Byzantine Agreement with Median Validity

David Stolz[1] and Roger Wattenhofer[1]

1 **ETH Zürich**
Switzerland
stolzda@ethz.ch, wattenhofer@ethz.ch

Abstract

We introduce a stronger validity property for the byzantine agreement problem with orderable initial values: The *median validity property*. In particular, the decision value is required to be close to the median of the initial values of the non-byzantine nodes. The proximity to the median scales with the desired level of fault-tolerance: If no fault-tolerance is required, algorithms have to decide for the true median. If the number of failures is maximal, algorithms must still decide on a value within the range of the input values of the non-byzantine nodes. We present a deterministic algorithm satisfying this property for $n \geq 3t + 1$ within $t + 1$ phases, where t is the maximum number of byzantine nodes and n is the total number of nodes.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Reliability, fault-tolerance, median, consensus

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

A distributed system consists of a number of nodes, which collaboratively achieve fault-tolerance: Even if some of the nodes fail, the system as a whole can continue to operate reliably. As such, distributed systems are omnipresent in areas that have strong availability and reliability requirements. The challenging problem is to keep the system in a consistent and reasonable state. The following example illustrates this problem.

An airplane control system consists of multiple machines. In order to reach a decision on what action the plane performs next, the system relies on data from various sensors. One type of sensor is the *altitude meter*. Assume that the system needs to decide whether or not it is safe to descend, and queries the altitude meters. As those sensors are susceptible to freezing – and known to malfunction when frozen – each plane is equipped with multiple altitude meters. The airplane control system needs to agree on a value, and in order to avoid a single point of failure, this agreement will be achieved in a distributed manner.

One may argue that this agreement operation is at the core of many distributed systems, and generally fundamental for distributed computing research. Depending on the failure model, it is sometimes referred to as *consensus* (to cope with crash failures, e.g. [10]) or as *byzantine agreement* (to cope with arbitrary failures, e.g. [24]). In our airplane scenario – as well as in many other scenarios – one must be able to tolerate arbitrary failures, as a frozen sensor might not simply crash but still continue to provide data, even though that data is totally bogus. Nodes not complying with the protocol or reporting wrong data are called byzantine, nodes which behave correctly we simply call good.

The byzantine agreement (BA) problem consists of three properties: Termination, agreement, and validity. We need termination since we require our algorithm to come to a decision within finite time, as not deciding how high a plane is flying for a long time is disastrous. We need agreement as we want the nodes of our distributed system to continue their operation

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the same state. To rule out trivial solutions ("The altitude is always 10,000 meters.") we need a third property: Validity. The purpose of the validity property is to make sure that the value agreed on is reasonable. There exist slightly different definitions for validity, but the most common one is the following (see, e.g. [4, 12]):

Validity If all good nodes start with the same input value, they agree on this value.

Unfortunately, there is no limitation on what the decision value can be for configurations in which not all good nodes start with the same input value. Surprisingly, in the literature, dissenting configurations receive only little attention. Informally speaking, one is obliged to accept a solution as long as the nodes agree on *any value*. The rationale for this weak validity condition is as follows: Since initially there is disagreement among the good nodes, a byzantine node can propose a bad value but follow the protocol of the algorithm, and there is no way to guarantee that the decision value will always be the input value of a good node. While this argument is formally correct, it implicitly assumes that it is impossible to distinguish good from bad values.

We believe that there is room for improvement, in particular in cases where input values can be ordered. Assume that we use an existing algorithm for BA in the previously described airplane example, and that we have four altitude meters which report the altitudes 995, 1002, 1004, and 5000, respectively. Even though it is clear that the correct altitude is roughly 1000, existing algorithms for BA might choose 5000, which might lead to a catastrophe. Often one must solve BA on input values in $\mathbb{R}$, e.g., measurements. Measured values are usually prone to noise, hence even the good nodes will have slightly different input values. The classic validity property (which is only concerned with the case where all input values are equal) does not help. To address this problem, we introduce a stronger validity property: The *median validity property*. In particular, we require the decision value to be "reasonably close" to the median of the values of the good nodes. We will later specify this requirement precisely.

