Bounds on Worst-Case Responsiveness for Agreement Algorithms

Hagit Attiya
Department of Computer Science, Technion, Haifa, Israel

Jennifer L. Welch
Department of Computer Science and Engineering, Texas A&M University,
College Station, TX, USA

Abstract

We study the worst-case time complexity of solving two agreement problems, consensus and broadcast, in systems with n processes subject to no more than t process failures. In both problems, correct processes must decide on a common value; in the consensus problem, each process has an input and if the inputs of correct processes are all the same, then that must be the common decision, whereas in the broadcast problem, only one process (the sender) has an input and if the sender is correct, then its input must be the common decision. We focus on systems where there is an upper bound Δ on the message delivery time but it is expected that typically, messages arrive much faster, say within some time d. While Δ may or may not be known in advance, d is inherently unknown and specific to each execution. The goal is to design deterministic algorithms whose running times have minimal to no dependence on Δ, a property called *responsiveness*.

We present a generic algorithm transformation that, when applied to appropriate eventually-synchronous consensus (or broadcast) algorithms, results in consensus (or broadcast) algorithms for send omission failures, authenticated Byzantine failures, and unauthenticated Byzantine failures whose running times have no dependence on Δ; their worst-case time complexities are all $O(td)$, which is asymptotically optimal. The algorithm for send omission failures requires $n > 2t$, while those for Byzantine failures, both authenticated and unauthenticated, require $n > 3t$. The failure-resilience of the unauthenticated Byzantine algorithm is optimal.

For authenticated Byzantine failures, existing agreement algorithms provide worst-case time complexity $O(t\Delta)$ when n is at most $3t$. (When $n \leq 2t$, broadcast is solvable while consensus is not.) We prove a lower bound on the worst-case time complexity of $\lfloor (3t-n)/2 \rfloor d + \Delta$ when n is at most $3t$. Although lower bounds of Δ and $(t+1)d$ were already known, our new lower bound indicates that, at least when $n \leq 2t$, it is impossible for an algorithm to pay these bounds in parallel.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases bounded-delay model, basic round model, omission failures, Byzantine failures

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.32

Funding *Hagit Attiya*: partially supported by the Israel Science Foundation (grant 22/1425).

1 Introduction

Reaching agreement in the presence of faulty processes is a fundamental problem in distributed computing, with applications ranging from control systems and databases to cloud storage and blockchains. We consider two variations of reaching agreement. In the *consensus* problem, every process begins with an input and every correct process must reach the same decision; if all the correct processes have the same input, then that must be the common decision, a condition called *strong unanimity*. In the *broadcast* problem, there is a distinguished "sender" process that begins with an input and every correct process must reach the same decision; if the sender is correct, then the sender's input must be the common decision. The consensus and broadcast problems are closely related to each other, and in fact can be transformed to

27th International Conference on Principles of Distributed Systems (OPODIS 2023).
Editors: Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi; Article No. 32; pp. 32:1–32:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

each other in most situations very simply: Broadcast can be solved by having the sender send its input to all the processes and then having the processes run a consensus algorithm with each process' input being the value received from the sender. Consensus can be solved by using one copy of a broadcast algorithm for each process to disseminate its input and then having each process apply a common function to the vector of broadcast decisions to obtain the consensus decision.

We consider *send omission* failures, in which faulty processes fail to send some messages, and *Byzantine* (i.e., malicious) failures, in which faulty processes can change state arbitrarily and send messages with arbitrary content. Cryptographic developments allow the assumption that processes have access to *authentication* primitives to mitigate the effects of Byzantine failures. Note that if half or more of the processes are Byzantine, then the consensus problem, as stated above, is not solvable, and we only consider the broadcast problem. In this case, the transformations between consensus and broadcast discussed above work if we consider a weaker version of consensus in which the decision must be the common input only if all the processes are correct, a validity condition called *weak unanimity.* We assume a *permissioned* model, in which there is a known fixed set of n processes, up to t of which can be faulty.

Agreement problems have been studied under a variety of assumptions about the timing behavior of the system. Initially, the lock-step rounds model was considered (e.g., [25]), and then extended to more realistic synchronous models, with a known upper bound on message delay Δ. Relaxations of this model include when Δ is not known and when Δ only starts holding after some unknown *global stabilization time* (GST) [13]. In the asynchronous model, there is no upper bound on the message delays (cf. [16]).

We consider a *bounded-delay* system model, where message delay is at most Δ. Motivated by real-world networks in which the message delays in an execution are typically much smaller than the maximum possible delay Δ, researchers looked for algorithms whose performance is adaptive with respect to the maximum message delay actually experienced in each execution, call it d. Herzberg and Kutten [19] proposed this style of analysis and gave an algorithm to detect message forwarding faults with good performance under it. This behavior is dubbed *responsiveness* by Pass and Shi [23].

When Δ is known, classic agreement algorithms for the lock-step rounds model, with optimal $t+1$ rounds (e.g., [12]), can be adapted by using Δ time to simulate each round, but this leads to algorithms with $\Theta(t\Delta)$ worst-case running times. In the other direction, the $t+1$ round lower bound of [12] for agreement implies a lower bound of $\Omega(td)$ on the worst-case running time.

This paper investigates the possibility of avoiding or minimizing dependence on Δ in the worst-case running time for consensus and broadcast algorithms. In situations where d is much smaller than Δ, this would provide a significant speedup, allowing faster agreement on various decisions and actions.

Our first result (Theorem 2) is an algorithm transformation that takes any consensus or broadcast algorithm designed for a specific timing model called the *basic round model* [13] and produces an algorithm for the bounded-delay model with responsiveness that depends only on d. In the basic round model, processes take steps in lock-step rounds, messages are only received in the round in which they are sent, and after some unknown round number, called GST, no messages are lost. The transformed algorithm solves the same problem as the input algorithm, tolerates the same number and type of process failures, and has running time $O(Td)$, where T is the number of rounds after GST required for the original algorithm to decide. The transformed algorithm does not require that Δ be known, or that it even exist; that is, it could be that there is a bound d on the message delay in each particular execution, but no global bound over all executions.

Table 1 Time complexity lower bounds for several values of n and t, ignoring floors.

	$n = t + 2$	$n = \frac{4}{3}t$	$n = \frac{3}{2}t$	$n = 2t$	$n = 2t + 1$	$n = 3t - 1$	$n = 3t$
Thm. 9	$(t-1)d + \Delta$	$\frac{5t}{6}d + \Delta$	$\frac{3t}{4}d + \Delta$	$\frac{t}{2}d + \Delta$	$\frac{t-1}{2}d + \Delta$	$\frac{1}{2}d + \Delta$	Δ
[4]	$\frac{t}{2}\Delta$	3Δ	2Δ	Δ	N/A	N/A	N/A

The transformation is inspired by an algorithm of Dwork and Stockmeyer [14] for crash failures. We have generalized the approach so that it works for send omission and Byzantine failures, both with and without authentication. The main algorithmic idea is to run the simulated algorithm, periodically doubling an estimate of the message delay, until we eventually reach the actual message delay bound d, at which point the GST round is reached for the simulation.

By applying our transformation to relevant algorithms of Dwork, Lynch and Stockmeyer [13] for the basic round model, we obtain consensus (and broadcast) algorithms for send omission failures and Byzantine failures, both with and without authentication, that have asymptotically optimal worst-case running time of $O(td)$. To the best of our knowledge, these are the first algorithms for these types of failures with asymptotically optimal worst-case responsiveness.

The resulting algorithm for send omission failures requires $n > 2t$, while the other two require $n > 3t$. The $n > 3t$ resilience requirement is optimal for unauthenticated Byzantine failures [25]. For authenticated Byzantine failures, the resilience of $n > 3t$ needed for our algorithm is not optimal, as there are existing algorithms for consensus and broadcast that work with a larger fraction of faulty processes. In fact, the broadcast problem is solvable even when $n = t + 1$, as demonstrated by the algorithm of Dolev and Strong [12]. As mentioned earlier, consensus can be solved by using one copy of a broadcast algorithm for each process; however, if processes can be Byzantine when using this scheme, the strong unanimity validity condition requires a majority of correct processes, i.e., $n > 2t$. This requirement is inherent, as a simple partitioning argument (see [1]) shows that $n > 2t$ is a necessary condition for solving consensus with strong unanimity in the presence of Byzantine failures, even with authentication.

