Technical Report

UCAM-CL-TR-668
ISSN 1476-2986

Number 668

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Optimistically Terminating Consensus

Piotr Zieliński

June 2006

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

Optimistically Terminating Consensus

Piotr Zieliński
University of Cambridge
Cavendish Laboratory
J J Thomson Avenue, Cambridge CB3 0HE, UK
piotr.zielinski@cl.cam.ac.uk

Abstract

Optimistically Terminating Consensus (OTC) is a variant of Consensus that decides if all correct processes propose the same value. It is surprisingly easy to implement: processes broadcast their proposals and decide if sufficiently many processes report the same proposal. This paper shows an OTC-based framework which can reconstruct all major asynchronous Consensus algorithms, even in Byzantine settings, with no overhead in latency or the required number of processes. This result does not only deepen our understanding of Consensus, but also reduces the problem of designing new, modular distributed agreement protocols to choosing the parameters of OTC.

Keywords: Consensus, fault tolerance, Byzantine faults.

1 Introduction

In the Consensus problem, a fixed group of processes communicating through an asynchronous network cooperate to reach a common decision. Each of the processes proposes a value, say a number, and then they all try to agree on one of the proposals. Despite the apparent simplicity, Consensus is surprisingly difficult to solve in a fault-tolerant manner, so that even if some processes fail, the others will always reach an agreement. Consensus is also universal: it can be used to implement *any* sequential object in a distributed and fault-tolerant way [15].

In the most popular approach to solve Consensus, a distinguished process, called the *leader* or *coordinator*, tries to impose its proposal on the others. This succeeds if sufficiently many processes accept the coordinator's proposal. Otherwise, another process becomes the coordinator and repeats the protocol. Coordinators keep changing until one of them succeeds and makes all correct processes decide.

The method just described forms the basis of a great majority of asynchronous Consensus protocols [4, 5, 7, 9, 16, 19, 21, 25, 29]. It is interesting that, despite this deep similarity in structure, all these protocols were constructed independently, from scratch. I argue that this non-modular approach results in wasted effort, especially with protocols tolerating malicious participants. Indeed, a small change to such protocols usually requires rewriting the already multiple-page long and repetitive proof.

Numbers	Correctness	Honesty	Behaviour
n	correct	honest	according to the specification
f	faulty	honest	according to the specification until it stops
m	faulty	malicious	completely arbitrary

Figure 1: Categories of processes.

This paper shows a possible solution to this problem. It presents an *agreement framework* that allows one to reconstruct all above Consensus protocols and construct new ones just by changing a small simple part of the algorithm. This approach greatly simplifies the design of new Consensus algorithms, especially in the most complicated Byzantine model. I believe that it also contributes to a better understanding of the Consensus problem itself.

This research focuses solely on the number of communication steps (latency) required in favourable scenarios, when the first coordinator is correct. Therefore, by "reconstructing" I mean matching the latency and the number of required processes. Other factors, such as memory requirements, message complexity, or channel properties, are beyond the scope of this work.

One of the motivations for this research was an enormous success of the failure detection framework [6], which encapsulates details of timing assumptions into *failure detectors*. What this paper proposes is encapsulating the details of individual rounds of Consensus algorithms, such as the latency, message patterns, and the number and type of faults tolerated, into a new abstraction called *Optimistically Terminating Consensus* (OTC).

The OTC approach is attractive for several reasons:

- The OTC framework can reconstruct all major asynchronous Consensus protocols [4, 5, 6, 9, 16, 18, 19, 21, 24, 29], significantly more than any other agreement framework [1, 2, 3, 13, 17, 28].

- Unlike other frameworks, OTC tolerates malicious processes. Consensus protocols for such settings are the most difficult to design, and benefit from modularization most.

- OTC instances are completely independent in their implementation and specification, so it is possible to use several different implementations of OTC in the same Consensus algorithm. For example, one might use a fast implementation for the first round, and a more fault-tolerant one for the others.

- The OTC abstraction is implementable in purely asynchronous settings. All other factors, such as choosing the proposals and the time for stopping a round, are clearly separated from the OTC specification.

Solving Consensus is only one of possible uses of OTC, which can also be used to obtain latency-optimal implementations of other agreement abstraction, notably Atomic Commitment for distributed databases [31]. In all cases, simplicity of OTC allows one to discover and verify new protocols automatically, which is especially handy for non-standard failure models [31].

This paper is structured as follows. Section 2.2 specifies the system model and the Consensus problem. Section 3 introduces the OTC abstraction and shows how it can be used to solve Consensus. Sections 4, 5, and 7 present different OTC implementations that are used to reconstruct a number of Consensus algorithms in Section 5. Section 8 shows the optimality of those implementations, and Section 9 concludes the paper.

1.1 Related work

Agreement frameworks have been investigated before. Mostéfaoui and Raynal [26] proposed a generic Consensus algorithm that could use one of the two failure detectors $\Diamond S$ and S [6]. Hurfin et al. [17] generalized this method by allowing the message exchange pattern to be chosen for each round of the protocol. In other words, the designer could specify their approach to the time vs. message complexity problem: whether they preferred a low latency or a small number of messages.Mostéfaoui et al. [28] extended the choice of options here to include the leader elector Ω and randomization, however, the protocols they presented have higher latency than ad-hoc solutions.

Boichat et al. [1] presented a modular deconstruction of Paxos into an *eventual leader elector*, similar to Ω, and a *ranked register*. By modifying the implementation of these two modules, they obtained Fast Paxos [1], Disk Paxos [12], and two variants of Paxos for the crash-recovery model. Later, they replaced the ranked register with the *eventual register* [2, 3].

Guerraoui and Raynal [13] unified the approaches outlined in the last two paragraphs. They presented a Consensus algorithm that uses a new *Lambda* abstraction, which can be implemented with different failure detectors in a modular way. Most known Consensus protocols for *crash-stop* asynchronous systems with failure detectors can be implemented in this framework without any latency overhead.

Recently, Guerraoui and Raynal [14] presented *Alpha*: an abstraction similar to Lambda but with a slightly different goal. Alpha provides an agreement framework that allows one to construct a Consensus algorithm for different communication models such as message passing, shared memory, and independent disks.

Lampson [22] presented Abstract Paxos, which can be used to obtain Byzantine Paxos [5], Classic Paxos [19], and Disk Paxos [12]. Recently, Li et al. [23] showed how to deconstruct Classic Paxos and Byzantine Paxos using two new abstractions: a *register* that encapsulates quorum operations and a *token* that encapsulates a proof that the leader has read a particular value from the register.

2 Definitions

2.1 System model

This paper assumes a system consisting of a fixed number of processes. Out of the total number n of processes, at most f can fail, out of which at most m in a malicious way, where $m \leq f \leq n$ are parameters of the model (Figure 1). Processes communicate through asynchronous reliable channels: each message sent from one *correct* process to another correct process will eventually be received (reliability) but the transmission time is unbounded (asynchrony). To make Consensus solvable [11], I additionally assume $\Diamond S$,

Name	Type	Meaning	Equivalent in Hurfin's algorithm
propose(v)	action	propose v	broadcast $phase_2(r, v)$ if no *stop* or *propose* before
stop	action	stop processing	broadcast $phase_2(r, \bot)$ if no *stop* or *propose* before
decision(v)	pred.	if true, then v is the decision	at least $n - f$ messages $phase_2(r, v)$ received
valid(v)	pred.	if true, then an honest process proposed v	at least one message $phase_2(r, v)$ received
possible(v)	pred.	if any process ever decides on v, then true	less than $n - f$ messages $phase_2(r, \text{non-}v)$ received

Figure 2: Summary of the primitives provided by OTC.

the weakest failure detector with this property [7]. Detector $\Diamond S$ provides each process with a constantly updated list of processes suspected to be faulty, and ensures:

Strong Completeness. Every faulty process will eventually be suspected by every correct process.

Eventual Weak Accuracy. There is a process that will eventually never be suspected by any correct process.

In the Byzantine model ($m > 0$), failure detection cannot be completely separated from the main algorithm [8]. Thus, in this model, I assume *eventual synchrony* instead [10]: there is *unknown* upper bound on message transmission times between correct processes. For example, message transmission times cannot grow ad infinitum.

2.2 Consensus

In Consensus, processes propose values and are expected to eventually agree on one of them. The following holds:

Validity. The decision was proposed by some process.

Agreement. No two processes decide differently.

Termination. All correct processes eventually decide.

In the Byzantine model, these requirements apply only to honest processes. Since malicious processes can undetectably lie about their proposals, I also assume that Validity must be satisfied only if all processes are honest in a particular run.

Let us start with the crash-stop model. The Consensus algorithm proposed by Hurfin and Raynal [16], shown in Figure 3, assumes a majority of correct processes ($n > 2f$) and progresses in a sequence of rounds $r = 1$, 2, etc. Each process p_i keeps an *estimate* est_i of the decision, initially equal to its own proposal v_i. In each round r, a deterministically chosen coordinator p_c broadcasts $phase_1(r, est_i)$ containing its estimate est_c, in an attempt to make est_c the decision (line 5).

In lines 6–8, each process p_i waits until it receives the estimate est_c or its failure detector $\Diamond S_i$ suspects the coordinator p_c. It broadcasts $phase_2(r, aux_i)$ with aux_i being, depending on which of the two conditions holds, the coordinator's estimate etc_c or a special symbol $\bot$, respectively. Note that reliability of the channels and Strong Completeness of $\Diamond S$ ensure that at least one of these two conditions will eventually hold.

```
1  when process p_i executes propose(v_i) do
2    est_i ← v_i
3    for r = 1, 2, ... do
4      c ← ((r − 1) mod n) + 1                          { p_c is the coordinator }
5      if i = c then broadcast phase_1(r, est_i)
6      wait for one of the following conditions:
7        if phase_1(r, v) received from p_c then aux_i ← v
8        if p_c is suspected by ◇S_i then aux_i ← ⊥
9      broadcast phase_2(r, aux_i)
10     wait for n − f messages phase_2(r, aux_j)
11       if all received aux_j ≠ ⊥ then bcast “decide on aux_j”
12       if at least one aux_j ≠ ⊥ then est_i ← aux_j
13 when received “decide on v” do
14   broadcast “decide on v”
15   decide on v
16   halt
```

Figure 3: Consensus algorithm from [16].

After broadcasting in line 9, p_i waits for $n-f$ messages $phase_2(r, aux_j)$, where $aux_j \in \{est_c, \bot\}$. If all $n-f$ received $aux_j = est_c \neq \bot$, process p_i broadcasts “decide on aux_j” to all processes, including itself. Upon reception of such a message, the recipient rebroadcasts it, decides on the specified value, and stops processing (lines 13–16).