In Section 4 we present a new BA algorithm called the Jack algorithm. Our algorithm has an optimal resilience, i.e., it works as long as $n \geq 3t + 1$, where n is the total number of nodes, and t denotes the *tolerance*, i.e., the maximum number of byzantine nodes. Note that we distinguish between t and f, where $f \leq t$ is the actual number of byzantine nodes during an execution of the algorithm, analogous to [8]. This distinction is necessary, as the nodes only know the tunable parameter t, since f is only known to the adversary. We will prove that the algorithm works for every $f \leq t$, thus the algorithm does not rely on any particular f. The Jack algorithm is deterministic and terminates in $t + 1$ phases, where each phase consists of a constant number of synchronous rounds. The algorithm additionally guarantees that the decision value will be reasonably close to the median of the good nodes. The proximity of the decision to the median of the good nodes is defined in an elastic way: In the best case, if we do not need any fault-tolerance, the algorithm agrees on the median itself. In the worst case, if the number of byzantine nodes is maximal, we require the value only to be within the interval spanned by the good nodes; in our airplane example, the algorithm will choose a value x with $995 \leq x \leq 1004$.

In addition, we present a lower bound for our validity property. If all byzantine nodes collude and suggest arbitrarily small (respectively large) input values, no algorithm can guarantee to find the correct median of the good nodes. In Section 5, we will provide exact bounds on this error.

2 Related Work

The BA problem has a long history, starting in 1980 when it was first introduced by Lamport et al. [24]. Since then, a vast amount of research has been conducted and many models and questions have been studied. The earlier works focused mainly on general computability questions, e.g., [2, 10]. The first algorithms required message sizes exponential in n, or had a runtime that was far from optimal, but these shortcomings have been addressed and solved later.

With the proliferation of data centers, companies started implementing a large number of BA protocols, and the interest from the practical distributed systems community exploded. Well-known protocols include e.g. PBFT [6], Farsite [1], Google's Chubby [5], PeerReview [14], Zyzzyva [17], AZyzzyva [13], Apache ZooKeeper [15], SMART [20], ZZ [27], or RAFT [22]. These protocols try to improve various practical aspects of BA, in particular runtime, but do not address validity. We believe that some of these protocols could benefit from our orthogonal goal of strengthening the validity property.

Our contribution is a relative of the Queen [3] and King [4] algorithms by Berman, Garay (and Perry). These algorithms operate in the synchronous model and terminate in $t + 1$ phases, which is asymptotically optimum for algorithms that terminate *simultaneously* [8]. While the Queen algorithm does not have optimal resilience, the King algorithm tolerates the maximum number of byzantine nodes. Our Jack algorithm achieves optimum runtime and resilience as well, and additionally satisfies a stronger validity condition as discussed in the introduction.

There are also algorithms that are closely related to the BA problem, but do not satisfy all three BA properties. In *distributed average consensus*, nodes try to achieve consensus on the average of all input (sensor) values. There are some major differences to BA: First, algorithms do not tolerate byzantine failures, but, e.g., only link failures [23, 29], systematic failures on the input (noise) [28], or both [16]. Second, the goal is not to agree on a specific value after a finite amount of time, but rather to *converge* towards a value. Hence, these algorithms do not satisfy the termination property, and cannot be used in many scenarios, e.g. in our airplane example. Please refer to the book of Ren and Beard [25] for an extensive survey.

The *clock synchronization problem* [18] is another agreement problem, which usually assumes that the clock values are initially close to each other, however, not exactly equal. While this problem was only loosely related to the BA with the classical validity property, the relation to BA with median validity is evident, since an agreement on the median is a reasonable goal for clock synchronization.