In an attempt to address the time complexity gap when $n \leq 3t$ in the case of authenticated Byzantine failures, we prove a lower bound (Theorem 9) when n is in the range $t + 2$ to $3t$ that the worst-case time complexity of consensus, even with weak unanimity must be at least $\lfloor (3t - n)/2 \rfloor d + \Delta$. Abraham, Nayak, Ren, and Xiang [4] showed a lower bound of $(\lfloor n/(n-t) \rfloor - 1)\Delta$ on the running time of broadcast for $n \leq 2t$. These two lower bounds are incomparable, depending on the relative values of t, d, and Δ; see Table 1. These lower bounds hint that the smaller n is compared to $3t$, the larger the reliance on the timeout must be. Ignoring constants, and looking at the key breakpoints, they indicate that when $n = t + 2$, the time complexity is $\Omega(t\Delta)$, which is asymptotically tight, and when $n = 2t + 1$, it is $\Omega(td + \Delta)$ compared with $\Theta(td)$ when $n = 3t + 1$.

Our focus on the *worst-case* responsiveness of broadcast and consensus algorithms, complements recent results for *optimistic* responsiveness, when the sender and a majority of the processes are correct, discussed in detail in the next section.

2 Related Work

Analyzing the running time of an algorithm for the bounded-delay model in terms of both d, the actual delay of messages, and Δ, the upper bound to time out message delivery, was proposed by Herzberg and Kutten [18,19] in the context of message communication protocols. Subsequently, the idea was applied to consensus algorithms.

Table 2 Worst-case time complexity bounds for n processes with t crash failures.

	$t+1 < n \leq 2t$	$2t < n$
upper bound	$O(td + \Delta)$ [5,6,21]	$O(td)$ [14]
lower bound	$(2t-n)d + \Delta$ [14]	$(t+1)d$

Table 3 Worst-case time complexity bounds for n processes with t send omission failures.

	$t+1 < n \leq 2t$	$2t < n$
upper bound	$O(\min\{t/(n-t), \sqrt{\Delta/d}\}td + \Delta)$ [26]	$O(td)$ (Cor. 3)
lower bound	$(2t-n)d + \Delta$ (cf. Table 2)	$(t+1)d$

Attiya, Dwork, Lynch and Stockmeyer [6,7] presented a *crash-tolerant* algorithm for consensus in the bounded-delay model with worst-case running time of $O(td + \Delta)$, for $n > t$. Constant-factor improvements were made in [5,21]. Dwork and Stockmeyer [14] improved on this running time when $n > 2t$ with an algorithm whose worst-case time complexity is $O(td)$, which is asymptotically optimal; they also showed that when $n \leq 2t$ no algorithm can have worst-case time better than $(2t-n)d + \Delta$. Thus dependence on Δ, that is, requiring at least one time-out, is necessary if and only if $n \leq 2t$. See Table 2.[1]

Ponzio [26] extended the results of [6,7] to *send omission failures* in the bounded-delay model, presenting a consensus algorithm for $n > t$ with worst-case time complexity $\Omega(td+\Delta)$. Bharali and Berman [9] further extended [26] to general omission failures as long as $n > 2t$, with the same asymptotic running time. Similar results are derived in a more structured manner by Attiya, Borran, Hutle, Milosevic and Schiper [5]. To the best of our knowledge, all previous algorithms for (send) omission failures in the bounded-delay model have worst-case running times that depend on Δ. In contrast, our algorithmic transformation, when applied to an appropriate base algorithm, yields an algorithm for send omission failures, where $n > 2t$, with worst-case time complexity $O(td)$, that is, with no dependence on Δ, which is asymptotically optimal. See Table 3.

For *authenticated Byzantine failures*, there is an algorithm for $n > 3t$ with worst-case running time of $O(d^2 + t^2)$ [13]. It was designed for the partially synchronous model when Δ is unknown and thus it uses estimates of the observed actual message delays instead of Δ timeouts. It works in the bounded-delay model as well with no dependence on Δ, although the resilience is no longer optimal.[2] A recent paper of Civit et al. [11] presents a communication-optimal algorithm for the partially synchronous model in which Δ holds eventually (i.e., the eventually-synchronous model). It requires $n > 3t$ and has worst-case running time of $O(t\Delta)$ after Δ starts holding. This algorithm also works in the bounded-delay model and has worst-case running time of $O(t\Delta)$ and optimal communication complexity, although the resilience is no longer optimal. Our results on worst-case running time together with the upper and lower bounds that follow from the classical round-based results are summarized in Table 4.

[1] The lower bounds in all the tables implicitly include the maximum of the expression given and $(t+1)d$, which is due to the $(t+1)$-round lower bound when $n > t+1$ [12].

[2] Algorithm 2^2 in [13, Section 4.2]. The discussion after Theorem 4.2 states $O(N^2 + \Delta^2)$, but N (number of processes) can be reduced to t by allowing multiple messages to be sent in each round, and Δ corresponds to our d as their algorithm is for the unknown-Δ model.

Table 4 Worst-case time complexity bounds for n processes with t authenticated Byzantine failures (only broadcast in the range $t+1 < n \leq 2t$).

	$t+1 < n \leq 2t$	$2t < n \leq 3t$	$3t < n$
upper bound	$(t+1)\Delta$	$(t+1)\Delta$	$O(td)$ (Cor. 4)
lower bound	$\max\{\lfloor(3t-n)/2\rfloor d + \Delta,$ $(\lfloor n/(n-t)\rfloor - 1)\Delta\}$ (Thm. 9 and [4])	$\lfloor(3t-n)/2\rfloor d + \Delta$ (Thm. 9)	$(t+1)d$

Table 5 Worst-case time complexity bounds for n processes with t (unauthenticated) Byzantine failures.

	$n \leq 3t$	$3t < n$
upper bound	impossible [15, 25]	$O(td)$ (Cor. 5)
lower bound	impossible [15, 25]	$(t+1)d$

For *Byzantine failures without authentication*, $n > 3t$ is a necessary condition even in lockstep synchronous systems [15, 25]. To handle such failures, we can apply our transformation to the algorithm in Section 3.2.3 of Dwork, Lynch and Stockmeyer [13] to obtain a consensus algorithm with $O(t \cdot d)$ worst-case time complexity, which is asymptotically optimal. See Table 5.

Recently, the broadcast problem in the presence of authenticated Byzantine failures has captured attention, motivated by state machine replication[3] and blockchains. In a complementary direction to the results in this paper which are for *worst-case* running time, the primary focus is on *good-case* behavior, where "good-case" means the sender is correct, as well as behavior in even more optimistic situations where, in addition to the sender, a large majority or even all the processes are correct. Typically, the worst-case running times of these algorithms, including when the sender is faulty, are either not considered or shown to be $\Omega(t\Delta)$.

Broadcast algorithms with good-case time complexity of $O(d)$ have been proposed by Castro and Liskov (PBFT) [10], Pass and Shi (Hybrid consensus) [23], Yin, Malkhi, Reiter, Gueta and Abraham (HotStuff) [30], and Abraham, Nayak, Ren, and Xiang [4]. All these algorithms assume $n > 3t$. However, there is a fundamental barrier to achieving such good performance if the fraction of faulty processes is larger: Abraham, Malkhi, Nayak, Ren and Yin [2] prove that if $n \leq 3t$, then the good-case running time, i.e., when the sender is correct, must be at least Δ; their proof can be viewed as a quantitative version of the proof of Theorem 4.4 in [13] that consensus is impossible with $n < 3t$ for authenticated Byzantine failures when the upper bound on message delay is either unknown or only holds eventually. Pass and Shi [23] have an analogous result for the permissionless model.

When n is between $2t+1$ and $3t$, several algorithms have been proposed that have $O(d)$ running time under some optimistic conditions, but by necessity they pay a price of at least Δ in the running times in other good-case executions. Pass and Shi's Thunderella [24] has $O(d)$ time when a super-majority of the processes are correct but otherwise has time $O(t\Delta)$. Follow-up work has provided algorithms that have $O(d)$ running time under certain optimistic conditions and $O(\Delta)+O(d)$ running time under other optimistic conditions (e.g., [2–4, 22, 27]). Tradeoff lower bounds on the running times achievable under different optimistic conditions have been proved [22, 27].

[3] State machine replication can be viewed as a collection of repeated instances of broadcast by various senders, where processes must agree on the values sent by the senders as well as an ordering for the broadcasts.