Line 12 is arguably the most crucial part of the algorithm. It ensures that if some process p_i broadcast “decide on aux_j” in line 11, all processes will update their estimates to aux_j. Indeed, the assumption $f < \frac{1}{2}n$ implies that $n - f > \frac{1}{2}n$; since p_i received messages $phase_2(r, aux_j)$ from a majority of processes, no process can receive messages $phase_2(r, \bot)$ from a majority. After line 12, all processes p_i have $est_i = aux_j$, so no decision different than aux_j can be reached in future rounds.

3 Optimistically Terminating Consensus

Optimistically Terminating Consensus (OTC) can be thought of as a version of Consensus that guarantees Termination only if all correct processes propose the same value. The full list of predicates and actions provided by this abstraction is shown in Figure 2. Before giving a rigorous specification of OTC, I will introduce it informally by presenting an OTC-based Consensus algorithm. This algorithm can implement different Consensus protocols, depending on what OTC implementation is used.

The OTC-based Consensus algorithm, shown in Figure 4, is a revised version of Hurfin’s algorithm from Figure 3. As in the original, each process keeps an estimate est_i and proceeds through a sequence of rounds. Each round $r = 1, 2, \ldots$ starts with the coordinator p_c broadcasting its current estimate est_c. The rest of the round consists of two parts: optimistic (lines 7–8) and pessimistic (lines 9–13).

```
1  when process p_i executes propose(v_i) do
2    est_i ← v_i
3    for r = 1, 2, ... do
4      c ← ((r − 1) mod n) + 1
5      if i = c then broadcast phase_1(r, est_i)
6      execute                                   { interrupt lines 7–8 when until holds }
7        wait for phase_1(r, v) received from p_c
8        OTC_r.propose(v)
9      until p_c suspected by ◇S_i or "stop round r" received
10     OTC_r.stop and broadcast "stop round r"
11     wait until OTC_r.possible(v) holds for at most one v
12     wait until OTC_r.possible(v) ⇒ OTC_r.valid(v) for all v
13     if OTC_r.possible(v) for some v then est_i ← v

14 when OTC_r.decision(v) or "decide on v" received do
15   broadcast "decide on v"
16   decide(v)
17   halt
```

Figure 4: Crash-stop Consensus using OTC.

3.1 Optimistic part, lines 7–8

Each round $r = 1, 2, \ldots$ uses its own, independent instance OTC_r. OTC provides each process with an action *propose*(v) to propose, and a predicate *decision*(v) to decide. (Predicates are functions that operate on the process' state and return immediately without affecting the state.) Predicate *decision*(v), defined for all possible values v, is initially false and becomes true when the process is sure that v is the decision. The semantic of *propose*(v) and *decision*(v) is the same as in Consensus, except that the eventual decision is guaranteed only if all correct processes proposed the same value. This paper assumes $v \neq \perp$.

In lines 7–8, each process waits for the coordinator's estimate est_c, and proposes it to OTC_r. If the coordinator is correct and not suspected, all correct processes will execute $OTC_r.propose(est_c)$. As a result, $OTC_r.decision(est_c)$ will eventually become true at all correct processes, which will decide on est_c and halt (lines 14–17).

As shown in Figure 2, in Hurfin's algorithm, $OTC_r.propose(v)$ corresponds to broadcasting $phase_2(r, v)$. The predicate $OTC_r.decision(v)$ holds iff the process received at least $n - f$ messages $phase_2(r, v)$.

3.2 Pessimistic track, lines 9–13

In addition to *propose* and *decision*, OTC provides three other primitives: action *stop* and predicates *possible*(v) and *valid*(v). Their purpose is to let processes keep their estimates correct, and progress to the next round if no decision has been made.

While executing the optimistic part (lines 7–8), each process p_i monitors its failure detector $\Diamond S$. When p_i suspects the coordinator, it interrupts the optimistic part and executes $OTC_r.stop$. It also informs other processes, who follow suit (lines 9–10). In Hurfin's algorithm, $OTC_r.stop$ corresponds to broadcasting $phase_2(r, \perp)$. Here, however, p_i can execute both $OTC_r.propose$ and $OTC_r.stop$, which could lead to broadcasting both $phase_2(r, est_c)$ and $phase_2(r, \perp)$. To avoid this, OTC_r allows p_i to broadcast $phase_2$ only once, and ignores the second attempt (Figure 2).

Predicate $OTC_r.valid(v)$ holds at p_i if p_i is sure that at least one honest process executed $OTC_r.propose(v)$. Predicate $OTC_r.possible(v)$ holds if p_i cannot exclude the possibility that OTC_r has decided or will decide on v at some honest process. Therefore, in lines 11–12, p_i waits until it can be sure that no decision has been made in round r, except possibly one value v, and that this v has indeed been proposed to OTC_r by some honest process. If such a v exists, p_i adopts it as its new estimate. Line 11 ensures that if $OTC_r.decision(v)$ holds at some process, then all processes will update their estimates to v before starting the next round. This implies that no decisions other than v will be reached in future rounds (Agreement). Line 12 ensures that only proposed values can become estimates (Validity).

Predicate $OTC_r.valid(v)$ is initially false for all v and becomes true when p_i has sufficient evidence that v has been proposed by an honest process. In Hurfin's algorithm, $OTC_r.valid(v)$ holds iff p_i received at least one message $phase_2(r, v)$.

Predicate $OTC_r.possible(v)$ is initially true for all v and becomes false when p_i has sufficient evidence that $OTC_r.decision(v)$ has never held and will never hold at any honest process. In Hurfin's algorithm, $OTC_r.possible(v)$ holds if p_i received fewer than $n - f$ messages $phase_2(r, \text{non-}v)$, that is, either $phase_2(r, \perp)$ or $phase_2(r, v')$ with $v' \neq v$.

The implementation of OTC must ensure that lines 11–12 do not block the algorithm. In Hurfin's algorithm, all correct processes eventually receive $n - f$ messages $phase_2(r, aux_j)$ with $aux_j \in \{est_c, \perp\}$. This implies that $OTC_r.possible(v)$ can hold only for $v = est_c$ because for any $v \neq est_c$, "non-v" includes both est_c and $\perp$. On the other hand, $OTC_r.possible(est_c)$ means that less than $n - f$ messages $phase_2(r, \perp)$ have been received, which implies $OTC_r.valid(est_c)$.

It is not difficult to see that the algorithm in Figure 4 with OTC implemented as in Figure 2 implements Consensus. This is true for *any* OTC implementation, not only for crash-stop settings but also, with minor modifications, for Byzantine settings. However, to prove this claim, we need a rigorous specification of OTC first.

3.3 OTC properties

As summarized in Figure 2, OTC equips each process with two actions: $propose(v)$ and *stop*, as well as three predicates: $valid(v)$, $possible(v)$, and $decision(v)$. These primitives satisfy the following properties:

Integrity. If $valid(v)$ holds at an honest process, then an honest process proposed v.

Possibility. If $decision(v)$ holds at an honest process, then $possible(v)$ holds at all honest processes, at all times.

Permanent Validity. Statement $possible(v) \Rightarrow valid(v)$ holds at any *complete* process (see below).

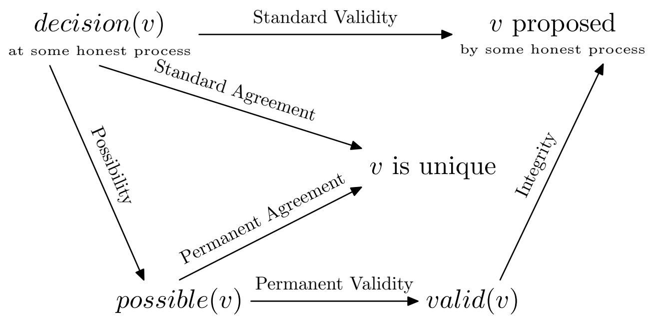


Figure 5: OTC properties graphically.

Permanent Agreement. Predicate $possible(v)$ holds for at most one v at any complete process.

Optimistic Termination (q, k). If at most q out of n processes are faulty, all correct processes propose v, and none of them executes *stop*, then $decision(v)$ will hold at all correct processes in k communication steps.

Properties Integrity and Possibility just formalize the definitions of predicates $valid(v)$ and $possible(v)$ from Figure 2.

In Permanent Validity and Permanent Agreement, an *honest* process p is *complete* if all correct processes executed *stop*, and p received all messages sent by those processes before or during executing *stop*. These properties ensure that, if all correct processes execute OTC_r*.stop* in line 10 (Figure 4), then all of them will eventually be complete, so lines 11 and 12 will not block. Note that p does not know which processes are correct, so it does not know whether it is complete or not.

Although not immediately obvious, Permanent Validity and Permanent Agreement are stronger versions of standard Validity and Agreement required by Consensus (Figure 5). For example, consider an OTC run that violates Standard Agreement and decides on two different values v and v'. By Possibility, $possible(v)$ and $possible(v')$ hold at all processes at all times. An extension of this run in which all correct processes execute *stop* and become complete violates Permanent Agreement. Similarly, Permanent Validity implies Standard Validity (Theorems A.1 and A.2).

3.4 Optimistic Termination

Optimistic Termination (q, k) of OTC_r ensures that a round r with a correct and non-suspected coordinator p_c will decide on est_c in $1 + k$ steps. In the first step, p_c broadcasts est_c, which is proposed to OTC_r by all correct processes (line 8). Optimistic Termination guarantees that all correct processes decide k steps later, provided that at most q processes are faulty. (No process ever executes OTC_r*.stop* because they do not suspect p_c.) Note that the same OTC_r can satisfy more than one Optimistic Termination condition.

This paper focuses on *favourable* runs, in which the first round coordinator is correct and not suspected, and at most q processes are faulty. In such runs, the Consensus algorithm from Figure 4 decides in $k + 1$ steps. For example, the OTC from Figure 2 satisfies Optimistic Termination $(f, 1)$; the resulting Consensus algorithm decides in two steps regardless of the number of faulty processes (which never exceeds f).

```
1  when process p_i executes propose(v_i) do
2    est_i ← v_i                                          { decision estimates }
3    signed_i ← ∅                                         { signed estimates }
4    for r = 1, 2, ... do
5      start timer_r
6      c ← ((r − 1) mod n) + 1
7      if i = c then broadcast phase_1(r, est_i, signed_i)
8      execute                              { interrupt lines 9–12 when until holds }
9        wait for phase_1(r, est_c, signed_c) from p_c
10       if r = 1 or est_c = est_i or
11         >m messages in signed_c have est_j ≠ est_i then
12           OTC_r.propose(v)
13     until timer_r expired or
14       "stop round r" received > m times
15     OTC_r.stop and broadcast "stop round r"
16     wait until OTC_r.possible(v) holds for at most one v
17     wait until OTC_r.possible(v) ⇒ OTC_r.valid(v) for all v
18     if OTC_r.possible(v) for some v then est_i ← v
19     digitally sign est_i and broadcast signed(r, est_i)
20     wait for f + m + 1 signed messages signed(r, est_j)
21       and store them in signed_i
22     if >f messages in signed_i have the same est_j then
23       est_i ← est_j
24 when OTC_r.decision(v) for the current r
25   or "decide on v" received > m times do
26     decide(v) and broadcast "decide on v"
27     wait until received "decide on v" > f + m times
28     halt
```

Figure 6: Byzantine Consensus using OTC.