In a very recent work [26] Su and Vaidya study the problem of byzantine distributed optimization of real valued convex cost functions. Instead of starting with an initial value, every node starts with a cost function $h(x)$. The goal is to reach consensus on a value x, such that the total cost (the sum of the costs of the good nodes) is minimized. Since the optimization problem is similar to the problem we study, it is not surprising that they establish a proof that the optimization problem cannot be solved *exactly* in the presence of byzantine nodes, which conveys a similar message as our lower bound (see Section 5).

The work which is most closely related to our contribution is the one which aims at strengthening the validity property. We are aware of two different approaches. The first one is the *approximate agreement problem* by Dolev et al. [7]. Dolev et al. strengthen the validity property at the expense of the agreement property. The agreement property by Dolev et al. does not require the nodes to agree consistently on one value, but allows the nodes to choose different values which are in ε-distance of each other. In their algorithm, the range of acceptable values is iteratively reduced, until the range satisfies the ε-property. They can guarantee that the algorithm terminates as early as possible. On the other hand, the agreement property cannot be guaranteed anymore. We believe that many applications depend on all three BA properties, as future actions depend on past decision values.

The second one is the *strong consensus problem* [11, 21]. The strong consensus problem allows input values from a finite Domain $\mathcal{D}$, and strengthens the validity property by requiring the nodes to agree on a value that was the initial value of at least one good node. It might seem that such a validity property is also the right choice for real valued inputs; however, Neiger [21] showed that n must be at least $|\mathcal{D}| \cdot t$, which is clearly not applicable to scenarios which require initial values in $\mathbb{R}$.

Our validity property is hence both in the spirit of Dolev et al. [7] and Neiger [21], and we think that we found a reasonable trade-off between agreement, validity of the decision and the required number of nodes.

3 Model

Let A denote an array of length l. Throughout this paper we will use zero-based arrays, meaning that the first element of an array is $A[0]$ and the last element is $A[l-1]$. When we refer to the *median* of an array A, we want the median to be a value which actually *occurs* in that array. As this raises a problem with arrays of even size, we define the median to be the element at index $A[\lceil \frac{l}{2} \rceil - 1]$, assuming A is sorted.

The network consists of n nodes which are fully connected, i.e., every node can communicate with every other node. Nodes communicate by exchanging messages, and we assume that time elapses in *synchronous rounds*. We allow every node to send exactly one message to every other node in each round, and we assume that these messages will be received in the following round. The network itself is assumed to be reliable, meaning that messages are not altered or delayed between sender and receiver. The different links between nodes are isolated from each other, i.e., only the sender and receiver of a message know about the message. Hence, if a node *broadcasts* a message, it sends a single message to every node, including itself.

The byzantine agreement problem tends to be specified in different ways in different papers: Sometimes the input values are only binary, sometimes there is a predetermined "leader node". In this paper, we define the BA problem as follows:

Every node has an input value $x \in \mathbb{R}$. There are $f \leq t$ byzantine nodes which can behave *arbitrarily*. This includes lying about their input value, not following the protocol, sending different messages to different nodes in the same round, crashing, and so on. We allow an *adaptive* adversary, which can determine during the execution of the protocol which nodes are byzantine, and how they behave. Note however, that the adversary is limited to selecting at most t nodes, meaning that a node that is byzantine cannot be declared as non-byzantine at later point in time. We call a node *good* if it is not selected by the adversary to be byzantine at any point throughout the execution; it follows directly that there are always at least $n - t$ good nodes. However, a good node does not a priori know *which* of the other nodes are good; a good node only knows that there must be at least $n - t$ good nodes in the system. At the end of the algorithm, every good node must have decided for a certain value, which we will refer to as the *decision*.

Any algorithm that solves byzantine agreement with median validity must satisfy the following three properties:

Agreement For every selection of input values and every selection of byzantine nodes, all good nodes must decide on the same value.
Termination Every good node must decide on a value in finite time.
Median Validity The decision is always a *valid value*[1] (see definition below).