If n is between $t+1$ and $2t$, an algorithm with good-case complexity of $O(\frac{n}{n-t}\Delta)$ has been proposed by Abraham, Nayak, Ren and Xiang [4] based on an algorithm of Wan, Xiao, Shi and Devadas [29]. As discussed in the introduction, there is an asymptotically matching lower bound of $\left(\left\lfloor \frac{n}{n-t} \right\rfloor - 1\right)\Delta$ in [4], which was inspired by a proof in [17].

We are not considering randomized algorithms for asynchronous systems, since their worst-case running times would be infinite, due to the impossibility of solving fault-tolerant consensus and broadcast deterministically in asynchronous systems [16]. It should be noted, however, that they have responsive running times, depending on d and not on Δ, *in expectation*, since they are inherently asynchronous.

3 Preliminaries

3.1 The Bounded-Delay System Model

Fix positive integers n, t, and Δ. We present a model of an algorithm for n processes, t of which may be faulty, that communicate by sending messages over reliable, point-to-point channels with delay at most Δ.

There is a set M of messages. Each *message* is a triple (s, m, r), where s indicates the sending process, m is the message contents, and r indicates the receiving process.

There is a set of n processes, $p_1, p_2, \ldots, p_n$, where each *process* p_i is a state machine with a (possibly infinite) set of states Q_i and two transition functions. The set of states Q_i includes a subset of *initial states*. The two transition functions of p_i are $\delta_i^m : Q_i \rightarrow 2^M$ (which produces the next set of messages to send depending on the current state) and $\delta_i^s : Q_i \times 2^M \rightarrow Q_i$ (which produces the next state depending on the old state and set of messages received).

A *history* of process p_i is an infinite sequence $(M_0^R, q_0, M_0^S, M_1^R, q_1, M_1^S, \ldots)$, where $q_t \in Q_i$, $M_t^R \subseteq M$, and $M_t^S \subseteq M$ for all $t \geq 0$. The triple (M_t^R, q_t, M_t^S) is the *step* of p_i occurring at *time* t, in which p_i receives the set M_t^R of messages, changes its local state to q_t, and sends the set M_t^S of messages. A history is *correct* if q_0 is an initial state of p_i, $q_t = \delta_i^s(q_{t-1}, M_t^R)$ for all $t \geq 1$, and $M_t^S = \delta_i^m(q_t)$ for all $t \geq 1$. In other words, the next state and the messages sent are determined by p_i's transition functions.

An *execution* α is a set of n histories, one for each process, satisfying the following properties.

- At least $n-t$ of the histories are correct; the corresponding processes are the *correct processes* while the rest are *faulty*.
- Every message in an M_t^S (resp., M_t^R) component of p_i's history has i as its sender (resp., recipient). That is, neither the sender nor the channel can lie about the sender, and the channel does not misdeliver messages.
- There exists a bijection from the set of messages appearing in the M_t^S components of the histories to the set of messages appearing in the M_t^R components of the histories such that every message sent is received exactly once and every message received is sent. A message contained in the M_b^R-th component of a process history (the recipient) and contained in the M_a^S-th component of a process history (the sender) is defined to have *delay* $b-a$.
- There exists $d(\alpha) \leq \Delta$ such that the delay of every message is at least 1 and at most $d(\alpha)$; the lower bound of 1 implies that M_0^R is the empty set in every history. When α is understood from context, we denote the bound simply as d.

Since every process takes a step at every nonnegative integer time in its history, in an execution all processes take steps together at every nonnegative integer time.

To model *send omission* failures, we restrict the faulty processes to behave the same as the correct processes, except that in each step, M_t^S can be a subset of $\delta_i^m(q_t)$, reflecting the fact that some prescribed messages are not sent. For *Byzantine* failures, the faulty processes need not follow their transition functions when changing state or sending messages. When an *authentication* mechanism is available, the possible behaviors of the faulty processes are limited: each message sent by a process is signed by the process and no signature can be forged by another process. Thus if process p_i sends a message to process p_j claiming that process p_k sent message m, then p_k did indeed send m. Such a mechanism can be implemented, for instance, using public-key cryptography.

3.2 Definition of Agreement Problems

Every execution of an algorithm solving the *consensus* problem must satisfy the following conditions. Every process p_i starts with an input value from some finite domain of values V. This is modeled by having $|V|$ initial states in the state set Q_i of p_i, one for each possible input $v \in V$.

Termination: Every correct process must eventually make an irrevocable decision on a value from V. We model this by requiring $|V|$ nonintersecting nonempty subsets of Q_i, one for each decision $v \in V$, and requiring the transition function δ_i^s to stay inside each subset. (Processes continue running even after they have decided.)

Agreement: The decision values of all correct processes must be the same.

We also require one of the following validity conditions, which relates the common decision value to the input values.

Strong Unanimity: If the input values of all the processes are the same, say v, then every correct process decides v.

Weak Unanimity: If the input values of all the processes are the same, say v, and there are no failures, then every process decides v.

For both validity conditions, if the inputs are different, then the common decision can be any input value.

In the broadcast problem, a single process p_s is identified as a *sender*. In every execution of an algorithm solving the *broadcast* problem, the sender starts with an input m from some finite domain of values V (modeled as for the consensus problem), and the following conditions must be satisfied:

Termination: Every correct process must eventually make an irrevocable decision on a value from V (modeled as for the consensus problem).

Agreement: All correct processes must decide on the same input m'.

Validity: If the sender p_s is correct then $m' = m$.

As mentioned in the introduction, the broadcast problem can be solved by having the sender send its input to all the processes and then having the processes run a consensus algorithm with each process' input being the value received from the sender. Thus consensus with weak unanimity is equivalent to broadcast and is not subject to the $n > 2t$ requirement for strong unanimity when faulty processes are Byzantine.

We are interested in the *worst-case running time* of agreement algorithms, defined as follows. For execution α, the running time is the smallest time t such that every correct process has decided by time t. The worst-case running time of an algorithm is the maximum, over all executions α, of the running time of α.

4 Upper Bounds on Worst-Case Running Times

In this section, we present algorithms for consensus in the bounded-delay model for send omission, authenticated and unauthenticated Byzantine failures that have asymptotically optimal worst-case running time of $O(fd)$. These consensus algorithms can be converted into algorithms for broadcast with no increase in the (asymptotic) running time using the method mentioned in Section 3. As mentioned before, if Δ is known, then any consensus algorithm A for the lock-step rounds model can be adapted to work in the bounded-delay model by simulating each round of A using Δ time. The resulting algorithm inherits the same fault-tolerance (type of failures and relationship between n and t) and the same unanimity condition as A. However, the running time becomes $T \cdot \Delta$, where T is the round complexity of A after GST, which must be at least $t+1$ (e.g., [12, 20]).

The main result of this section shows how to significantly improve on the running time of $(t+1) \cdot \Delta$ when $n > 3t$, by replacing the dependence on Δ with dependence on the per-execution delay bound d, without relying on any knowledge of Δ. The algorithms result from applying a new transformation to appropriate base algorithms. In Section 4.1, we present the transformation. In Sections 4.2, 4.3, and 4.4, we explain how to use the transformation to achieve asymptotically time-optimal algorithms for send omission failures, authenticated Byzantine failures, and unauthenticated Byzantine failures, respectively.

4.1 Transformation from Basic Round Model to Bounded-Delay Model

We present a generic transformation inspired by an algorithm for crash failures in [14]. The transformation operates on algorithms designed for a more abstract model, called the *basic round model* [13].

▶ **Definition 1** (Basic Round Model). *In the* basic round model*, processes operate in lock-step rounds. In each round, every correct process sends a set of messages to the other processes, then receives a subset of the messages sent at that round destined for it, and then performs some local computation. Starting at some unknown round, called* GST*, every message sent by a correct process to a correct process in a round is received in that round.*

Suppose A is an algorithm for the basic round model that solves consensus for n processes in the presence of up to t process failures, and decides by T rounds after GST. We show how to transform A into algorithm $Tr(A)$ that solves consensus for the same n and t and the same type of process failure in the bounded-delay model. The key feature of the transformed algorithm is that its running time in any execution is $O(T \cdot d)$, where d is the actual maximum message delay in the execution. The transformed algorithm does not rely on knowledge of Δ.