To guarantee Termination of the Consensus protocol in any run, I assume that all OTC_r satisfy Optimistic Termination (f, k) for some k. Then, Strong Completeness of $\Diamond S$ implies that in any round, all correct processes will either decide or pass line 9 in Figure 4. Eventual Weak Accuracy of $\Diamond S$ ensures that there will be a round with a correct and non-suspected coordinator. That round will decide.

3.5 Byzantine Consensus

Although OTC solves crash-stop Consensus, its real aim greatly facilitate (re)-construction Consensus algorithms for Byzantine settings (Figure 8). These protocols are notoriously difficult to design, and require long, elaborate proofs of correctness. The main purpose of the OTC framework is to stop designing such protocols being a nightmare.

To deal with malicious processes, the OTC Consensus algorithm from Figure 4 requires a few modifications (Figure 6). Although individual instances OTC_r are required to remain correct even in the presence of malicious processes, different instances OTC_r might reach different decisions. Therefore, we need to ensure that, if some round decided on v, no malicious coordinator p_c can broadcast $est_c \neq v$ in a later round.

To solve this problem, at the end of each round, processes digitally sign and broadcast their estimates est_i. Each process p_i waits for such estimates from $f + m + 1$ processes and stores them in the variable $signed_i$. Then, p_i checks whether more than f of these estimates are the same, and updates its own estimate if so. This ensures that, for any $v \neq est_i$, more than m signed estimates differ from v, so p_i can *prove* that no decision, other than possibly est_i, has been made.

At the beginning of each round, the coordinator broadcasts not only its estimate est_c but also the $f + m + 1$ estimates $signed_c$ it received in the previous round. If a process p_i receives $est_c \neq est_i$, it checks whether more than m estimates in $signed_c$ received from the coordinator are different from est_i. This proves that no decision has been made yet, and it is safe to accept the coordinator's estimate est_c.

Note that in favourable runs, in which all correct processes decide in the first round, no digital signatures are used.

Since traditional failure detectors are not implementable in Byzantine settings [8], the algorithm from Figure 6 assumes an unknown upper bound d on message transmission times [10]. Each process starts its $timer_r$ at the beginning of each round r, and executes $OTC_r.stop$ when $timer_r$ expires. The timeout periods grow from round to round, so that they will eventually become longer than kd, for any fixed k.

Note that line 13 can never block the algorithm forever. For this reason, we can allow a finite number rounds r to have OTC_r that does not satisfy Optimistic Termination (f, k).

Finally, minor modifications in lines 14, 25, 27 results from the fact that, in the Byzantine model, one can trust only groups containing more than m processes (Appendix F).

4 Implementing OTC in one step

Figure 7 presents an implementation of OTC that satisfies Optimistic Termination $(q, 1)$, that is, decides in one step in favourable runs. It uses a new simple broadcast abstraction called *monocast*, which is similar to ordinary broadcast, except that each process is allowed to broadcast only one message; the possible second and subsequent attempts are silently ignored. A similar restriction applies to *monoreceiving* messages: the second and subsequent messages received from the same process are ignored. This prevents malicious senders from broadcasting multiple messages.

To prevent identity spoofing in the Byzantine case, I assume pairwise shared keys for *symmetric* encryption and authentication [5]. The setup of such keys is beyond the scope of this paper. It has little impact on performance as it needs to be done only once.

As shown in Figure 7, processes execute *propose*(v) by monocasting v, and *stop* by monocasting $\bot$. If a process executes both, the monocast abstraction ensures that the first action wins and the other is ignored (compare with Figure 2).

For any process p_i, predicates *valid*(v), *decision*(v), and *possible*(v) are determined by monoreceived messages. Predicate *valid*(v) is true if p_i has monoreceived v more than m times. Since there are at most m malicious processes, at least one of the senders is honest,

Primitive	Implementation / Definition
propose(v)	monocast v
stop	monocast $\perp$
decision(v)	v monoreceived at least $n-q$ times
possible(v)	non-v monoreceived at most $q+m$ times
valid(v)	v monoreceived more than m times

Figure 7: One step OTC implementation.

so it must have proposed and monocast v (Integrity). Predicate *decision*(v) holds if p_i monoreceived v at least $n-q$ times. This means that if all $n-q$ correct processes propose v and do not execute *stop*, they will all decide in one communication step (Optimistic Termination). Finally, predicate *possible*(v) is true if p_i received values different from v at most $q+m$ times. If *decision*(v) holds, then at least $n-q-m$ honest processes must have monocast v. As a consequence, no process can monoreceive non-v values more than $q+m$ times, which makes predicate *possible*(v) true at all honest processes, at all times (Possibility).

As just shown, the OTC algorithm in Figure 7 satisfies Integrity, Possibility, and Optimistic Termination $(q,1)$ regardless of the values of parameters n, f, m, q. The other two properties require (see the proof below):

Permanent Validity	$n > f + 2m + q$
Permanent Agreement	$n > f + 2m + 2q$

For example, using one-step OTC with $q = m = f$ in the Byzantine Consensus from Figure 6 requires $n > 5f$. In favourable runs, this algorithm decides in two steps: one for the coordinator's estimate est_1 to reach other processes, and the other for OTC_1 to decide. Deciding in two steps given $n > 5f$ matches the properties of [24]. Other reconstructions, shown in Figure 8, will be discussed in Section 6.

Proof. Let p_i be a complete process. By the definition of completeness, p_i has monoreceived one message from each of the $n-f$ correct processes. If *possible*(v) holds, then at most $q+m$ of these messages are other than v. This means that p_i monoreceived at least $n-f-q-m > m$ messages v, which implies *valid*(v) (Permanent Validity).

For Permanent Agreement, assume that *possible*(v) and *possible*(v') hold some values v and v'. This means p_i monoreceived at most $2q+2m$ messages that are either non-v or non-v'. Since p_i has monoreceived at least $n-f > 2q+2m$ messages, at least one of them is neither non-v nor non-v'. This message is v and v' at the same time, which implies $v = v'$ (Permanent Agreement). □

Recall, that completeness of p_i requires all correct processes to execute *stop*. Note however that the above proof uses completeness only to ensure that p_i monoreceived one message from each correct process. Since *stop* and *propose*(v) both involve a monocast, Permanent Validity and Permanent Agreement hold even if we replace "*stop*" by "*stop* or *propose*" in the definition of completeness. This property is specific to one-step OTC and will be used in Section 6.1.

Algorithm	m	q	Steps	Type of OTC_1		Requirements
crash-stop [7, 16, 19, 29]	0	f	2	single-value	one-step	$n > f + 2m + q \ = 2f$
Cheap Paxos [21]	0	0	2	single-value	one-step	$n' > f + 2m + q \ = \ f$
Brasileiro et al. [4]	0	f	1*	multi-value	one-step	$n > f + 2m + 2q = 3f$
cheap one-step Consensus	0	0	1*	single-value	one-step	$n' > f + 2m + 2q = \ f$
Kursawe [18]	f	0	2	multi-value	one-step	$n > f + 2m + 2q = 3f$
Martin and Alvisi [24]	f	f	2	multi-value	one-step	$n > f + 2m + 2q = 5f$
one-step Byzantine (1)	f	0	1*	multi-value	one-step	$n > f + 2m + 2q = 3f$
one-step Byzantine (2)	f	f	1*	multi-value	one-step	$n > f + 2m + 2q = 5f$
Castro and Liskov [5]	f	f	3	multi-value	two-step	$n > f + \ m + \ q = 3f$
Zieliński [30]	f	$q/f/f$	2/3	multi-value	multi-step†	$n > f + m + f + q$
Dutta et al. [9]	m	$q/f/f$	2/3	multi-value	multi-step†	$n > f + m + f + \min\{m, q\}$
multi-step Byzantine	m	$q/q'/f$	2/3/4	multi-value	multi-step†	$n > f + m + q' + \min\{m, q\}$

† All multi-step algorithms additionally require $n > f + 2m + 2q$ and $n > 2f + m$.

Figure 8: Reconstructing various Consensus protocols in the OTC framework.

4.1 Single-value OTC

In the crash-stop Consensus algorithm in Figure 4, processes use the coordinator's estimate est_c as their proposal to OTC_r (line 8). Since the crash-stop model does not allow malicious coordinators, all values proposed to OTC_r are the same (est_c). Some OTC implementations for this model can explicitly assume that this is the case; I call them *single-value* OTCs, as opposed to the normal, *multi-value* OTCs, which tolerate different proposals.

Single-value OTCs differ from multi-value OTCs in that they do not have to explicitly satisfy Permanent Agreement, which in that case follows automatically from Permanent Validity. Indeed, assume that $possible(v) \Rightarrow valid(v)$ for all v. If $possible(v)$ holds for two different v, then $valid(v)$ holds for two different v, which implies that two different values have been proposed by honest processes. This contradicts the single-value assumption that all honest processes propose the same value. Standard Agreement follows from Standard Validity in a similar way.

Automatic satisfaction of Permanent Agreement means that single-value OTCs might require fewer processes than multi-value OTCs. For example, one-step OTC from Figure 7 requires $n > f+2m+2q$ to implement multi-value OTC, but only $n > f+2m+q$ for single-value OTC. Setting $m = 0$, $q = f$ leads to a two-step crash-stop Consensus algorithm that requires $n > 2f$, the same as in Hurfin's and similar algorithms [7, 16, 19, 29] (Figure 8).

5 Implementing OTC in two steps

In the Byzantine model, the requirements on n can be further reduced by using multiple-step OTC implementations implemented as *chains* of one-step OTC instances:

$$A_1 \quad \rightarrow \quad A_2 \quad \rightarrow \quad \cdots \quad \rightarrow \quad A_k.$$

Processes propose their value to the first instance A_1. Then, decisions are propagated along the chain: if a process reaches a decision v in instance A_i, it immediately proposes v to the next instance A_{i+1}. The decision of the last instance A_k becomes the final decision. Stopping the algorithm involves stopping all instances $A_1, \ldots, A_k$. Each A_i uses a separate instance of monocast.

The predicate *valid* is taken from the first instance, whereas predicates *possible* and *decision* come from the last instance:

$$\begin{aligned} valid(v) &\stackrel{\text{def}}{=} valid_{A_1}(v), \\ possible(v) &\stackrel{\text{def}}{=} possible_{A_k}(v), \\ decision(v) &\stackrel{\text{def}}{=} decision_{A_k}(v). \end{aligned}$$

Properties Integrity and Possibility of $A_1 \to \cdots \to A_k$ follow immediately from analogous properties of instances A_1 and A_k, respectively. For Optimistic Termination, assume that at most q processes are faulty and none of them executes *stop*. Each instance A_i satisfies Optimistic Termination $(q, 1)$: if all correct processes propose v to A_i, they will all decide on v in one step, and propose it to A_{i+1}. By simple induction, $A_1 \to \cdots \to A_k$ satisfies Optimistic Termination (q, k).