Let us now define which values are valid. For that purpose, we denote by G the *sorted* array of the input values of the good nodes. Note that the length of the array is $n-f \geq n-t$, and the median of the good nodes is $G[\lceil\frac{n-f}{2}\rceil - 1]$.

▶ **Definition 1** (valid value)**.** We call a value x *valid*, if it holds that

$$G[\left\lceil\frac{n-f}{2}\right\rceil - 1 - t] \leq x \leq G[\left\lceil\frac{n-f}{2}\right\rceil - 1 + t].$$

Note that in the best case, where $f = t = 0$, the median of the good nodes is the only valid value. In the worst case, where $n = 3t+1$ and $f = t$ (which corresponds to the maximum number of byzantine nodes so that BA is solvable, see [19, 24]), every value which is between the smallest input value of a good node and the largest input value of a good node is valid.

Observe that even though the decision must be between the smallest and largest input value of a good node, the decision is *not* necessarily the input value of a good node! For example, we allow that the value proposed by a byzantine node can be chosen, as long as it satisfies the required proximity to the median. Even though this relaxation has only little negative impact on the quality of the decision, it is a crucial enabler of the validity property: As shown by [11], a validity property which also satisfies that the decision is the input value of a good node is impossible for real values, and even with input values from a finite domain, the failure tolerance decreases linearly with the size of the domain, which is an unfeasible requirement for many applications.

4 The Jack Algorithm

We developed an algorithm that uses some ideas from the *Phase King algorithm* developed by Berman, Garay and Perry [4]. One such idea is to have $t+1$ nodes predetermined as *kings*. In $t+1$ phases there is always one of those kings the depicted king for that phase, assuming a special role. We use the same idea in our algorithm, but refer to this special role as the *jack*.

Note that the Jack algorithm requires the tolerance as an input parameter `t`, which is tunable. The choice of `t` affects the runtime of the algorithm, its fault-tolerance, and how close to the median of the good nodes the final decision value will be. In particular, `t` determines the maximum number of byzantine nodes that the algorithm can tolerate, and it determines which values are considered valid, according to the definition of a valid value (see Definition 1).

All nodes start the execution of the algorithm at the same time, and execute the same steps at the same time; the only exception is the step marked with **Only Jack**, in which only the respective jack of that phase is active, and all other nodes remain silent. Note that since it is predetermined which node is the jack of which phase, every node can simply ignore byzantine nodes pretending to be the jack in a wrong phase.

[1] Note that the classic validity property is implied by our validity property as a special case.

Algorithm 1: The Jack algorithm

```
input  : inputValue, n, t
output : consensusValue
Setup Stage
1  Broadcast(inputValue)
2  Store and sort all received values in array values
   interval = values.Subarray($\lceil \frac{n-t}{2} \rceil - 1$, $n - \lfloor \frac{n-t}{2} \rfloor - 1$)
   Broadcast("bounds: interval.first, interval.last")
3  Store all received interval bounds
   if inputValue is in at least n-t bounds then
       suggestion = inputValue
   else
       suggestion = pick any value from interval that is in at least n-t bounds
   current = suggestion
Search Stage
   for $i$ = 1 to t + 1 do
4      Broadcast(current)
5      if some value x appears $\geq$ n-t times then
           Broadcast("propose x")
6      if some "propose x" received > t times then
           current = x
Only Jack
       if some "propose x" received > t times then
           jackSuggestion = current
       else
           jackSuggestion = suggestion
       Broadcast("suggest jackSuggestion")
7      if current == jackSuggestion or
       (interval.first $\leq$ jackSuggestion $\leq$ interval.last) then
           Broadcast("support jackSuggestion")
8      if received "propose x" < n-t times then
           if some "support jackSuggestion" received > t times then
               current = jackSuggestion
9  consensusValue = current
```

The Jack algorithm works as follows: At the start there is a *setup stage* consisting of 3 rounds, in which all nodes determine which values they will support later on. It is followed by the *search stage* consisting of t+1 phases, in which the nodes agree on the decision value. In each of these phases there is at first a part where all nodes perform the same actions, and a second part, in which a predetermined jack tries to suggest a value. As there are t+1 jacks, we are guaranteed to have at least one non-byzantine jack. Since we do not know in which of these phases we have a non-byzantine jack, we need the first part: This part prevents a byzantine jack to change an existing agreement. The second part (the part of the jack) allows a good jack to suggest a valid value, so that we are assured to reach agreement.