In the transformation, each process partitions its steps into groups so that the g-th group contains $2^g \cdot T$ steps, for $g = 1, 2, 3 \ldots$. Group g simulates T rounds of A using 2^g as an estimate of Δ in this group; thus each simulated round in the group takes 2^g steps.[4] In more detail, group g consists of rounds $g \cdot T + 1$ through $(g+1) \cdot T$. In the first step of round r, which is part of group g, a process sends the messages it is supposed to send in round r of A, tagged with the round number. It then waits 2^g steps and collects all messages received that are tagged with round number r; any other messages are discarded. It then uses the messages received and its current simulated local state to simulate its round r state transition according to A, producing the new local state and set of messages to send at the beginning of the next simulated round. Pseudocode for the transformation is presented in Algorithm 1.

[4] To avoid the corner case when a round is simulated by a single step, we start with $g = 1$ instead of $g = 0$.

Algorithm 1 Transformation of algorithm A for the basic round model that decides by T rounds after GST into algorithm $Tr(A)$ for the bounded-delay model; code for process p_i.

```
    initially:
1:  $g := 1$                                          ▷ group number: $1, 2, 3, \dots$
2:  $r := 1$                                          ▷ round number: $1, 2, 3, \dots$
3:  $s := 1$                      ▷ counts steps in current round: $1, 2, \dots, 2^g, 1, 2, \dots$
4:  $sim\_state$ := initial state of $p_i$ in $A$
5:  $M_R := \emptyset$                  ▷ set of messages received during current round of $A$
6:  $M_S$ := set of messages to send at beginning of round 1 of $A$

7:  while true do                                                   ▷ execute a step
8:      add to $M_R$ all messages received at this step with tag $r$   ▷ part of round $r$ receive
9:      if $s = 1$ then                                         ▷ first step of round $r$
10:         send $M_S$ (each message is tagged with $r$)                 ▷ round $r$ send
11:     else if $s = 2^g$ then                                   ▷ last step of round $r$
12:         $(M_S, state) := \delta_i^A(sim\_state, M_R)$           ▷ round $r$ computation
13:         $sim\_state := state$       ▷ update simulated state; $M_S$ will be sent in next step
14:         if $r = (g + 1) * T$ then                                 ▷ end of group $g$
15:             $g := g + 1$                                         ▷ start next group
16:         $r := r + 1$                                        ▷ start next round of $A$
17:         $M_R := \emptyset$               ▷ clear set of received messages for new round
18:         $s := 1$                                  ▷ reset step counter for new round
19:     else                                    ▷ neither first nor last step of round $r$
20:         $s := s + 1$
```

▶ **Theorem 2.** *Let A be a consensus algorithm for n processes in the basic round model that tolerates up to t process failures and decides by T rounds after GST. Then $Tr(A)$, the result of applying Algorithm 1 to A, is a consensus algorithm with the same unanimity condition for n processes in the bounded delay model that tolerates up to t process failures of the same type and has running time $O(T \cdot d)$ in every execution with maximum message delay d.*

Proof. Consider any execution τ of $Tr(A)$ with upper bound d on the message delays. We can extract from it an execution α of A in the basic round model as follows. For every $r \geq 1$ and every process p_i, the send of round r by p_i in α corresponds to the execution by p_i in τ of Line 10 when p_i's local variable r_i equals r. The receive of round r by p_i in α is distributed over all the executions by p_i in τ of Line 8 when p_i's local variable r_i equals r. The compute of round r in α corresponds to the execution in τ of Line 12 when p_i's local variable r_i equals r. Once $2^g \geq d$, that group (g) of steps will consist of a simulation of T rounds of A after GST in the basic round model, since every message sent in a simulated round that is part of group g (or later) is received in the same simulated round. Since A is correct, α satisfies termination, agreement, and its designated unanimity condition. Thus the same is true of τ and $Tr(A)$ is correct.

We now calculate the time complexity in execution τ of the transformed algorithm. By the assumption that A decides by T rounds after GST in the basic round model and the correspondence between τ and α just presented, correct processes in τ decide by the end of group g, where g is the smallest integer such that $2^g \geq d$. The time until group g ends is at most

$$T \cdot \sum_{g=0}^{\lceil \log d \rceil} 2^g = T \cdot (2^{\lceil \log d \rceil} - 1) = \Theta(T \cdot d). \qquad ◀$$

4.2 Transformed Algorithm for Send Omission Failures

For the base algorithm of our transformation for send omission failures, we start with the algorithm in [13, Section 3.2.1] for consensus in the basic round model. We modify the algorithm to include the optimizations discussed in Remark 1 (p. 302) of [13] that reduce the running time in the basic round model to be $O(t)$ rounds after GST. Call the modified algorithm A_{om}. The algorithm works for n processes, t of which can experience send omission failures, and achieves strong unanimity assuming $n \geq 2t + 1$. For completeness, we present a brief overview of A_{om}, followed by detailed pseudocode[5], and key ideas for correctness (see [13] for detailed correctness proof and worst-case time complexity analysis).

A_{om} is structured as a series of phases, each consisting of four rounds; process $p_{k \bmod n}$ is the *leader* of phase k. Once a correct process is the leader after GST, processes quickly decide. Each process keeps track of the set of values that it has learned are inputs (cf. *Proper* variable); this information is propagated on all messages sent. It also keeps track of the set of values that are candidates for decision (cf. *Locked* variable); each value is associated with the phase number in which it is chosen.

In the first round of a phase, each process sends to the leader a LIST message with all values in its *Proper* set that have no competing value in its *Locked* set. In the second round, the leader chooses any one of the values received in the first round (if any) as its preference, and sends a LOCK message with that value to all the processes. In the third round, if a process receives the LOCK message from the leader, then it updates its *Locked* set and sends an acknowledgment (ACK message) back to the leader. If the leader receives at least $n - t$ acknowledgements, then it decides on its preference. In the fourth round, the leader sends its *Locked* set and its decision (in a LOCK_REL_DEC message) to all the processes. Each recipient removes from its own *Locked* set any value that has a smaller associated phase than any locked value received from the leader. If the recipient has not yet decided, then it adopts the decision in the leader's message.

The pseudocode for A_{om} appears in Algorithm 2; it assumes $n > 2f$. In the pseudocode, *leader*(k) is $p_{k \bmod n}$ and "update (v, k) in *Locked*" means to add (v, k) to the local set variable *Locked* and delete any (v, k') in *Locked* where $k' < k$.

We now sketch the key arguments for correctness. First, it is straightforward to argue that only input values of some (nonfaulty) processes are added to the *Proper* set. Since only values in this set are ever proposed or locked, it follows that if the initial value of every process is v, then no value $v' \neq v$ is ever proposed or locked. This implies *validity.*

The key property that is proved is *lock continuation*, namely, if a process decides on v at the end of phase k, then for every for every phase $k' \geq k$, at least $f + 1$ processes hold a lock (v, k'). The lock continuation property implies that no value $v' \neq v$ is proposed or decided in later phases, which ensures *agreement.*

Finally, it is also straightforward to prove that at most one value v is proposed in the second round of each phase k, and hence, there is a lock only on (v, k). Then, consider what happens at the first phase k after GST such that the leader $p_{k \bmod n}$ is nonfaulty. At phase k, the leader $p_{k \bmod n}$ proposes a single value and since the communication is reliable after GST, it will get acknowledgements for its proposed value and decide. Furthermore, all other processes will receive the decision value in the last round of the phase, and they will decide as well. This implies *termination* within t phases, and hence, $O(t)$ rounds, after GST.

[5] This algorithm, like those used in the next two subsections, is only presented in prose in [13]; we believe we have correctly captured the intent.