Theorems C.2 and C.3 prove that $A_1 \to \cdots \to A_{k\geq 2}$ requires:

Permanent Validity	$n > f + m + q$
Permanent Agreement	$n > f + m + q$

Since these requirements are the same for any $k \geq 2$, let us focus on the two-step OTC algorithm $A_1 \to A_2$, which satisfies Optimistic Termination $(q, 2)$.

For example, using this two-step OTC with $q = m = f$ in the Byzantine Consensus from Figure 6 results in a three-step algorithm that requires $n > 3f$, which matches the properties of [5].

6 Reconstructing Consensus algorithms

Figure 8 shows an (incomplete) list of Consensus algorithms that can be (re)constructed in the OTC framework. Each row specifies the maximum number of malicious processes m and the number of communication steps in which a decision is made in favourable runs (with at most q faulty processes).

The remaining columns provide information on how to construct a matching algorithm in the OTC framework. They show the type of the first round OTC and the resulting requirements on n. The OTCs used in other rounds are not shown because they do not affect the latency in favourable runs. For these rounds, I assume single-value one-step OTC in crash-stop settings and a multi-value two-step OTC in Byzantine settings. With $q = f$, these OTCs require $n > 2f$ and $n > 2f+m$, respectively, which are necessary to implement Consensus anyway [20]. Note that Cheap Paxos [21] also requires $n > 2f$ processes in general, but uses only $n' > f$ *primary* processes in the first round (which is the only round executed in favourable runs).

6.1 Eliminating the first coordinator

Some algorithms in Figure 8 require "1*" steps. In these, processes propose their Consensus proposals to OTC_1 directly, instead of waiting for the coordinator's proposal. This

Theorem	Opt. Termination	Proposals	Necessary condition	Shows optimality of
Theorem G.1	$(q_1, 1)$	single-value	$n > f + q_1 + 2m$	one-step single-value OTC
Theorem G.2	$(q_1, 1)$	multi-value	$n > f + 2q_1 + 2m$	one-step multi-value OTC
Theorem G.3	(q_k, k)	single-value	$n > f + q_k + m$	two-step OTC
Theorem G.4	$(q_1, 1)$ and $(q_2, 2)$	multi-value	$n > f + q_2 + m + \min\{q_1, m\}$	multi-step OTC

Figure 9: Summary of OTC lower boundsproved in Appendix G.

saves one communication step but decides in the first round only if all correct processes propose the same value. To tolerate runs in which processes propose different values, one must use a multi-value OTC_1, even in crash-stop settings. For example, multi-value one-step OTC with $m = 0$, $q = f$ allows us to reconstruct the one-step Consensus algorithm from [4]. Setting $q = 0$ leads to a new algorithm that requires only $n' > f$ primary processes and decides in one step provided that all primary processes are correct and propose the same value. The same technique with $m = f$, gives us two new analogous one-step Byzantine Consensus algorithms that require $n > 5f$ and $n > 3f$, respectively.

With no coordinator to fail, proposing to OTC_1 directly ensures that all correct processes execute $OTC_1.propose$. Assuming a one-step implementation of OTC_1, this is sufficient to ensure that lines 11–12 in Figure 4 will not block (Section 4). For this reason, $OTC_1.stop$ is not necessary any more, and lines 9–10 can be eliminated from round 1.

Eliminating $OTC_1.stop$ implies that the first round does not use failure detectors (which process would they monitor if there is no coordinator?) As a result, OTC_1, satisfying Optimistic Termination $(q, 1)$, does not require $q = f$ (Section 3.4). The same holds for the Byzantine model.

7 Multi-step OTC

There is a trade-off between OTC implementations: one-step OTCs are fast but require many processes, whereas two-step OTCs are slower but can work with fewer processes. This section presents the multi-step OTC algorithm, which satisfies the three Optimistic Termination conditions $(q_1, 1)$, $(q_2, 2)$, and $(q_3, 3)$ at the same time. It consists of three OTC chains from Section 5 executed in parallel:

$$\begin{array}{lllllll} A_1 & & & & & & \text{with } q = q_1, \\ B_1 & \rightarrow & B_2 & & & & \text{with } q = q_2, \\ C_1 & \rightarrow & C_2 & \rightarrow & C_3 & & \text{with } q = q_3. \end{array}$$

Instances A_1, B_1, and C_1 share the same monocast instance; each proposed value is proposed to all three chains at the same time. In other words, $propose(v)$ consists of $propose_{A_1}(v)$, $propose_{B_1}(v)$, and $propose_{C_1}(v)$. Stopping the algorithm involves stopping all six one-step OTC instances.

OTC predicates are defined as:

$$\begin{aligned} valid(v) &\overset{\text{def}}{=} valid_{A_1}(v) \lor valid_{B_1}(v) \lor valid_{C_1}(v) \\ decision(v) &\overset{\text{def}}{=} decision_{A_1}(v) \lor decision_{B_2}(v) \lor decision_{C_3}(v) \\ possible(v) &\overset{\text{def}}{=} \big(possible_{A_1}(v) \land \neg\exists\, v' \neq v : valid_{C_2}(v')\big) \\ &\quad \lor possible_{B_2}(v) \lor possible_{C_3}(v) \end{aligned}$$

In other words, the multi-step predicate $valid(v)$ is true if it holds for at least one of the instances A_1, B_1, C_1. Similarly, $decision(v)$ holds if it holds for at least one of A_1, B_2, C_3. Predicate $possible(v)$ is an improved version of the more natural definition

$$possible(v) \overset{\text{def}}{=} possible_{A_1}(v) \lor possible_{B_2}(v) \lor possible_{C_3}(v).$$

It states that v is a possible decision of A_1 only if $valid_{C_2}(v')$ holds for no $v' \neq v$. Indeed, honest processes propose v' to C_2 only after deciding on v' in C_1. Instances A_1 and C_1 share the same proposals and monocast instances, so they cannot reach different decisions v and v' (Lemma B.3).

The system of three chains A_1, $B_1 \to B_2$, $C_1 \to C_2 \to C_3$ implements OTC. Properties Integrity, Possibility, and Optimistic Termination (q_i, i) for $i = 1, 2, 3$ follow easily from the analogous properties of the individual chains. The same applies to Permanent Validity, which therefore requires

$$n > f + 2m + q_1, \quad n > f + m + q_2, \quad n > f + m + q_3.$$

Theorem D.1 shows that Permanent Agreement requires

$$n > f + 2m + 2q_1, \quad n > f + m + q_2 + \min\{m, q_1\}.$$

The former requirement ensures Permanent Agreement of instance A_1. The other is necessary for Permanent Agreement between decisions made by A_1 and the two other chains.

7.1 More Consensus algorithms

The multi-step OTC described in this section allows us to (re)construct the last three algorithms from Figure 8. In favourable runs, both [9, 30] decide in two steps if at most q processes are faulty, and in three steps otherwise. Algorithm [30] assumes that all faulty processes are malicious ($m = f$), whereas [9] works for any $m \leq f$. They can both be reconstructed by using multi-step OTC with $q_1 = q$ and $q_2 = q_3 = f$ for the first round. By setting $q_2 < q_3 = f$, one can construct a multi-step Byzantine Consensus algorithm that subsumes all known Byzantine Consensus algorithms, and is sometimes able to decide in the first round even when [9] cannot, for example: $n = 8$, $m = q = 1$, $q' = 2$, $f = 3$.

8 Lower bounds

Figure 9 summarizes four lower bound theorems proved in Appendix G. Each of them states the minimum number of processes n necessary to implement OTC with a given Optimistic Termination property. The theorems show that the requirements of OTC implementations presented in this paper are optimal.

9 Conclusion and future work

This paper introduced *Optimistically Terminating Consensus*, a variant of Consensus that decides if all correct processes propose the same value. Unlike condition-based Consensus [27], OTC is designed to be used to implement full Consensus protocols. For this reason, it provides stronger Validity and Agreement properties, which allow another OTC instance to take over if the current one failed.

OTC is simple to implement, even with malicious participants: processes broadcast their proposals and decide if sufficiently many processes report the same proposal. This simple one-step implementation is sufficient to reconstruct a large number of Consensus algorithms [4, 6, 16, 18, 19, 21, 24, 29]. By combining several instances of one-step OTC, one can reconstruct even more protocols [5, 9, 30]. New Consensus algorithms can be obtained: cheap one-step crash-stop Consensus, two variants of one-step Byzantine Consensus, and a Byzantine Consensus algorithm that subsumes [5, 9, 30]. For all OTC implementations presented in this paper, the number of processes n required is optimal.

In comparison to other agreement frameworks [1, 2, 3, 13, 17, 28], the OTC approach makes it possible to (re)construct the highest number of known and new Consensus algorithms. Firstly, OTC allows us to (re)construct Consensus protocols not only for the crash-stop model, but also for the much more difficult Byzantine model. Secondly, individual OTC instances are fully self-contained and independent, which gives us additional modularity and flexibility in designing agreement protocols. Thirdly, OTC implementations are simple and fully asynchronous, which makes it possible to discover new implementations automatically [31].

The concept of OTC greatly simplifies the design of new Consensus algorithms, especially in the most complicated, Byzantine, model. I also believe that with this overhead-free modular construction comes a deeper understanding of the Consensus problem itself. Encouraged by this, I envisage applying similar ideas to other agreement abstractions as a probable direction of my future work.

References

[1] Romain Boichat, Partha Dutta, Svend Frolund, and Rachid Guerraoui. Deconstructing Paxos. Technical Report DSC/2002/032, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, January 2001.

[2] Romain Boichat, Partha Dutta, Svend Frolund, and Rachid Guerraoui. Deconstructing Paxos. *ACM SIGACT News*, 34, 2003.

[3] Romain Boichat, Partha Dutta, Svend Frolund, and Rachid Guerraoui. Reconstructing Paxos. *ACM SIGACT News*, 34, 2003.

[4] Francisco Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. *Lecture Notes in Computer Science*, 2127:42–50, 2001.

[5] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pages 173–186, New Orleans, Louisiana, February 1999. USENIX Association.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[7] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving Consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[8] Assia Doudou, Benoît Garbinato, and Rachid Guerraoui. Encapsulating failure detection: From crash to Byzantine failures. In *Proceedings of the 7th International Conference on Reliable Software Technologies*, pages 24–50, June 2002.

[9] Partha Dutta, Rachid Guerraoui, and Marko Vukolic. Asynchronous Byzantine Consensus: Complexity, resilience and authentication. Technical Report 200479, EPFL, September 2004.

[10] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[11] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed Consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[12] Eli Gafni and Leslie Lamport. Disk Paxos. In *International Symposium on Distributed Computing*, pages 330–344, 2000.

[13] Rachid Guerraoui and Michel Raynal. The information structure of indulgent Consensus. Technical Report PI-1531, IRISA, April, 2003.