In the following sections, we prove the correctness of the algorithm.

4.1 Preliminaries

▶ **Lemma 2.** *The median of the good nodes* $G[\lceil \frac{n-f}{2} \rceil - 1]$ *is in the* interval *of every good node.*

Proof. The intervals are created during the setup stage in Step 2 by truncating the array of received input values on both sides, i.e., by removing a subarray with the smallest values, and by removing a subarray with the largest values. In order to show that the median of the good nodes will always remain in the subarray interval, we show that it cannot be in either of two parts of the array that are removed.

Let us first look at the part of the array which is removed since the values within this part are considered too small. The smallest index of the sorted array that is kept in the array interval is $\lceil \frac{n-t}{2} \rceil - 1$. Since we use arrays that start at index 0, the number of values that are smaller or equal to the value at a certain index of the array is equal to the index. Thus, we know that the construction of the subarray interval discards the smallest $\lceil \frac{n-t}{2} \rceil - 1$ values. As there are $\lceil \frac{n-f}{2} \rceil - 1$ input values of good nodes which are smaller than the median of the good nodes (by definition of the median), and since $\lceil \frac{n-f}{2} \rceil - 1 \geq \lceil \frac{n-t}{2} \rceil - 1$, it follows that the median of the good nodes will never be removed because it is considered too small.

Analogously we show that the median of the good nodes cannot lie in the part of the array that is removed due to the values being too large. The largest index of the received value that is kept is $n - \lfloor \frac{n-t}{2} \rfloor - 1$. Therefore, the number of received values that are discarded since they are considered too large is $\lfloor \frac{n-t}{2} \rfloor$. Again, due to definition of the median, this is at most the number of input values of good nodes which are larger or equal to the median of the good nodes. Thus, the median of the good nodes will also never be considered to be too large, which concludes the proof. ◀

▶ **Lemma 3.** *All values stored in the* interval *arrays of good nodes are valid.*

Proof. Note that valid values are around the median of the good nodes. Thus, for a value to be not valid, it must either be too small, or too large. We prove the lemma by proving the two cases individually.

The smallest value of the received input values that is taken into the interval is at the index $\lceil \frac{n-t}{2} \rceil - 1$ of the sorted array of all received values, thus the number of input values that are discarded since they are too small is $\lceil \frac{n-t}{2} \rceil - 1$. Note that $f \leq \lceil \frac{n-t}{2} \rceil - 1$, since:

$$\begin{aligned}
3t &< n && \mid -t, \cdot\frac{1}{2} \\
t &< \frac{n-t}{2} \\
t &\leq \left\lceil \frac{n-t}{2} \right\rceil - 1 && \mid f \leq t \\
f &\leq \left\lceil \frac{n-t}{2} \right\rceil - 1
\end{aligned}$$

Within those values that are discarded, there are at most f values from byzantine nodes. Therefore, the number of good values that are discarded is at least $\lceil \frac{n-t}{2} \rceil - 1 - f$, meaning that the largest input value of a good node that is discarded because it is too small is at least $G[\lceil \frac{n-t}{2} \rceil - 1 - f - 1]$.