Algorithm 2 Consensus in the basic round model, for send omission failures; code for process p_i, $0 \leq i < n$.

```
initially:
$Proper := \{input\}$                ▷ set of values known to be inputs; input is $p_i$'s input
$Locked := \emptyset$                ▷ set of locked values with phase numbers
$decision := \bot$                   ▷ $p_i$'s decision
$M := \emptyset$                     ▷ set of messages received most recently

procedure calc_proper($M$):
add $Proper$ set in $m$ to (local variable) $Proper$ for each $m \in M$

first round of phase $k$ $(k = 1, 2, \ldots)$:                ▷ all-to-leader
$PA := \{v \mid v \in Proper$ and no $(u, *)$ is in $Locked$ with $u \neq v\}$   ▷ proper and acceptable
send $\langle$LIST, $k$, $PA$, $Proper\rangle$ to $leader(k)$
receive set $M$ of messages; call calc_proper($M$)

second round of phase $k$:                                    ▷ leader-to-all
if $p_i = leader(k)$ then                                     ▷ choose value to propose
    $W :=$ union of $PA$ set in $m$ for each $m \in M$        ▷ LIST messages from previous round
    if $|W| > 0$ then
        $pref :=$ arbitrary element of $W$
        send $\langle$LOCK, $k$, $pref$, $Proper\rangle$ to all
receive set $M$ of messages; call calc_proper($M$)

third round of phase $k$:                                     ▷ all-to-leader
if $M = \{m\}$ for some message $m$ then                      ▷ $M$ is either empty or one LOCK message
    update $(v, k)$ in $Locked$ where $v$ is $pref$ in $m$
    send $\langle$ACK, $k$, $Proper\rangle$ to $leader(k)$
receive set $M$ of messages; call calc_proper($M$)            ▷ only $leader(k)$ has $|M| > t$
if $(|M| \geq n - t)$ and $(decision = \bot)$ then            ▷ got $n - t$ ACK's and undecided
    $decision := pref$                                        ▷ $p_i$ decides when it is $leader(k)$

fourth round of phase $k$:                                    ▷ leader-to-all
if $p_i = leader(k)$ then
    send $\langle$LOCK_REL_DEC, $k$, $Locked$, $decision$, $Proper\rangle$ to all
receive set $M$ of messages; call calc_proper($M$)
for each $\langle$LOCK_REL_DEC,$k, L, d, *\rangle$ message in $M$ do
    for each $(w, h') \in L$ do                               ▷ release old locks
        delete all $(v, h)$ in $Locked$ with $(w \neq v)$ and $(h' \geq h)$
    if $(d \neq \bot)$ and $(decision = \bot)$ then
        $decision := d$                                       ▷ $p_i$ decides, might not be $leader(k)$
```

After applying our transformation to A_{om} (Algorithm 2), we obtain an asymptotically time-optimal algorithm for the bounded-delay model:

▶ **Corollary 3.** *There exists a consensus algorithm (with strong unanimity) for n processes that tolerates up to t send omission failures, $n \geq 2t + 1$, and has running time $O(t \cdot d)$ in every execution with maximum message delay d.*

4.3 Transformed Algorithm for Authenticated Byzantine Failures

For the base algorithm of our transformation for authenticated Byzantine failures, we use the algorithm in [13, Section 3.2.2] with modifications to decide within $O(t)$ rounds after GST in the basic round model. Let A_{au} be the modified algorithm. The algorithm works for n processes, t of which can be Byzantine, and uses authentication to achieve strong unanimity, as long as $n \geq 3t+1$. For completeness, we present a brief overview of A_{au}, followed by detailed pseudocode, and key ideas for correctness (see [13] for detailed correctness proof and worst-case time complexity analysis).

A_{au} is structured similarly to A_{om}, with four rounds per phase. We next mention the key differences. (1) Messages are signed before being sent. (2) Keeping track of the input values of correct processes is more involved due to the possibility that faulty process lie, requiring that processes tag every message sent with its input as well as its *Proper* set. If no input occurs at least $t+1$ times once $2t+1$ inputs have been reported, then all values in V are possible decisions and are added to *Proper*; otherwise if a value appears in $t+1$ *Proper* sets reported by other processes, then w is added *Proper*. See procedure calc_proper. (3) The preference chosen by the leader in the second round must have a "proof", meaning that it is contained in the values of at least $n-t$ LIST messages received in the first round. This proof (set of $n-t$ signed messages) is included in the leader's LOCK message. (4) In the third round, the leader decides on its preference if it receives at least $2t+1$ acknowledgements. (5) In the fourth round, locks are sent all-to-all, not just from the leader. (6) In order for a process to decide in the fourth round when it is not the leader, it must receive at least $t+1$ identical decision values from other processes, not just one.

The pseudocode for A_{au} appears in Algorithm 3; it assumes $n > 3f$. Whenever a process receives a signed message, it handles the message only after validating the signature, and similarly for signed messages forwarded inside other messages. For clarity, we omit these checks from the code, and implicitly assume they are carried out. In the pseudocode, *leader*(k) is $p_{k \bmod n}$ and "update $E_i(v,k,proof)$ in *Locked*" means to add $E_i(v,k,proof)$ to the local set variable *Locked* and delete any $E_i(v,k',proof')$ in *Locked* where $k' < k$.

Much of the correctness proof for A_{au} is similar to that for A_{om}. A key difference is that the impossibility of two different values acquiring a valid lock in the same phase is now due to the fact that processes send proofs in their lock messages. Termination is also argued differently and holds because of the more involved management of proper values.

After applying our transformation to A_{au} (Algorithm 3), we obtain an asymptotically time-optimal algorithm:

▶ **Corollary 4.** *There exists a consensus algorithm (with strong unanimity) for n processes that tolerates up to t authenticated Byzantine failures, $n \geq 3t+1$, and has running time $O(t \cdot d)$ in every execution with maximum message delay d.*

4.4 Transformed Algorithm for Unauthenticated Byzantine Failures

For the base algorithm of our transformation for unauthenticated Byzantine failures, we use the algorithm in [13, Section 3.2.3] with modifications to decide within $O(t)$ rounds after GST in the basic round model. Let A_B be the modified algorithm. The algorithm works for n processes, t of which can be Byzantine, and achieves strong unanimity, as long as $n \geq 3t+1$. For completeness, we present a brief overview of A_B, followed by detailed pseudocode, and key ideas for correctness (see [13] for detailed correctness proof and worst-case time complexity analysis).

Algorithm 3 Consensus in the basic round model, for authenticated Byzantine failures; code for process p_i, $0 \leq i < n$.

```
initially:
$Proper := \{input\}$          ▷ set of values known to be inputs; input is $p_i$'s input
$Input\_vals := \emptyset$          ▷ keep track of input values reported
$Other\_proper[] :=$ array of sets of values, one per process; initially all $\emptyset$
                                   ▷ keep track of Proper sets reported
$Locked := \emptyset$          ▷ set of locked values with phase numbers and proofs
$decision := \bot$          ▷ $p_i$'s decision
$Other\_decisions := \emptyset$          ▷ keep track of decisions reported
$M := \emptyset$          ▷ set of messages received most recently

first round of phase $k$ ($k = 1, 2, \ldots$):          ▷ all-to-leader
$PA := \{v \mid v \in Proper$ and no $(u, *, *)$ is in $Locked$ with $u \neq v\}$ ▷ proper and acceptable
send $E_i\langle$LIST, $k$, $PA$, $Proper$, $input\rangle$ to $leader(k)$          ▷ $E_i$ is $p_i$'s authentication function
receive set $M$ of messages; call calc_proper($M$)  ▷ calc_proper definition appears below

second round of phase $k$:          ▷ leader-to-all
if $p_i = leader(k)$ then
    $W := \emptyset$          ▷ choose value to propose
    $Proof[v] := \emptyset$ for each $v \in V$
    for each $m = E_j\langle$LIST, $k$, $S$, $*$, $*\rangle$ message in $M$ do          ▷ from previous round
        for each $v \in S$ do
            add $m$ to $Proof[v]$
    add $v$ to $W$ for each $v \in V$ such that $|Proof[v]| \geq n - t$
    if $|W| > 0$ then
        $pref :=$ arbitrary element of $W$
        send $E_i\langle$LOCK, $k$, $pref$, $Proof[pref]$, $Proper$, $inpu\rangle$ to all
receive set $M$ of messages; call calc_proper($M$)

third round of phase $k$:          ▷ all-to-leader
if $M = \{m\}$ for some message $m$ then          ▷ $M$ is either empty or one LOCK message
    let $v$ be the $pref$ in $m$
    if $|proof$ in $m| \geq n - t$ then
            ▷ at least $n - t$ processes find $v$ proper and acceptable in this phase
        update $E_i(v, k, proof)$ in $Locked$
        send $\langle$ACK, $k$, $Proper$, $input\rangle$ to $leader(k)$
receive set $M$ of messages; call calc_proper($M$)          ▷ only $leader(k)$ has $|M| > t$
if $(|M| \geq 2t + 1)$ and $(decision = \bot)$ then          ▷ got $2t + 1$ ACKs and undecided
    $decision := pref$          ▷ $p_i$ decides when it is $leader(k)$
                                   ▷ continued........................................
```

```
                                              ▷ ............................Continuation of Algorithm 3
fourth round of phase $k$:                                                            ▷ all-to-all
send ⟨LOCK_REL_DEC, $k$, Locked, decision, Proper, input⟩ to all
receive set $M$ of messages; call calc_proper($M$)
for each ⟨LOCK_REL_DEC, $k$, $L$, $d$, $*$, $*$⟩ message in $M$ do
    for each $(w, h', proof)$ in $L$ do                                           ▷ release old locks
        delete all $E_i(v, h, proof)$ in Locked with $(w \neq v)$ and $(h' \geq h)$
    if there is no element in Other_decisions for $j$, where $p_j$ is the sender of $m$ then
        add $(j, d)$ to Other_decisions
    if (there exist $t+1$ elements in Other_decisions with the same value, $v$) and (decision
    $= \perp$) then
        decision := $v$                                          ▷ $p_i$ decides, might not be leader($k$)

procedure calc_proper($M$):
for each $m \in M$ do
    let $p_j$ be the sender of $m$, $v$ be the input value in $m$, and $pr$ be the Proper set in $m$
    if there is no element in Input_vals for $j$ then
        add $(j, v)$ to Input_vals
    Other_proper[$j$] := $pr$              ▷ over-write any previously reported Proper set from $j$
if ($|$Input_vals$| = 2t+1$) and (no value occurs at least $t+1$ times in Input_vals) then
    Proper := $V$                                                    ▷ all possible inputs are valid
else
    for each $w \in V$ do                        ▷ check if $w$ is considered proper by $t+1$ processes
        if $w$ appears in (at least) $t+1$ entries of Other_proper[] then
            add $w$ to Proper
```