[14] Rachid Guerraoui and Michel Raynal. The alpha and omega of asynchronous Consensus. Technical Report PI-1676, IRISA, January 2005.

[15] Maurice P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 276–290, New York, NY, USA, 1988.

[16] Michel Hurfin and Michel Raynal. A simple and fast asynchronous Consensus protocol based on a weak failure detector. *Distributed Computing*, 12(4):209–223, 1999.

[17] Michel Hurfin, A. Mostfaoui, and Michel Raynal. A versatile family of Consensus protocols based on Chandra-Toueg's unreliable failure detectors. *IEEE Transactions Comput.*, 51(4):395–408, 2002.

[18] Klaus Kursawe. Optimistic asynchronous Byzantine agreement. Technical Report RZ 3202 (#93248), IBM Research, January 2000.

[19] Leslie Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):18–25, December 2001.

[20] Leslie Lamport. Lower bounds on asynchronous Consensus. In André Schiper, Alex A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 22–23. Springer, 2003.

[21] Leslie Lamport and Mike Massa. Cheap Paxos. In *Proceedings of 2004 International Conference on Dependable Systems and Networks*, pages 307–314, Florence, Italy, June 2004.

[22] Butler Lampson. The ABCD of Paxos. In *Proceedings of the twentieth Annual ACM Symposium on Principles of Distributed Computing*, page 13. ACM Press, 2001.

[23] Harry C. Li, Lorenzo Alvisi, and Allen Clement. The game of Paxos. Technical Report CS-TR-05-24, The University of Texas at Austin, Department of Computer Sciences, May 2005.

[24] Jean-Philippe Martin and Lorenzo Alvisi. Fast Byzantine Paxos. Technical Report TR-04-07, University of Texas at Austin, Department of Computer Science., 2004.

[25] Achour Mostéfaoui and Michel Raynal. Leader-based Consensus. Technical Report 1372, IRISA, 2000.

[26] Achour Mostéfaoui and Michel Raynal. Solving Consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 49–63, London, UK, 1999. Springer-Verlag.

[27] Achour Mostéfaoui, Sergio Rajsbaum, and Michel Raynal. Conditions on input vectors for Consensus solvability in asynchronous distributed systems. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 153–162. ACM Press, 2001.

[28] Achour Mostéfaoui, Sergio Rajsbaum, and Michel Raynal. A versatile and modular Consensus protocol. In *Proceedings of International IEEE Conference on Dependable Systems and Networks*, pages 364–373, 2002.

[29] André Schiper. Early Consensus in an asynchronous system with a weak failure detector. *Distributed Computing*, 10(3):149–157, April 1997.

[30] Piotr Zieliński. Paxos at war. Technical Report UCAM-CL-TR-593, Computer Laboratory, University of Cambridge, June 2004. Available at `http://www.cl.cam.ac.uk/TechReports/`.

[31] Piotr Zieliński. *Minimizing latency of agreement protocols*. PhD thesis, Computer Laboratory, University of Cambridge, 2006. Technical Report UCAM-CL-TR-667. Available at `http://www.cl.cam.ac.uk/TechReports/`.

A General OTC

Theorem A.1. *If an OTC algorithm satisfies Permanent Validity, Possibility, and Integrity, then it also satisfies Standard Validity.*

Proof. Consider a run r_1 in which some honest, but not necessarily correct, process p decides on v. To show Standard Validity, we have to prove that some honest process executed $propose(v)$.

Consider a run r_2 which is identical to r_1 except that, at some time t, after process p decided, all correct processes execute *stop* and no honest process executes any *propose* actions after that time. Runs r_1 and r_2 are identical until time t, so process p decides on v in run r_2 as well.

In run r_1, all correct processes execute *stop* at time t, so every correct process q will eventually enter a complete state. At that time, predicate $possible(v)$ must hold at q because process p decided on v (Possibility). Permanent Validity implies that $valid(v)$ must hold as well. Then, the Integrity property implies that an honest process executed $propose(v)$ in run r_2; this must have happened before time t because no honest process executed *propose* afterwards. Since runs r_1 and r_2 are identical until time t, the same honest process must have executed $propose(v)$ in run r_1 as well, which implies the assertion. □

Theorem A.2. *If an OTC algorithm satisfies Permanent Agreement and Possibility, then it also satisfies Standard Agreement.*

Proof. Consider a run r_1 in which some honest process p decides on v and another honest process p' decides on v'. To show standard Agreement, we have to prove that $v = v'$.

Consider a run r_2 which is identical to r_1 except that, at some time t, after both p and p' decided, all correct processes execute *stop*. Runs r_1 and r_2 are identical until time t, so processes p and p' decide on v and v', respectively, in run r_2 as well.

In r_2, all correct processes executed *stop* at time t, so every correct process q will eventually enter a complete state. At that time, predicates $possible(v)$ and $possible(v')$ must hold at q because processes p and p' decided on v and v', respectively (Possibility). Permanent Agreement implies the assertion ($v = v'$). □

B One-step OTC

Theorem B.1 (Strong Standard Validity). *Assume that $n > f+m+q$. If $decision(v)$ holds at an honest process, then $valid(v)$ holds at all complete processes.*

Proof. Predicate $decision(v)$ implies that at least $n-q-f$ *correct* processes have monocast v. Consider a complete process p. Every execution of *stop* involves monocasting, so p has monoreceived messages from all correct processes. At least $n - q - f > m$ of these messages are v, so $valid(v)$ holds. □

Theorem B.2 (Weak Permanent Validity). *Assume that $n > f+m+q$. If $possible(v)$ holds at a complete process, then an honest process executed $propose(v)$.*

Proof. Consider a complete process p. Every execution of *stop* involves monocasting, therefore p has monoreceived messages from all $n - f$ *correct* processes. Predicate *possible*(v) holds, so at most $q + m$ of these messages were different from v. Therefore, at least $n - f - q - m > 0$ of these messages were v, which means that at least one correct process monocast v, which implies the assertion. □

Lemma B.3 (Standard Agreement). *If $n > m + 2q$, then no two honest processes decide on different values.*

Proof. Assume *decision*(v) holds at some honest process p. Thus, p has monoreceived v from at least $n - q$ processes, which implies that at least $n - q - m$ honest processes proposed v. To obtain a contradiction, assume that *decision*(v) holds, possibly at different processes, for at least two different v. Since no honest process proposes two different values, $2(n-q-m) > n-m$ honest processes proposed something. This contradicts with the fact that there are only $n - m$ honest processes. □

Lemma B.4 (Standard Validity). *Assume $n > m + q$. If decision(v) holds at an honest process, then some honest process proposed v.*

Proof. Predicate *decision*(v) at an honest process, it must have monoreceived v from at least $n - q$ processes. Since $n - q > m$, at least one of these processes must be honest, which implies the assertion. □

C Two-step OTC

Lemma C.1. *Assume $n > f + m + q$ and consider a chain $A_1 \to \cdots \to A_k$ of one-step OTC instances A_i. If possible$_{A_k}(v)$ holds at a complete process, then an honest process proposed v in A_1.*

Proof. By induction on k. The base case $k = 1$ follows directly from Theorem B.2. For $k > 1$, if *possible*$_{A_k}(v)$ holds at a complete process, then the inductive assumption for the subchain $A_2 \to \cdots \to A_k$ implies that an honest process p proposed v to A_2. To execute *propose*$_{A_2}(v)$, process p must have decided on v in A_1. Since $n > f + m + q \geq m + q$, Lemma B.4 applied to A_1 implies the assertion. □

Theorem C.2 (Permanent Validity). *Assume $n > f + m + q$ and consider a chain $A_1 \to \cdots \to A_k$ with $k \geq 2$. For any complete process, possible$(v) \Rightarrow$ valid(v) for all v.*

Proof. Predicate *possible*$(v) \stackrel{\text{def}}{=}$ *possible*$_{A_k}(v)$, so Lemma C.1 applied to the subchain $A_2 \to \cdots \to A_k$ implies that an honest process p proposed v to A_2. To execute *propose*$_{A_2}(v)$, process p must have decided on v in A_1. Theorem B.1 implies the assertion. □

Theorem C.3 (Permanent Agreement). *Assume $n > f+m+q$ and consider a chain $A_1 \to \cdots \to A_k$ with $k \geq 2$. For any complete process, possible(v) holds for at most one v.*

Proof. Predicate *possible*$(v) \stackrel{\text{def}}{=}$ *possible*$_{A_k}(v)$, so Theorem B.2 applied to A_k implies that some honest process p proposed v in A_k. To execute *propose*$_{A_k}(v)$, process p must have decided on v in A_{k-1}. Since $n > f + m + q \geq m + 2q$, Lemma B.3 applied to A_{k-1} implies the assertion. □

D Multi-step OTC

Theorem D.1 (Permanent Agreement). *Assume that*

$$\begin{aligned} n &> f + 2m + 2q_1, \\ n &> f + \ \ m + q_2 + \min\{m, q_1\}, \\ n &> f + \ \ m + q_3. \end{aligned}$$

For any complete process, possible(v) holds for at most one v.

Proof. Predicate $\mathit{possible}(v)$ is defined as

$$\mathit{possible}(v) \stackrel{\text{def}}{=} \big(\mathit{possible}_{A_1}(v) \wedge \neg\exists\, v' \neq v : \mathit{valid}_{C_2}(v')\big) \\ \vee\, \mathit{possible}_{B_2}(v) \vee \mathit{possible}_{C_3}(v)$$

The assumption $n > f + 2m + 2q_1$ implies Permanent Agreement of A_1, whereas the other two assumptions imply Permanent Agreement of chains $B_1 \to B_2$ and $C_1 \to C_2 \to C_3$ (Theorem C.2). Therefore, predicates $\mathit{possible}_{A_1}(v)$, $\mathit{possible}_{B_2}(v)$, and $\mathit{possible}_{C_3}(v)$ can each hold for at most one v. To complete the proof, consider three values v_A, v_B, v_C, which – if they exist – satisfy

$$\begin{aligned} &\mathit{possible}_{A_1}(v_A) \wedge \neg\exists\, v' \neq v_A : \mathit{valid}_{C_2}(v'), \\ &\mathit{possible}_{B_2}(v_B), \\ &\mathit{possible}_{C_3}(v_C). \end{aligned}$$

We need to prove that all existing v_A, v_B, v_C must be the same. In other words, we have to show that $v_A = v_B$, $v_A = v_C$, and $v_B = v_C$.

- *Equality $v_A = v_C$.* Since $n > f + m + q_3$, Theorem C.2 states that the subchain $C_2 \to C_3$ satisfies Permanent Validity. Therefore, $\mathit{possible}_{C_3}(v_C) \Rightarrow \mathit{valid}_{C_2}(v_C)$, which implies $v_A = v_C$.