We can distinguish two cases: The first case is that all the byzantine nodes have sent a value that is discarded since it was too small. In this case, the smallest value in the interval must be from a good node, and since $\lceil\frac{n-t}{2}\rceil - 1 - f$ good values have been discarded, the smallest value of the interval is $G[\lceil\frac{n-t}{2}\rceil - 1 - f]$, which is at least as large as the lower bound of being valid, since:

$$\begin{aligned} f &\le t \\ \left\lceil\frac{n-f}{2}\right\rceil + f &\le \left\lceil\frac{n-t}{2}\right\rceil + t & \quad | -f-t-1 \\ \left\lceil\frac{n-f}{2}\right\rceil - 1 - t &\le \left\lceil\frac{n-t}{2}\right\rceil - 1 - f \end{aligned}$$

And therefore $G[\lceil\frac{n-f}{2}\rceil - 1 - t] \le G[\lceil\frac{n-t}{2}\rceil - 1 - f]$.

The second case is where at least one byzantine node sent a value that is not discarded for being too small. In that case, the number of good values that are discarded for being too small is at least one more than in the previous case. In particular, the value $G[\lceil\frac{n-t}{2}\rceil - 1 - f]$ will be discarded. Since all nodes that are not discarded must be bigger than the discarded ones, the lower bound holds.

We show that no value in the interval can be too large to be valid analogously. The number of values that are not taken into the interval since they are too large is $\lfloor\frac{n-t}{2}\rfloor$. As there might be at most f values from byzantine nodes, there will be at least $\lfloor\frac{n-t}{2}\rfloor - f$ values from good nodes discarded because they are too large. Hence, the smallest discarded good value is at most $G[(n-f) - (\lfloor\frac{n-t}{2}\rfloor - f)] = G[\lceil\frac{n-t}{2}\rceil + t]$. We distinguish the two cases again: First, the case where all byzantine values are discarded, as they are too large. In that case, the largest value kept in the interval is $G[\lceil\frac{n-t}{2}\rceil + t - 1]$, and since $\lceil\frac{n-t}{2}\rceil \le \lceil\frac{n-f}{2}\rceil$, it follows that $G[\lceil\frac{n-t}{2}\rceil + t - 1] \le G[\lceil\frac{n-f}{2}\rceil + t - 1]$, i.e., the required bound is satisfied. Second, the case where there is at least one byzantine value that is not discarded. In that case, the value $G[\lceil\frac{n-t}{2}\rceil + t - 1]$ is discarded, implying that all values in the interval are smaller than this value, and thus the upper bound holds. ◀

4.2 Main Theorem

▶ **Theorem 4.** *The* Jack algorithm *achieves byzantine agreement with median validity in* $O(t+1)$ *rounds, if and only if* $n \ge 3t+1$.

Note that the Jack algorithm matches the upper bound on the number of byzantine nodes that it can tolerate [19], and that it has an asymptotically optimal runtime [8,9]. To facilitate the readability of the proof, we split it up into separate lemmas. Note that throughout the proof, we often use the fact that $n-f \ge n-t$ implicitly.

▶ **Lemma 5.** *Each good node has at least one value in its* interval *which is within the bounds of at least* $n-f$ *received bounds.*

Proof. We proved that the median of the good nodes is in the interval of every good node (Lemma 2). Hence the value of the median of the good nodes is at least in all good bounds, i.e., every good node receives at least $n-f$ bounds containing the median of the good nodes. Since every good node has the median in its own interval, the lemma holds. ◀

This lemma guarantees that the setup stage can always be completed successfully, i.e., that every good node can always find an eligible suggestion.

▶ **Lemma 6.** *If all good nodes have the same value stored in the variable* current *at the beginning of a phase, they will not change the value of* current *during this phase.*

Proof. Let v denote the value which is stored in the variable current at all good nodes. In Step 4 all $n-f$ good nodes broadcast v, and thus all good nodes hear at least $n-f$ messages containing v. Therefore, all good nodes propose v in Step 5, and all good nodes hear at least $n-f$ proposals for v in Step 6. They update their value to v, i.e., they keep v. The only other step where the nodes update their variable current is Step 8, but since every good node received at least $n-f$ proposals for v, no good node updates to the jackSuggestion. ◀