To handle Byzantine failures, this algorithm replaces the authentication mechanism of A_{au} by having processes communicate using a reliable broadcast primitive. The high-level structure is similar to A_{om} and A_{au} except that each round of a phase is replaced with two rounds, that implement the reliable broadcast primitive. These pairs of rounds are called *superrounds*. The procedure of releasing locks, which was accomplished during the fourth round in each phase in the previous two algorithms, no longer requires additional communication and thus the lock release is done at the end of the third superround instead of during a fourth superround. As a result, each phase takes three superrounds, which is six rounds. The *reliable broadcast* primitive ensures three properties:

Correctness: If a correct process p broadcasts message m in a superround that starts after GST, then m is delivered from p at every correct process in the same superround.

Unforgeability: If a correct process p does not broadcast message m, then m is never delivered from p at any correct process.

Relay: If message m is delivered from p at any correct process in superround r, then m is delivered from p at every correct process by superround $r+1$ or GST, whichever is later.

Pseudocode for an implementation of the reliable broadcast primitive with a cost of two rounds for broadcast-deliver is given in [13], based on [28]. In the first round, the message to be broadcast is sent to all. In the second round, messages are echoed (sent again to all) and if at least $n-t$ distinct echoes of a message are received, that message is delivered. The complication is that, in order to handle the possible loss of messages before GST, messages continue to be echoed in all later rounds: if a process receives at least $n-2t$ distinct echoes of the same message, then it sends send an echo to all, and once it has received at least $n-t$ distinct echoes for a message it is delivered.

The pseudocode for A_B appears in Algorithm 4; it assumes $n > 3f$. We assume some mechanism for dropping messages that are ill-formed, duplicated, etc. from faulty processes.

After applying our transformation to A_B (Algorithm 4), we obtain an asymptotically time-optimal algorithm:

▶ **Corollary 5.** *There exists a consensus algorithm (with strong unanimity) for n processes that tolerates up to t Byzantine failures, $n \geq 3t + 1$, and has running time $O(t \cdot d)$ in every execution with maximum message delay d.*

5 Lower Bound on Worst-Case Running Time

In this section, we present a lower bound on the worst-case running time of any consensus or broadcast algorithm for n processes that tolerates t Byzantine failures and has access to an authentication mechanism. The lower bound is shown for consensus with weak unanimity, and thus it also holds for consensus with strong unanimity (when $n > 2t$) and for broadcast. Recall that Δ is the global upper bound on message delays across all executions and $d(\alpha)$ is the upper bound on message delays in a specific execution α. Our lower bound assumes that Δ is known, meaning that the algorithm can explicitly use Δ, e.g., to time out waiting for messages. On the other hand d is not known, as it can vary from execution to execution.

Recall that, as mentioned in [6], a simple adaptation of the $(t + 1)$-round lower bound for crash failures in the synchronous model provides a lower bound of $(t + 1)d(\alpha)$ time for every execution α. (Below, we use d instead of $d(\alpha)$, when α is clear from the context.) This lower bound shows that the algorithm of Corollary 4 has asymptotically optimal running time.

Our lower bound states that when $n \leq 3t$, the worst-case running time must be at least $\lfloor (3t - n)/2 \rfloor \cdot d + \Delta$. Table 1 displays our lower bound as well as the lower bound of $\lfloor n/(n - f) - 1 \rfloor \cdot \Delta$ when n is between $t + 2$ and $2t$ [4].

Our result is inspired by one in [14] for crash failures, which showed a lower bound of $(2t - n)d + \Delta$ when $n \leq 2t$. Of course this bound also holds for authenticated Byzantine failures, but by exploiting the worse behavior of the faulty processes, we are able to increase the factor of $(2t - n)$ to $\lfloor (3t - n)/2 \rfloor$ and to increase the range to $n \leq 3t$. The main technical novelties are (1) finding the right partitioning of the processes so that the faulty processes can be substituted for (temporarily) disconnected correct processes, and (2) identifying the desired behavior of the faulty processes, and showing it is possible despite the use of authentication, which requires an involved argument.

The proof considers executions that mimic the behavior of the synchronous rounds model with crash failures. We call such executions "synchronized" and define them next.

Given a positive integer d and a history of process p_i, the subsequence $(M^S_t, M^R_{t+1}, q_{t+1}, \ldots, M^S_{t+d-1}, M^R_{t+d}, q_{t+d})$, where $t = (r - 1)d$, is called *round r of p_i*, for $r \geq 1$. Given an execution, the collection of round r subsequences of all the processes is called *round r* (of the execution). Note that round r starts with the sending of messages at time $(r - 1)d$ and ends with the receiving of messages and subsequent state changes at time rd, but does not including the sending of messages at time rd.

An execution is *synchronized* if

1. every message sent in round r is received in round r, $r \geq 1$, implying that $d(\alpha) \leq d$;
2. the behavior of a faulty process p_i only deviates from that of a correct process by sending a proper subset of the specified messages at some time t, i.e., $M^S_t \subsetneq \delta^m_i(q_t)$, and no messages subsequently; we say the process *fails in round r* if t is in round r; and
3. at most one process fails in each round.

An *s-round synchronized execution prefix* is the result of taking a synchronized execution and truncating each process history after the end of round s.

Algorithm 4 Consensus in the basic round model, for unauthenticated Byzantine failures; code for process p_i, $0 \leq i < n$.

```
initially:
$Proper := \{input\}$    ▷ set of values known to be inputs; input is $p_i$'s input
$Input\_vals := \emptyset$    ▷ keep track of input values reported
$Other\_proper[] :=$ array of sets of values, one per process; initially all $\emptyset$
    ▷ keep track of Proper sets reported
$Locked := \emptyset$    ▷ set of locked values with phase numbers and proofs
$decision := \bot$    ▷ $p_i$'s decision
$M := \emptyset$    ▷ set of messages delivered most recently
$H := \emptyset$    ▷ set of messages delivered so far

procedure check_dec($H$):    ▷ to speed up decision
if $H$ contains at least $t+1$ messages with different senders and the same non-$\bot$ decision value, say $v$) and ($decision = \bot$) then
    $decision := v$

first superround of phase $k$ ($k = 1, 2, \ldots$):
$PA := \{v \mid v \in Proper$ and no $(u, *)$ is in $Locked$ with $u \neq v\}$    ▷ proper and acceptable
broadcast ⟨LIST, $k$, $PA$, $Proper$, $input$, $decision$⟩
deliver set $M$ of messages; call calc_proper($M$)    ▷ same calc_proper as in Alg. 3
add $M$ to $H$; call check_dec($H$)
    ▷ check_dec called frequently on $H$ due to possible lag in delivery of broadcast messages

second superround of phase $k$:
if $p_i = leader(k)$ then
    $W := \{v \in V \mid$ there exist $n - t$ ⟨LIST,$k, S, *, *, *$⟩ messages in $M$ with $v \in S\}$
    if $|W| > 0$ then
        $pref :=$ arbitrary element of $W$
        broadcast ⟨LOCK, $k$, $v$, $Proper$, $input$, $decision$⟩
deliver set $M$ of messages; call calc_proper($M$)
add $M$ to $H$; check_dec($H$)

third superround of phase $k$:
$S := \{v \in V \mid H$ contains a ⟨LOCK$, v, k$⟩ message from $leader(k)$ and $n - t$ ⟨LIST$, k, T, *, *, *$⟩ messages from different senders with $v \in T\}$
for each $v \in S$ do
    update $(v, k)$ in $Locked$
if $|S| > 0$ then
    broadcast ⟨ACK, $k$, $S$, $Proper$, $input$, $decision$⟩ to $leader(k)$
        ▷ other recipients are to ignore this message
deliver set $M$ of messages; call calc_proper($M$)    ▷ only $leader(k)$ has $|M| > t$
add $M$ to $H$; call check_dec($H$)
if (at least $2t+1$ ACK messages for the same value, $v$, are in $M$) and ($decision = \bot$) then
    $decision := v$
delete from $Locked$ every entry with phase smaller than the largest phase number of any entry in $Locked$    ▷ lock release without any communication
```