- *Equality $v_B = v_C$.* If $\mathit{possible}_{C_3}(v_C)$, then Lemma C.1 used for the subchain $C_2 \to C_3$ implies that an honest process proposed v_C to C_2, which implies that v_C was a decision in C_1. Similarly, Theorem B.2 applied to B_2 implies that v_B was a decision in B_1. The assumption $q_3 \geq q_2$ implies that $\mathit{decision}_{B_1}(v_B) \Rightarrow \mathit{decision}_{C_1}(v_B)$. Since $n > f + m + q_3 \geq m + 2q_3$, Lemma B.3 implies that $v_B = v_C$.

- *Equality $v_A = v_B$.* Showing $v_A = v_B$ requires considering two cases of the assumption $n > f + m + q_2 + \min\{m, q_1\}$:

  - *Case $n > f + m + q_2 + m$.* In this case, instance B_2 satisfies Permanent Validity. As a result, $\mathit{possible}_{B_2}(v_B) \Rightarrow \mathit{valid}_{B_2}(v_B) \Leftrightarrow \mathit{valid}_{C_2}(v_B)$, so $v_A = v_B$.
  - *Case $n > f + m + q_2 + q_1$.* Theorem B.2 applied to B_2 implies that v_B was a decision in B_1. This implies that at least $n - q_2 - f$ correct processes proposed v_B to B_1. On the other hand, $\mathit{possible}_{A_1}(v_A)$ at a complete process implies that at most $q_1 + m$ correct processes proposed a non-v_A to A_1. If $v_A \neq v_B$, then v_B is non-v_A, so $n - q_2 - f \leq q_1 + m$, which contradicts the assumption $n > f + m + q_1 + q_2$. □

E Crash-stop Consensus

Theorem E.1 (Termination). *Assume that all instances OTC_r satisfy Optimistic Termination (f, k) for some k. All correct processes will eventually decide.*

Proof. If some correct process p halts, then all correct processes will eventually decide. Indeed, p must have broadcast "decide on v" in line 15. This message will eventually be received by all correct processes, which will all decide on v and halt (lines 14–17). We can therefore assume that no correct process halts.

I will show that, in any round r started by all correct processes, either all of them will decide or proceed to the next round. Let p_c be the coordinator of round r. There are two cases:

1. No correct process ever suspects p_c after starting round r, which implies:

   - No correct processes ever executes $OTC_r.stop$.
   - By Strong Completeness of $\Diamond S$, coordinator p_c is correct. Therefore, all correct processes will eventually receive $phase_1(r, est_c)$ broadcast by p_c in line 5, and execute $OTC_r.propose(est_c)$.

   Therefore, Optimistic Termination (f, k) of OTC_r ensures that all correct processes will decide.

2. Some correct process suspects p_c after starting round r. This process will broadcast "stop round r", line 10, and as a result all correct processes will eventually execute $OTC_r.stop$. Thanks to Permanent Validity and Permanent Agreement of OTC_r, all correct processes will eventually satisfy the conditions in lines 11–12, and progress to round $r + 1$.

The Termination property can only be violated if Case 2 holds for all rounds r. This is impossible, however, because Weak Eventual Accuracy of $\Diamond S$ implies that there will eventually be a round r such that its coordinator p_c will never be suspected. □

F Byzantine Consensus

F.1 Validity

Lemma F.1. *If an honest process decides on v, then $OTC_r.decision(v)$ holds at some honest process for some round r.*

Proof. Let p be the first honest process to decide on v. Process p can decide in two cases (line 25):

1. It received messages "decide on v" from more than m processes. One of these message must come from an honest process that decided on v before, which conflicts with the definition of p as the first process to do so.

2. Predicate $OTC_r.decision(v)$ holds at p, which proves the assertion. □

Lemma F.2. *If all processes are honest, then any est_i has been proposed by some process.*

Proof. By induction on the round number r. At the beginning of the first round, $est_i = v_i$, the value proposed by process p_i. For the induction step, assume that the assertions hold for round r. We have to prove that the assignments in lines 18 and 22 preserve the assertion.

If "$est_i \leftarrow v$" in line 18 is performed, then a$OTC_r.possible(v)$ holds and so does $OTC_r.valid(v)$ (line 17). Integrity of OTC_r implies that some honest process p executed $OTC_r.propose(v)$. We assume all processes to be honest, so $v = est_c$, the estimate held by the coordinator p_{c_r} at the beginning of round r. By the inductive assumption, est_c has been proposed by some honest process.

The assignment $est_i \leftarrow v$ in line 22 is performed only if more than f processes report to have the same estimate v. Since $f \geq m$, at least one of these processes is honest, so the assertion holds. □

Theorem F.3 (Validity). *If all processes are honest and one of them decides on v, then v was proposed by some process.*

Proof. Lemma F.1 shows that $OTC_r.decision(v)$ holds at some process. By Standard Validity of OTC_r, at least one honest process must have executed $OTC_r.propose(v)$. We assume all processes to be honest, so $v = est_c$, the estimate held by the coordinator p_{c_r} at the beginning of round r. Lemma F.2 states that est_c has been proposed by some honest process, which proves the assertion. □

F.2 Agreement

Lemma F.4. *If, after line 18 of round r, all honest processes have the same estimate $est_i = est$, then the same will be true for round $r + 1$. Moreover, no honest process will execute $OTC_{r+1}.propose(v)$ for any $v \neq est$.*

Proof. Consider any honest process p. In line 20, p has received $f + m + 1$ messages $signed(r, est_j)$. At least $f + m + 1 - m > f$ of them come from honest processes, so they carry the same $est_j = est$. As a result, all honest processes p_i have $est_i = est$ at the end of round r (line 23).

In round $r + 1$, an honest process can execute $OTC_{r+1}.propose(v)$ for $v \neq est_i = est$, only if it $signed_c$ contains $> m$ signed estimates $est_j \neq est$. This is impossible, however, because all $signed(r, est_j)$ from honest p_j have $est_j = est$.

Finally, if any process executes $est_i \leftarrow v$ in line 18, then the $OTC_{r+1}.valid(v)$ holds, so an honest process executed $OTC_{r+1}.propose(v)$, which implies $v = est$. □

Theorem F.5 (Agreement). *If no two processes decide on different values.*

Proof. To obtain a contradiction, assume that two different values, v and v', become decisions. Lemma F.1 shows that both $OTC_r.decision(v)$ and $OTC_{r'}.decision(v')$ must hold at some processes p and p' and some rounds r and r'. If $r = r'$, then Standard Agreement of $OTC_r = OTC_{r'}$ guarantees that $v = v'$. Therefore, $r \neq r'$, and without loss of generality, we can assume $r < r'$.

Since $OTC_r.decision(v)$ holds at some process, $OTC_r.possible(v)$ holds at all processes at all times. In particular, each honest process p_i will execute line 18, and have $est_i = v$. By inductive application of Lemma F.4, we get that no honest process proposes anything other than v to $OTC_{r'}$. Standard Validity of $OTC_{r'}$ implies the assertion. □

F.3 Termination

The algorithm assumes that $n > 2f + m$.

Lemma F.6. *If a correct process halts, then all correct processes will eventually decide and halt.*

Proof. The assumption implies that some process received more than $f + m$ messages "decide on v". More than m of them must have been sent by correct processes, so all $n-f$ correct processes will eventually receive these messages, decide on v, and broadcast "decide on v" (line 25). Therefore, all correct processes will eventually receive $n - f > f + m$ such messages, and halt. □

Lemma F.6 showed that Termination is ensured if at least one correct process halts. Thus, to prove Termination, we can assume that no correct process ever halts.

Lemma F.7. *Assume that more than m correct processes executed $OTC_r.stop$. Then, all correct processes will start round $r + 1$ within three communication steps.*

Proof. Let us measure time in communication steps, with 0 being the current time. By time 1, all correct processes will have received more than m messages "stop round r", and executed $OTC_r.stop$ (lines 13–14). Permanent Validity and Permanent Agreement of OTC_r guarantee that the conditions in lines 16–17 will be satisfied at all $n - f$ correct processes by time 2. Therefore, all of them will have received $n - f > f + m$ messages $signed(r, est_j)$ in line 20 and started $timer_r$ by time 3. □

Lemma F.8. *All correct processes start round r within three communication steps from the first correct process to do so.*

Proof. Since all processes start the Consensus algorithm at the same time, the assertion obviously holds for $r = 1$. For any $r > 1$, consider the first correct process p to start $timer_r$. In round $r-1$, process p must have received more than $n+f$ messages $signed(r-1, est_j)$, out of which more than m were sent correct processes (line 19). Therefore, these m correct processes must have broadcast "stop round $r - 1$" in line 15. Lemma F.7 concludes the proof. □

Lemma F.9. *Let r be any round with a correct coordinator p_{c_r}, and OTC_r satisfying Optimistic Termination (f, k) for some k. If p_{c_r} ever starts round r and the timeout for $timer_r$ is sufficiently long, then all correct processes will decide in that round.*

Proof. Let p be the first correct process to start $timer_r$. Let us measure time in communication steps, with the start of $timer_r$ by process p as time 0.

Lemma F.8 ensures that the coordinator c_k starts its $timer_r$ and broadcasts message $phase_1(r, est_c, signed_c)$ by time 3. By time 4, all correct processes receive this message, and execute $OTC_r.propose(est_c)$, provided that the test in lines 10–11 does not fail (see below). Optimistic Termination (f, k) satisfied by OTC_r ensures that all correct processes will decide by time $4 + k$. This of course requires that no correct process executes $OTC_r.stop$ before; for this reason I assume that the timeout for $timer_r$ is "sufficiently long", which means longer than $4 + k$ communication steps.

We still have to show that the test in lines 10–11 will not fail. This is obvious for $r = 1$. For $r > 1$, consider the set $signed_{c_r}$ at the end of round $r - 1$ at the coordinator p_{c_r}. The conditional assignment in line 22 ensures that no value, except possibly est_{c_r}, occurs in $signed_{c_r}$ more than f times. Therefore, for any $est_i \neq est_{c_r}$ in line 11 of round r, the set $signed_c$ will contain at least $f + m + 1 - f > m$ estimates $est_j \neq est_i$, which proves the assertion. □

Lemma F.10. *All correct processes will eventually start each round r.*

Proof. By induction on r. Obvious for $r = 1$. If all correct process starts round r, then all their timers will eventually expire and they will eventually execute $OTC_r.stop$ in line 15. The assertion follows from Lemma F.7. □

Theorem F.11 (Termination). *Assume all, except for possibly a finite number, instances OTC_r satisfy Optimistic Termination (f, k) for some k. All correct processes will eventually decide and halt.*

Proof. To obtain a contradiction, assume that no correct process ever halts (Lemma F.6). Lemma F.10 shows that all correct processes will start all rounds r.