▶ **Corollary 7.** *If all good nodes have the same value v stored in the variable* current *in the beginning of any phase, they will agree on v when the algorithm terminates.*

▶ **Lemma 8.** *If all good nodes start with the same input value v, they will agree on this value. (The classic validity property holds.)*

Proof. Since all good nodes start with the input value v, this value is also the median of the good nodes. Lemma 2 states that v must be in the interval of every good node. Thus, every good node sets its variable current to v in Step 3. As every good node starts the search stage with current set to v, it follows from Corollary 7 that they agree on v when the algorithm terminates. ◀

▶ **Lemma 9.** *A good node will only store values in its variable* current *which are within the bounds of an* interval *of at least one good node.*

Proof. Note that every good node picks a value from its own interval as the starting value for current, thus the lemma holds at the end of the setup stage. For the search stage, it is sufficient to show, that if the lemma holds at the beginning of a phase, it will also hold at the end of the phase (and thus at the beginning of the next phase). There are two steps in which a good node can update its variable current: Step 6 and Step 8.

If a good node updates the variable current in Step 6 to a new value x, we know that more than t nodes sent a proposal for x, i.e., at least one good node sent a *"propose x"*. Hence, the good node which sent the proposal must have received x in Step 5 least $n-t \geq 2t+1 > f$ times, meaning that at least one good node must have sent x in Step 4. Since we assumed that the lemma holds at the beginning of the phase, this value must have been in the interval of a good node.

The second possibility to update current is in Step 8. If a good node updates the value in Step 8 to a jackSuggestion x, it must have received a message *"support x"* from at least one good node. Such a message is sent in Step 7, but a good node only supports a suggestion from the jack if the suggestion is either the current of the good node, or if the suggestion is within the bounds of its interval. In the latter case the lemma holds, as this is condition enforces exactly what the lemma states. Looking at the former case, we know that the lemma holds after Step 6, and thus it also holds in Step 7, as current was not changed in between. Therefore, x must be within the interval of at least one good node, concluding the lemma. ◀

Combining this lemma with Lemma 3 yields the following corollary:

▶ **Corollary 10.** *A good node always has a valid value stored in its variable* current*.*

Since the value for which a good node decides when the algorithm terminates is the value stored in the variable current, it follows that every good node will always decide for a valid value. Before we can establish the last lemma that concludes the proof of Theorem 4, we need to show the following lemma. Please note that this lemma has already been shown in slightly different form in [4], since the authors use the same idea in the Phase King algorithm.

▶ **Lemma 11.** *If a good node updates its variable* current *in Step* 6 *to x, no good node updates its variable* current *to y in Step* 6 *in the same phase, if $x \neq y$.*

Proof. Assume for a contradiction that there are two good nodes, one updating its value to x, and one updating its value to y. Therefore, one node must have heard at least $t + 1$ proposals for x, and another node heard at least $t + 1$ proposals for y. Thus, at least one good node proposed x, and one good node proposed y. In order for a good node to propose a certain value in Step 5, it must have heard this value at least $n - t$ times. Since at most t of those values can come from byzantine nodes, it must have heard the value from at least $n - 2t$ good nodes. Therefore at least $n - 2t$ good nodes sent x in Step 4, and at least $n - 2t$ good nodes sent y; with $x \neq y$, the total number of good nodes must be at least $2n - 4t$. Since the number of good nodes is by definition $n - t$, it follows that $n - t \geq 2n - 4t$, i.e., $n \leq 3t$ must hold, which contradicts our initial assumption that $n \geq 3t + 1$. ◀

With the help of this lemma, we only need to show the following lemma to complete the proof of Theorem 4.

▶ **Lemma 12.** *At the end of a phase with a non-byzantine jack, all good nodes store the same value in their variable* current*.*

Proof. If all good nodes store the same value in current at the beginning of the phase, this lemma follows directly from Lemma 6. Hence we only need to prove the case where not all good nodes start with the same value in current.