The classic proofs of the $(t+1)$-round lower bound for consensus in the synchronous model with crash failures (e.g., [12, 20], see [8, Theorem 5.7]) construct a sequence of executions in which adjacent executions are *indistinguishable* to at least $n-1$ of the processes (meaning that each of the processes has the same history in the two executions), starting with an execution that decides 0 and ending with an execution that decides 1. Essentially the same construction can be applied to synchronized executions, resulting in the following:

▶ **Lemma 6.** *Consider any consensus algorithm that ensures weak unanimity for n processes that tolerates t Byzantine failures (possibly using authentication) where $n > t+1$. Let $s \le t$. There exists a chain $\alpha_1, \alpha_2, \ldots, \alpha_m$ of s-round synchronized execution prefixes, for some m, such that:*

(a) *No process fails in α_1 (resp., α_m) and all the input values are 0 (resp., 1).*

(b) *For all i, $1 \le i < m$, the number of processes that fail in either α_i or α_{i+1} (or both) is at most s.*

(c) *For all i, $1 \le i < m$, there exists a process q such that α_i and α_{i+1} are indistinguishable to all processes except possibly q and one of the following holds:*

  (1) *the same processes are faulty in α_i and α_{i+1}, or*

  (2) *the only difference between the faulty processes in α_i and α_{i+1} is that q fails in α_{i+1} but not in α_i, or*

  (3) *the only difference between the faulty processes in α_i and α_{i+1} is that q fails in α_i but not in α_{i+1}.*

▶ **Lemma 7.** *Assume $n = 3t - 2s$, for some $s < t$. Any consensus algorithm for n processes that tolerates t Byzantine failures (possibly using authentication) and ensures weak unanimity has an execution in which every message has delay at most d but at least one correct process does not decide before time $s \cdot d + \Delta$.*

Proof. Let A be any consensus algorithm for n processes and t authenticated Byzantine failures. Let α be an s-round synchronized execution prefix of A. A *partition for α* is a partition (X, Y, F_1, F_2) of the processes such that $|X| = |Y| = |F_2| = t - s$, $|F_1| = s$, and F_1 contains all the processes that fail in α; since $t > s$, X and Y are nonempty. Such a partition is possible since $3(t-s)+s = 3t-2s = n$. Note that $|F_1|+|F_2| = |F_1|+|X| = |F_1|+|Y| = t$.

The *X-extension* of α is the execution that extends α in which

- processes in $X \cup F_2$ remain correct,
- every process in $Y \cup F_1$ (that has not already failed) fails at the beginning of the extension (giving $t - s + s = t$ failures),
- failed processes behave by sending no messages, and
- every message sent after the end of α has delay d.

By the correctness of A, the correct processes must eventually decide in the extension; denote the decision value by $dec_X(\alpha)$.

Define the *Y-extension* of α and $dec_Y(\alpha)$ analogously, by swapping the roles of X and Y.

▷ Claim 8. For every s-round synchronized execution prefix α and every partition (X, Y, F_1, F_2) for α, $dec_X(\alpha) = dec_Y(\alpha)$.

Proof. Suppose in contradiction there exist α and (X, Y, F_1, F_2) such that $dec_X(\alpha) \neq dec_Y(\alpha)$. Let α_X (resp., α_Y) be the X-extension (resp., Y-extension) of α. By construction all message delays in α_X and α_Y are at most d. Let t_{dec} be the latest time at which some process in X decides in α_X or some process in Y decides in α_Y.

Assume in contradiction to the claim of the lemma that $t_{dec} < s \cdot d + \Delta$, so the decision is before Δ time has elapsed in the extension after the end of α. Now consider the extension β of α in which

- any process in F_1 that hasn't failed in α fails just after α, by sending no more messages;
- all processes in F_2 fail at the end of α by behaving in a "two-faced" manner until time $s \cdot d + \Delta$: toward X as they did in α_X and toward Y as they did in α_Y; after time $s \cdot d + \Delta$, processes in F_2 send no messages;
- all processes in $X \cup Y$ remain correct,
- every message sent after α within X or within Y has delay d, and
- every message sent after α between X and Y has delay Δ.

The next subclaim, which is the heart of the argument, shows that this behavior is allowed even if an authentication scheme is available.
Subclaim: For all t, $(s-1)d \leq t < (s-1)d + \Delta$,

1. Every process p_i in X (resp., Y) sends the same set of messages in step t of β as it does in step t of α_X (resp., α_Y).
2. Every process p_i in F_2 sends the same set of messages to processes in X (resp., Y) in step t of β as it does in step t of α_X (resp., α_Y).
3. Every process p_i in X (resp., Y) receives the same set of messages in step $t+1$ of β as it does in step $t+1$ of α_X (resp., α_Y).
4. Every process p_i in F_2 receives the same set of messages from processes in X (resp., Y) as it does in step $t+1$ of α_Y (resp., α_Y).
5. Every process p_i in X (resp., Y) has the same state in step $t+1$ of β as it does in step $t+1$ of α_X (resp., α_Y).

Proof of Subclaim: By induction on t.

Suppose $t = (s-1)d$. Consider $p_i \in X \cup Y \cup F_2$. The set of messages that p_i sends in step t of β is determined by p_i's state at the end of α, which is the immediately preceding prefix of β, α_X, and α_Y. Thus Properties (1) and (2) hold.

Property (3) holds since the set of messages S to a process in X (resp., Y) that are in transit just before step $t+1$ of β is a superset of that S' in α_X (resp., α_Y), thanks to Properties (1) and (2). The potential messages in S that are not in S' are those from Y to X (resp., X to Y), since all processes in Y (resp., X) are faulty in α_X (resp., α_Y) and send no more messages. However, the same set of messages are delivered because of how the message delays are specified and the fact that $t+1 \leq (s-1)d + \Delta$.

Property (4) holds since the same set of messages to a process in F_2 from a process in X (resp., Y) are in transit just before step $t+1$ of β as in α_X (resp., α_Y), thanks to Properties (1) and (2), and since the delays are specified to be the same.

Property (5) holds since a process in X (resp., Y) receives the same set of messages in step $t+1$ of β as in α_X (resp., α_Y) by Property (3) and its state in step t of β is the same as in step t of α_X (resp., α_Y) by the fact that both executions have α as a prefix, which ends with the state at time t.

Suppose $t \geq (s-1)d + 1$. Essentially the same argument as for $t = (s-1)d$ holds, with the following differences. Properties (1) and (5) rely on the inductive hypothesis for Property (5) (states being the same) instead of the fact that the executions have the same prefix. Finally, the crucial part of the proof is showing Property (2). This relies on the fact that, by the inductive hypothesis, each process in F_2 has received the same set of messages so far in β from processes in $X \cup Y$ as it does in α_X and in α_Y, so the authentication mechanism cannot prevent the faulty processes in β from sending the same set of messages to X (resp., Y) as in α_X (resp., α_Y).
End of proof of Subclaim.

The total number of failures in β is $|F_1| + |F_2| = s + (t - s) = t$. Up to time t_{dec}, processes in X cannot distinguish β and α_X and decide $dec_X(\alpha)$, while processes in Y cannot distinguish β and α_Y and decide $dec_Y(\alpha)$, contradicting agreement. ◁

Thanks to Claim 8, we can simply refer to the common value of $dec_X(\alpha)$ and $dec_Y(\alpha)$ as the "decision of the partition for an execution" when the partition is clear from the context.

Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be the chain of s-round synchronized execution prefixes from Lemma 6. Since all the inputs in α_1 are 0 and there are no failures, the weak unanimity condition implies that every partition for α_1 has decision 0. Similarly, every partition for α_m has decision 1. So there must be some i, $1 \leq i < m$, such that all partitions for α_i have decision 0 while at least one partition for α_{i+1} has decision 1, call it (X, Y, F_1, F_2).

Let q be a process such that α_i and α_{i+1} are indistinguishable to all processes except possibly q (q exists by Lemma 6). We consider the three cases of how α_i, α_{i+1} and q are related from Lemma 6, and show a contradiction for each one, implying that t_{dec} must be at least $s \cdot d + \Delta$.

Case 1: The same processes are faulty in α_i and in α_{i+1}. If q is not in X, then α_i and α_{i+1} are indistinguishable to all processes in X. Since the partition (X, Y, F_1, F_2) has decision 1 for α_{i+1}, it also has decision 1 for α_i, which is a contradiction. If q is in X, then it is not in Y and the analogous argument holds.

Case 2: The only difference between the faulty processes in α_i and α_{i+1} is that q fails in α_{i+1} but not in α_i. Then q is in F_1 so α_i and α_{i+1} are indistinguishable to all processes in $X \cup Y$. So (X, Y, F_1, F_2) has decision 1 for α_i, a contradiction.

Case 3: The only difference between the faulty processes in α_i and α_{i+1} is that q fails in α_i but not in α_{i+1}.

If q is in $F_1 \cup F_2$, then the same argument as in Case 2 holds.

Suppose q is not in $F_1 \cup F_2$; without loss of generality, q is in Y. There must be some process p such that p is in F_1 but not faulty in α_i. The reason is that $|F_1| = s$, at most s processes are faulty in α_i (by part (b) of Lemma 6), and there is a faulty process in α_i not in F_1 (namely, q).

Thus q is in Y but not in F_1, p is in F_1 but not in Y, p is not faulty in α_i, and α_i and α_{i+1} are indistinguishable to all processes in X. Thus $(X, (Y - \{q\}) \cup \{p\}, (F_1 - \{p\}) \cup \{q\}, F_2)$ is a partition for α_i with decision 1, which is a contradiction. ◀

Note that Lemma 7 applies for any $n \leq 3t$. When $n = 3t$, then $s = 0$ and the bound is Δ; when $n = 3t - 2$, then $s = 1$ and the bound is $d + \Delta$; when $n = 2t$ (and t is even), then $s = t/2$ and the bound is $(t/2)d + \Delta$; finally, when $n = t + 2$, then $s = t - 1$ and the bound is $(t - 1)d + \Delta$. In general, we have:

▶ **Theorem 9.** *Assume $n \leq 3t$. Any consensus algorithm for n processes that tolerates t Byzantine failures (possibly using authentication) and ensures weak unanimity has an execution in which every message has delay at most d but at least one correct process does not decide before time $\lfloor (3t - n)/2 \rfloor d + \Delta$.*

6 Discussion

This paper studies whether the cost of timeout can be avoided when solving agreement problems in the bounded-delay model. On the positive side, we prove that the consensus problem can be solved with asymptotically optimal time of $O(t \cdot d)$, in the presence of t failures, for send omission, authenticated and unauthenticated Byzantine failures. For send omission failures, our algorithm requires n to be greater than $2t$, while the Byzantine algorithms require n to be greater than $3t$. On the negative side, we show that dependence on Δ cannot be avoided if $n \leq 3t$ for authenticated Byzantine failures. Specifically, when $n \leq 3t$, the time complexity for consensus with weak unanimity is at least $\lfloor 3t - n)/2 \rfloor \cdot d + \Delta$. The results show that the cost of timeouts can be avoided for these agreement problems if and only if the resilience is the same as that needed for solving consensus in eventually-synchronous systems.

An immediate open question is to find the optimal time complexity when $n \leq 3t$ for authenticated Byzantine failures. In this direction, note that if $3t - n$ faulty processes are taken out, then the remaining faulty processes constitute less than a third of the remaining system. This might indicate a path to finding an upper bound that matches our lower bound. There are analogous open questions for the case of crash and (send) omission failures when $n \leq 2t$. Interestingly, the crash lower bound when $n \leq 2t$ has a $2t - n$ term, which corresponds to the number of failed processes that should be taken out in order to have a majority of correct processes.

A challenging research direction is to study the time complexity of consensus algorithms in the *eventually-synchronous* model [13], where the upper bound of Δ on message delay only holds after GST. Numerous consensus algorithms have been proposed for this model, e.g., [10, 30], but to the best of our knowledge, their time complexity is in $\Omega(t \cdot \Delta)$ *after GST*.

References

1 Ittai Abraham. State machine replication for two servers and one omission failure is impossible even in a lock-step model. `https://decentralizedthoughts.github.io/2019-11-02-primary-backup-for-2-servers-and-omission-failures-is-impossible/`, 2019.

2 Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020. `doi:10.1109/SP40000.2020.00044`.

3 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Brief announcement: Byzantine agreement, broadcast and state machine replication with optimal good-case latency. In *34th International Symposium on Distributed Computing (DISC)*, 2020. `arXiv:2003.13155`.

4 Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of Byzantine broadcast: A complete categorization. In *2021 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 331–341, 2021. `doi:10.1145/3465084.3467899`.

5 Hagit Attiya, Fatemeh Borran, Martin Hutle, Zarko Milosevic, and André Schiper. Structured derivation of semi-synchronous algorithms. In *International Symposium on Distributed Computing (DISC)*, pages 374–388. Springer, 2011. `doi:10.1007/978-3-642-24100-0_37`.

6 Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41(1):122–152, 1994. `doi:10.1145/174644.174649`.

7 Hagit Attiya, Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 359–369. ACM, 1991. `doi:10.1145/103418.103457`.

8 Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics.* McGraw-Hill Publishing Company, 1st edition, 1998.

9 A.A. Bharali and P. Berman. Distributed consensus with general omission failures and timing uncertainty. In *Proceedings of Phoenix Conference on Computers and Communications*, pages 168–174, 1993. `doi:10.1109/PCCC.1993.344468`.

10 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002. `doi:10.1145/571637.571640`.

11 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Byzantine consensus is $\Theta(n^2)$: The Dolev-Reischuk bound is tight even in partial synchrony! In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 14:1–14:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.DISC.2022.14.

12 Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. doi:10.1137/0212045.

13 Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

14 Cynthia Dwork and Larry Stockmeyer. Bounds on the time to reach agreement as a function of message delay. Technical Report RJ 8181 (75140), IBM Thomas J. Watson Research Division, jun 1991. available in https://drive.google.com/file/d/1zoCe05kzhOWBRi2DQ7OmFpZJxBPMpzSS/view?usp=sharing.

15 Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986. doi:10.1007/BF01843568.

16 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

17 Juan A Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 658–668, 2007. doi:10.1109/FOCS.2007.61.

18 Amir Herzberg and Shay Kutten. Fast isolation of arbitrary forwarding faults. In Piotr Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 339–353. ACM, 1989. doi:10.1145/72981.73006.

19 Amir Herzberg and Shay Kutten. Early detection of message forwarding faults. *SIAM Journal on Computing*, 30(4):1169–1196, 2000. doi:10.1137/S0097539796312745.

20 Michael Merritt. Notes on the Dolev-Strong lower bound for Byzantine Agreement. unpublished manuscript, 1985.

21 Dimitris Michailidis. Fast set agreement in the presence of timing uncertainty. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC, '99Atlanta, Georgia, USA, May 3-6, 1999*, pages 249–256. ACM, 1999. doi:10.1145/301308.301365.

22 Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. Hybrid-BFT: Optimistically responsive synchronous consensus with optimal latency or resilience. Cryptology ePrint Archive, Paper 2020/406, 2020. URL: https://eprint.iacr.org/2020/406.

23 Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31 International Symposium on Distributed Computing (DISC)*, 2017. doi:10.4230/LIPICS.DISC.2017.39.

24 Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology (EUROCRYPT): 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2018. doi:10.1007/978-3-319-78375-8_1.

25 Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.

26 Stephen Ponzio. The real-time cost of timing uncertainty: Consensus and failure detection. Master's thesis, MIT, 1991.

27 Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 839–857, 2020. doi:10.1145/3372297.3417284.

28 TK Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987. doi:10.1007/BF01667080.

29 Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round Byzantine broadcast under dishonest majority. In *18th International Conference on Theory of Cryptography (TCC)*, pages 381–411, 2020. doi:10.1007/978-3-030-64375-1_14.

30 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019. doi:10.1145/3293611.3331591.