Timeouts for successive rounds grow indefinitely. Therefore, there will eventually be a round r with a correct coordinator p_{c_r}, with OTC_r satisfying Optimistic Termination (f, k), and with the timeout longer than $4 + k$ communication steps. Lemma F.9 states that all correct processes will decide in round r. □

G Lower bounds

All the proofs share a similar structure. They assume there is an OTC algorithm that does not require the given condition, and construct a sequence of runs that leads to a contradiction. The runs are illustrated with standard message-exchange diagrams. All messages shown have the same delay d; the messages not shown are delayed by the system and arrive at their destinations after all events shown in the diagram occurred. The diagrams use ✖ for crash events and ○ for malicious behaviour. Some of the proof ideas are borrowed from [9].

All the lower bound in this section assume $f, q > 0$. The case $f = 0$ corresponds to solutions that are not fault-tolerant. If $q = 0$, all the bounds become weaker than $n > 2f + m$, which is necessary to solve Consensus anyway [20].

To provide stronger results, the proofs in this section assume a weaker version of the Optimistic Termination conditions which additionally assumes that no honest process proposes anything other than v.

A process is *semi-complete* if $possible(v)$ holds for at most one v and $valid(v) \Rightarrow possible(v)$ for all v. Permanent Validity and Permanent Agreement imply that every complete process is semi-complete.

Theorem G.1. *Any single-value OTC algorithm satisfying Optimistic Termination* $(q_1, 1)$ *requires* $n > f + q_1 + 2m$.

Proof. To obtain a contradiction, consider a one-step single-value OTC algorithm with $n \leq f + q_1 + 2m$. Figure 10 shows four runs of this algorithm. Processes have been divided into four groups: Q, F, M_1, M_2, with sizes of at most q_1, f, m, m, respectively. Sets Q and F are not empty. In all runs, all processes from the same group behave identically.

In run r_1, processes in Q crash at time 0, and all the other processes are correct, propose 1, and send their messages to processes F. (Other messages sent are, as explained before, significantly delayed.) Since at most q_1 processes failed, Optimistic Termination $(q_1, 1)$ requires processes F to decide in, what F perceive as, one communication step (by time d).

In run r_2, all processes are correct, except for those in F, which crash at the beginning. Only processes in group M_1 propose (1), the others do not propose anything. At some time $t > d$, all correct processes execute *stop*. At time $t + d$, processes Q have received all messages sent by correct processes at time t or before. Permanent Validity and Permanent Agreement imply that processes Q are semi-complete ($possible(v)$ holds for at most one v and $valid(v) \Rightarrow possible(v)$ for all v).

In run r_3, all processes are correct, except for those in M_2. Processes in M_2 are malicious and send a message to F claiming that they proposed 1, whereas in fact they did not propose anything. Apart from that, processes M_2 behave correctly. Processes F and M_1 propose 1 and send messages to processes F.

At time d, processes F cannot distinguish r_3 from r_1, so they decide on 1. Now, consider the state of processes Q at time $t + d$. Predicate $possible(1)$ holds because processes F decided on 1 (Possibility). Processes Q cannot distinguish r_3 from r_2, so their states are semi-complete. This implies $possible(1) \Rightarrow valid(1)$, so $valid(1)$ holds as well.

Finally, in run r_4, all processes are correct except for those in group M_1. No process proposes anything, but processes in M_1 maliciously behave as if they had proposed 1. All processes except for F execute *stop* at time t. At time $t + d$, processes Q cannot distinguish runs r_4 and r_3, so $valid(1)$ holds. This violates Integrity, because no (honest) process proposed 1 in this run. □

Theorem G.2. *Any multi-value OTC algorithm satisfying Optimistic Termination* $(q_1, 1)$ *requires* $n > f + 2q_1 + 2m$.

Proof. To obtain contradiction, consider a one-step multi-value OTC algorithm with $n \leq f + 2q_1 + 2m$. Figure 10 shows five runs of the algorithm. Processes have been divided into five groups: Q_1, Q_2, F, M_1 and M_2 with sizes of at most q_1, q_1, f, m, and m, respectively. Sets Q_1 and F are not empty. In all runs, all processes from the same group behave identically.

In run r_1, all processes are correct, except for F, which crash at time 0. Processes Q_1 and M_1 propose 0, whereas processes in Q_2 and M_2 propose 1. At some time $t > d$, all correct processes execute *stop*. Permanent Validity and Permanent Agreement imply that processes Q_1 are semi-complete at time $t + d$

In run r_2, all processes are correct and propose 1 and send messages to F, except for those in group Q_1, which crash at time 0 without proposing anything. Optimistic

Termination $(q_1, 1)$ requires processes F to decide on 1 in one communication step, that is, by time d.

In run r_3, all processes are correct, except for those in M_1, which are malicious. Processes Q_2, F, and M_2 propose 1 and send messages to F. Processes in Q_1 and M_1 propose 0, but M_1 maliciously send messages to F claiming they have proposed 1; otherwise processes M_1 behave correctly. At time t, all processes execute *stop*, except for those in group F.

At time d, processes F cannot distinguish runs r_3 and r_2, so they decide on 1. At time $t + d$, processes Q_1 cannot distinguish runs r_3 and r_1, so they enter semi-complete states. Predicate *possible*(1) holds because processes F decided on 1 (Possibility).

In run r_4, all processes are correct and propose 0 and send messages to F, except for those in group Q_2, which crash at time 0 without proposing anything. Optimistic Termination $(q_1, 1)$ requires processes F to decide on 0 by time d.

In run r_5, all processes are correct, except for those in M_2, which are malicious. Processes Q_1, F, and M_1 propose 0 and send messages to F. Processes in Q_2 and M_2 propose 1, but M_2 maliciously send messages to F claiming that they have proposed 0; otherwise processes M_2 behave correctly. At time t, all processes execute *stop*, except for those in group F.

At time d, processes F cannot distinguish runs r_5 and r_4, so they decide on 0. At time $t + d$, processes Q_1 cannot distinguish runs r_5 and r_1, so they enter semi-complete states. Predicate *possible*(0) holds because processes F decided on 0.

At time $t + d$ processes Q_1 cannot distinguish runs r_3 and r_5, so in both cases they are semi-complete states with both *possible*(0) and *possible*(1) holding. This violates the definition of semi-completeness. □

Theorem G.3. *Any single-value OTC algorithm satisfying Optimistic Termination* (q_k, k) *requires* $n > f + q_k + m$.

Proof. To obtain contradiction, consider a single-value OTC algorithm with $n \leq f + m + q_k$. Figure 12 shows three runs of this algorithm. Processes have been divided into three groups: F, M, Q with sizes of at most f, m, q_k, respectively. Sets Q and F are not empty. In all runs, all processes from the same group behave identically.

In run r_1, all processes are correct and propose 1, except for those in group Q, which crash at time 0 and propose nothing. Optimistic Termination (q_k, k) requires that processes F eventually decide on 1, say at time t_1.

Run r_2 is the same as r_1, except that processes Q do not crash. At time t_1, processes F cannot distinguish r_2 and r_1, so they decide on 1. At some time $t > t_1$, processes F crash, and processes M and Q execute *stop*. Permanent Validity and Permanent Agreement imply that processes Q enter semi-complete states at time $t + d$. Predicate *possible*(1) holds at processes Q because processes F decided on 1, and semi-completeness implies that *valid*(1) holds as well.

In run r_3, all processes except M are correct. No processes propose anything. Processes M maliciously behave as in run r_2, for example, by claiming that processes F reported to have proposed 1. At time t, processes M and Q execute *stop*. At time $t + d$, processes Q cannot distinguish runs r_3 and r_2, so *valid*(1) holds. This violates Integrity, as no process proposed anything in r_3. □

Theorem G.4. *Any OTC algorithm satisfying Optimistic Termination* $(q_1, 1)$ *and* $(q_2, 2)$ *requires* $n > f + m + q_2 + \min\{q_1, m\}$.

Proof. To obtain contradiction, consider an OTC algorithm satisfying Optimistic Termination $(q_1, 1)$ and $(q_2, 2)$ with $n \leq f + m + q_2 + \min\{q_1, m\}$. Figure 13 shows five runs of this algorithm. Processes have been divided into four groups: F, M, Q_2 and MQ_1 with sizes of at most f, m, q_1, and $\min\{m, q_1\}$, respectively. Sets Q_2 and F are not empty. In all runs, all processes from the same group behave identically.

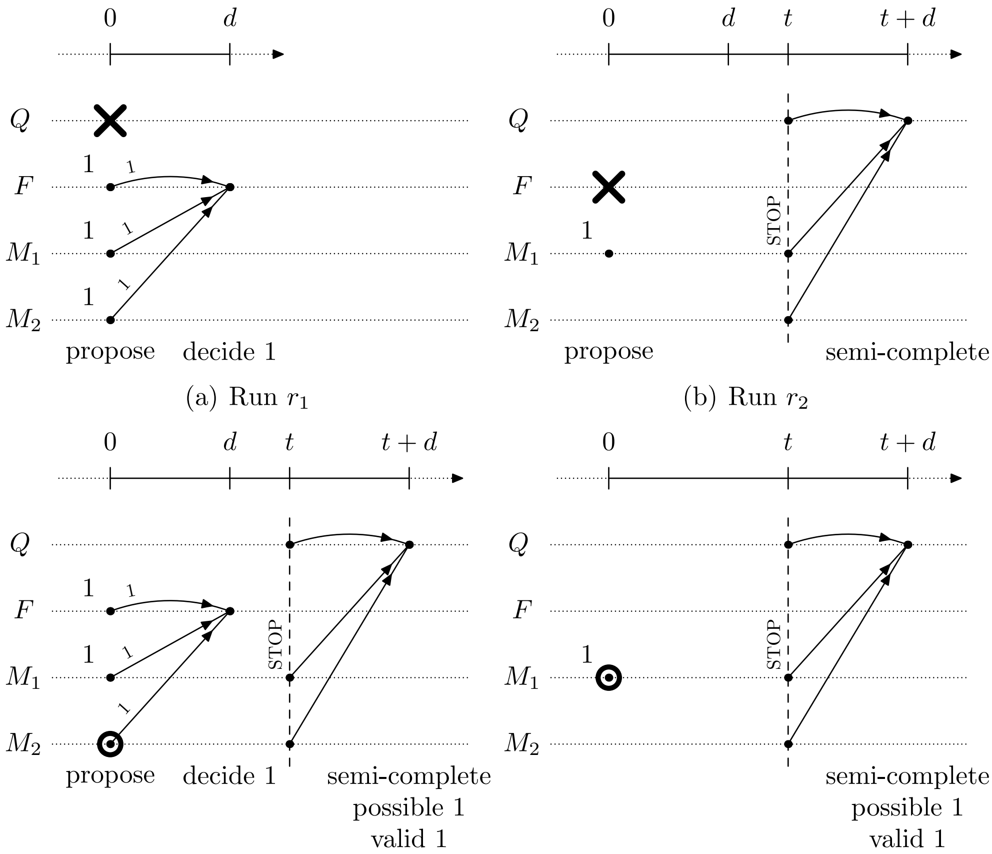
In run r_1, all processes, except for MQ_1, are correct, propose 1 and send their messages to F. Processes MQ_1 crash at time 0 without proposing anything. Optimistic Termination $(q_1, 1)$ makes processes F decide on 1 at time d.

