Management and Computer Networks*, 6, pages 69-80, February 1982.

Appendix: the proofs

Proposition 1 (AC-Validity). *A participant can decide commit only if every participant sends a yes vote to all participants (at line 10).*

Proof. A participant can decide *commit* either in Task 1 or in Task 2. A participant decides *commit* in Task 2 only if, in Task 1, some participant has decided *commit*. In Task 1, a participant p_i can decide *commit* at lines 16, 22, or 24. For p_i to reach line 22 or line 24, p_i must have received *yes* votes from all participants. To decide *commit* at line 16, p_i must have got *commit* as the result of the uniform consensus of line 15. By the C-Validity condition of uniform consensus (Sect. 2.1), some participant p_j must have started the uniform consensus with *commit* as its initial value, i.e at line 22. For p_j to reach line 21, p_j must have received *yes* votes from all participants. □

Proposition 2 (AC-Uniform-Agreement). *No two participants can decide differently.*

Proof. A participant decides *abort* in Task 2 only if some participant has decided *abort* in Task 1. Similarly, a participant decides *commit* in Task 2 only if some participant has decided *commit* in Task 1. We show that no two participants can decide differently in Task 1. By Proposition 1, if a participant does not send a *yes* vote to all (line 10), then no participant can decide *commit*, and no two participants can decide differently. Assume thus that all participants send a *yes* vote to all (line 10), and start *step 2*. In this case, a participant can only decide either at line 16, line 22, or line 24. There are two cases to consider: *(a)* at least one participant suspects another participant in *step 2* (at line 12), or *(b)* no participant suspects another participant in *step 2*.

Case (a): A participant p_i suspects another participant in step 2. In this case, p_i will start uniform consensus at line 15 (with *abort* as its initial value), and will not send its *preCommit* estimate to all (line 17). As a consequence, no participant decides at line 24, and any participant which decides, decides through uniform consensus (started either at line 15 or line 21). By the *C-Uniform-Agreement* condition of uniform consensus (Sect 2.1), no two participants can decide differently.

Case (b): No participant suspects another participant in step 2. In this case, no uniform consensus protocol is started with *abort* as initial value. By the C-Validity condition of uniform consensus (Sect 2.1), no participant will decide *abort* at line 22. Hence, every participant that decides (at lines 22 or 24), decides *commit*. □

Proposition 3 (AC-Termination). *Assume that a majority of participants are correct, and consider the property of the $\diamond\mathcal{S}$ failure detector. Then every correct participant eventually decides.*

Proof. There are three cases to consider: *(a)* at least one correct participant does not start step 2, or *(b)* all correct participants start step 2, and at least one correct participant does not start step 3, or *(c)* all correct participants start step 3. We show that in the three cases, every correct participant eventually decides.

Case (a): At least one correct participant does not start step 2. If the coordinator fails, then by the *strong completeness* property of $\diamond\mathcal{S}$, every correct participant eventually suspects it. If the coordinator is correct, then it will send the *(voteRequest,step1)* message to all (line 2). By the reliable channel assumption, every correct participant waiting at line 3 eventually receives this message. As a consequence, if one correct participant p_i does not start *step 2*, then p_i

must have either suspected the coordinator, or must have voted *no* (line 7). If p_i suspects the coordinator, p_i sends the decision message $(p_i, abort, decide)$ to all (line 5), and decides *abort* (line 6). If p_i votes *no*, then p_i sends the same decision message $(p_i, abort, decide)$ to all (line 8), and decides *abort* (line 9). By the reliable channel assumption, in both situations, every correct participant that has not already decided, eventually receives the decision message $(p_i, abort, decide)$, and decides *abort* (in Task 2).

Case (b): All correct participants start step 2, and at least one correct participant does not start step 3. The only way for a participant p_i to not start step 3 is by suspecting another participant in step 2 (at line 12). This holds for the following reason. Assume that no correct process suspects another participant: then by the *strong completeness* property of $\diamond\mathcal{S}$, all participants are correct. In this case, and as we assume that all correct participants start step 2 (and thus have sent their *yes* vote), p_i will receive their *yes* vote, and proceeds to step 3: a contradiction.

Assume thus that the correct participant p_i suspects another participant in step 2 (at line 12), sends a *startCons* message to all (line 14), and starts uniform consensus (line 15). At least one *preCommit* message will be missing in *step 3* (p_i's), and by the reliable channel assumption, every correct participant that starts step 3, eventually either suspects another participant, or receives p_i's *startCons* message. In both situations, every correct participant that starts step 3, starts uniform consensus. Therefore, all correct participants eventually start uniform consensus (either in step 2 or in step 3). As we assume a majority of participants to be correct, a majority of correct participants eventually start uniform consensus. By the C-Termination condition of uniform consensus (Sect 2.1), every correct participant eventually decides.

Case (c): All correct participants start step 3. If some correct participant p_i receives the *preCommit* message from all participants, p_i sends the decision message $(p_i, commit, decide)$ to all (line 23), and decides *commit* (at line 24). Hence, every correct participant that has not already decided, eventually decides *commit*. If no correct participant receives the *preCommit* message from all, then every correct participant eventually suspects another participant. This holds for the following reason. If some correct participant does not suspect any another participant, then by the *strong completeness* property of $\diamond\mathcal{S}$, all participants are correct. In this situation, and as we assume that all correct participants start step 3 (and thus have sent *preCommit*), some correct participant eventually receives *preCommit* from all: a contradiction. Thus every correct participant eventually suspects another participant in step 3 (at line 19). Hence every correct participant eventually starts uniform consensus. As we assume a majority of participants to be correct, a majority of correct participants eventually start uniform consensus. By the C-Termination condition of uniform consensus (Sect 2.1), every correct participant eventually decides. □

Proposition 4 (AC-Non-Triviality) *If no participant votes no, and no participant is ever suspected, then the decision is commit.*

Proof.We prove the result by showing that *abort* can be decided only if some participant votes *no* or suspects another participant. If a participant decides *abort* in Task 2, then some participant has decided *abort* in Task 1. A participant can decide *abort* in Task 1 either at lines 6, 9, 16, or 22. At lines 6, 9, and 16, a participant decides *abort* if it has suspected another participant, or if it has voted *no*. At line 22, a participant decides *abort* only if *abort* has been returned by the uniform consensus protocol. In this case, by the *C-Validity* condition of uniform consensus (Sect2.1), some participant p_j must have started uniform consensus at line 15 with *abort* as its initial value. In order to do so, p_j must have suspected some participant. $\Box$