In run r_2, all processes are correct, except for M, which are malicious. Processes F and Q_2 propose 1 and send their messages to F. Processes MQ_1 propose 0 and send their messages to M. Malicious processes M propose 0 but send messages to F claiming that they have proposed 1. At time d, they behave as if they received 0 from F, otherwise they behave correctly. At some time $t > 2d$, all processes except for F execute *stop* and send messages to Q_2. At time d, processes F cannot distinguish runs r_1 and r_2, so they decide on 1 in both of them. Thus, *possible*(1) holds at all processes at all times (Possibility).

In run r_3, processes Q_2 crash at time 0. All other processes are correct and propose 0. No first round messages, nor second round messages to F, are delayed. Optimistic Termination $(q_2, 2)$ makes processes F decide on 0 by time $2d$.

Run r_4 is similar to r_3, except that processes in Q are correct, propose 1, but do not send any messages. Processes MQ_1 are malicious and pretend, to all processes other than F, that they have neither received nor sent any messages at time d. At time t, all processes except for F execute *stop* and send messages to Q_2. At time $2d$, processes F cannot distinguish runs r_3 and r_4, so they decide on 0 in both of them. Thus, *possible*(0) holds at all processes at all times (Possibility).

In run r_5, all processes, except for Q_2, propose 0 and send their messages to M. Processes Q_2 propose 1 and do not send anything. All processes are correct, except those in F, which crash at time d. At time t, all correct processes execute *stop* and send messages to Q_2. As a result, Permanent Validity and Permanent Agreement imply that the states of processes Q_2 at time $t+d$ are semi-complete. Processes Q_2 cannot distinguish runs r_2, r_4, and r_5, so both *possible*(0) and *possible*(1) hold, which violates the definition of semi-completeness. □



(a) Run r_1 (b) Run r_2 (c) Run r_3 (d) Run r_4

Figure 10: Runs examined in the proof of Theorem G.1.

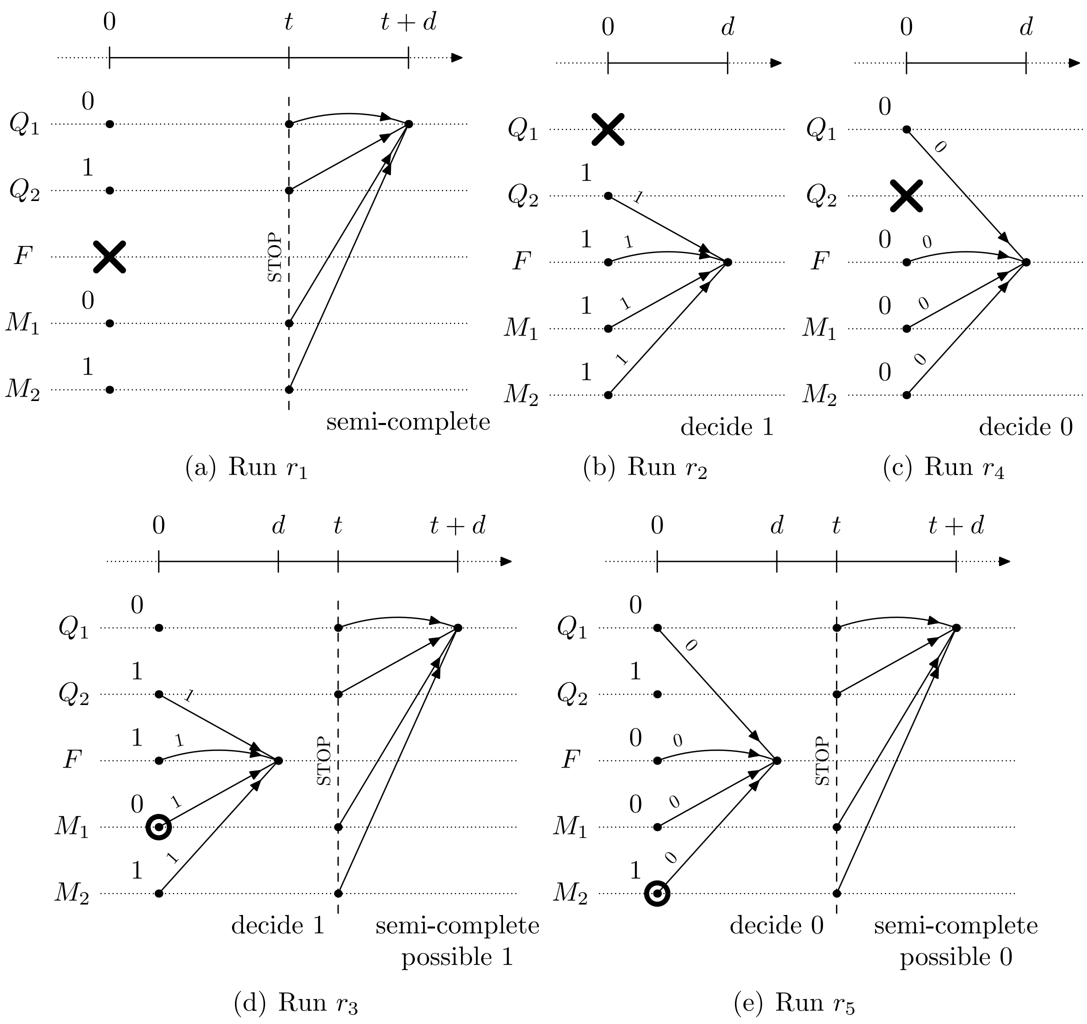
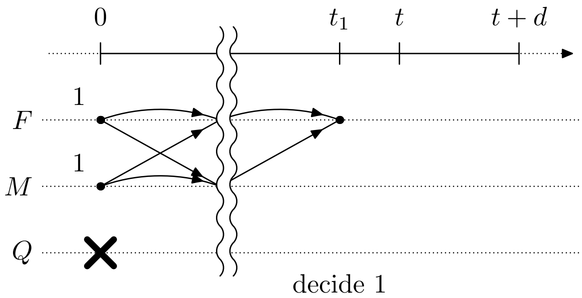
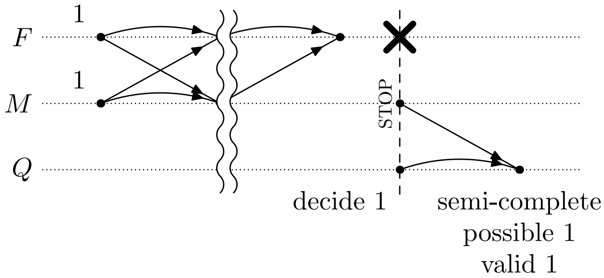


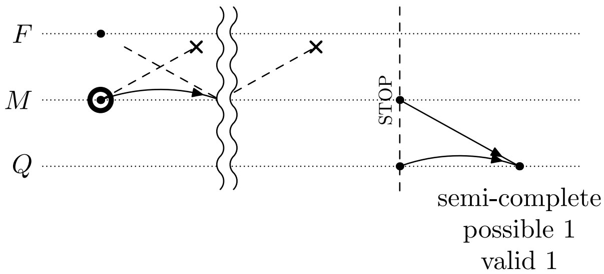
Figure 11: Runs examined in the proof of Theorem G.2.



(a) Run r_1.



(b) Run r_2.



(c) Run r_3.

Figure 12: Runs examined in the proof of Theorem G.3.

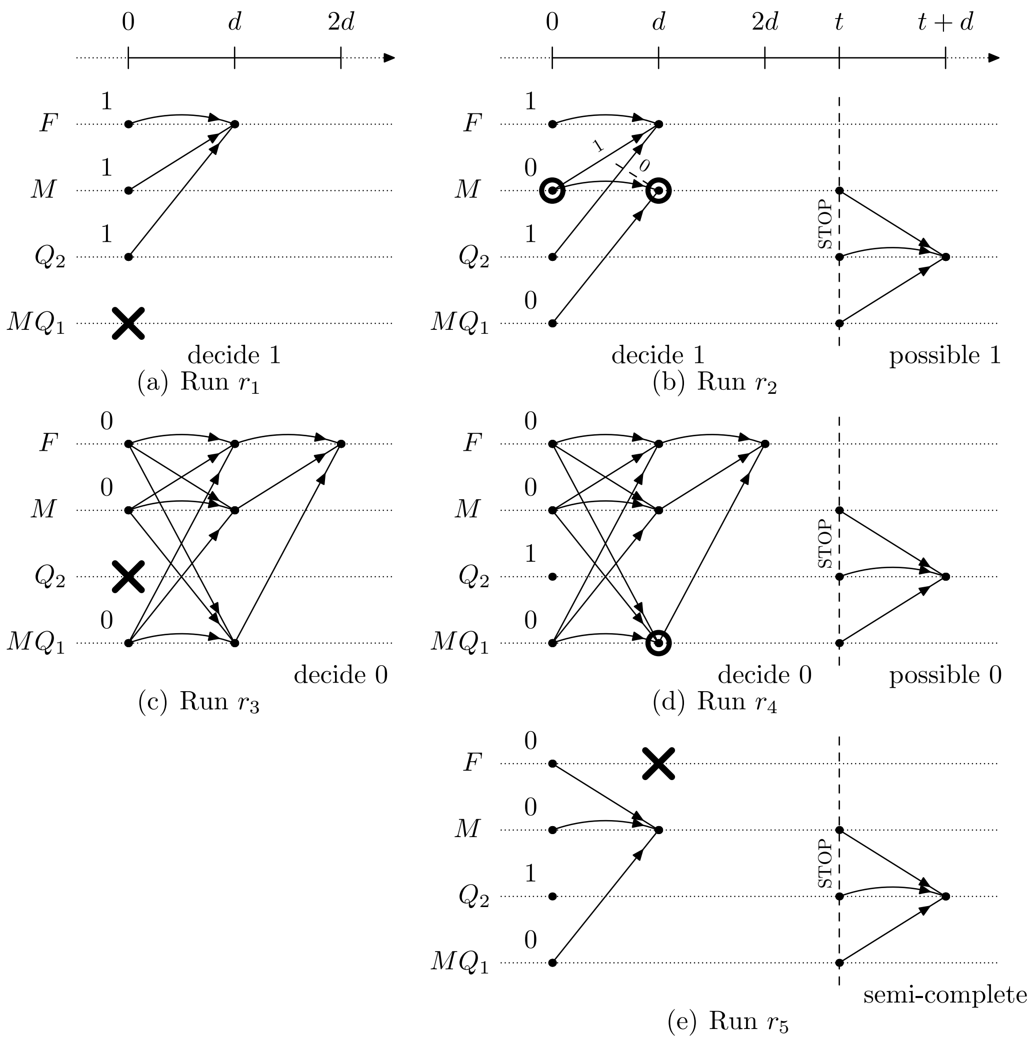


Figure 13: Runs examined in the proof of Theorem G.4.