Note that the jack has two values which it can propose: Either the value stored in the suggestion created in the setup stage, or the value stored in its variable current. Both these values are valid: suggestion is chosen from the interval in the setup stage, and thus valid (see Lemma 3), and the value stored in current is always valid (see Corollary 10).

Let us first look at the case where the jack suggests its value stored in its variable current. As the jack suggests its current, it must have received more than t proposals for a certain value x, and therefore it has updated its current to x in Step 6. Since the jack heard more than t proposals, at least one good node proposed x. This node must have heard at x in Step 5 at least $n - t$ times, i.e., it heard it at least from $n - 2t$ good nodes. As we assume that $n \geq 3t + 1$, it follows that at least $t + 1$ good nodes started this phase with the value x. As a consequence of Lemma 11, those nodes will not change the *value* in their variable current in Step 6, as the only value for which they can hear enough proposals is in fact x. This implies that the number of good nodes which have value x stored in current in Step 7 must be at least $t + 1$, and because the jack suggests x, there will be at least $t + 1$ good nodes that support x. Any good node that heard less than $n - t$ proposal messages for x will therefore accept the suggestion of the jack, which is x. Every node that ignores the suggestion from the jack does so, because it heard at least $n - t$ proposal messages for the same value x in Step 6. It follows from Lemma 11 that these nodes updated their current to the same value x as the jack did in Step 6. Therefore, all nodes have value x stored in current at the end of the phase.

It remains to be shown that if the jack suggests its value stored in suggestion, that all good nodes update their variable accordingly. If the jack suggests its suggestion, it heard at most t proposals for any value in Step 6. Hence, any other good node can hear at most $2t$ proposals for any value. Recall that the jack picked a value for its suggestion in the setup stage that was in at least $n - t$ bounds (note that there is always such a value, see Lemma

5). As at most t of those bounds might have been sent by byzantine nodes, the chosen value is in within at least $n - 2t \geq t + 1$ bounds of good nodes, where we used that $n \geq 3t + 1$. Therefore, at least those $t + 1$ good nodes support the suggested value in Step 7. Since every good node heard at most $2t < n - t$ proposals, and the suggestion of the jack has at least $t + 1$ support, every good node accepts the suggested value of the jack in Step 8. ◀

Since we perform $t + 1$ phases, we are guaranteed to have at least one phase with a non-byzantine jack. In combination with Corollary 7, this establishes the agreement property, and with Corollary 10 follows the median validity property. As the termination property follows from the construction of the algorithm, Theorem 4 follows. ◀

5 Discussion

In the following, we put our result into context, by providing a tight bound on how close to the median of the good nodes any algorithm can get, and by doing so, we also discuss an alternative approach to the initially stated problem of achieving agreement on a reasonable value.

▶ **Theorem 13.** *There is no deterministic algorithm that can guarantee that, for every selection of input values and every selection of byzantine nodes, the decision value x will always satisfy*

$$G[\left\lceil \frac{n-f}{2} \right\rceil - 1 - \left\lfloor \frac{t}{2} \right\rfloor] < x < G[\left\lceil \frac{n-f}{2} \right\rceil - 1 + \left\lceil \frac{t}{2} \right\rceil].$$

Proof. To prove this theorem, it is not necessary to assume byzantine *behavior* throughout the execution of the protocol; it suffices to let the byzantine nodes select tedious input values. We assume that $f = t$. Let us look at two different selections of input values: First, we let the start the good nodes with input values $G_1 = \{1, \dots, n - t\}$, and the byzantine nodes pick the input values $\{n - t + 1, \dots, n\}$. Second, we start the good nodes with input values $G_2 = \{t + 1, \dots, n\}$, and the byzantine nodes with $\{1, \dots, t\}$. Throughout the execution of the protocol, the byzantine nodes adhere to the protocol defined by the algorithm. As the two executions will be exactly equal, the good nodes agree on the same value v in both runs.

We will now show by contradiction that no value satisfies the bounds for both scenarios. Note that for all values which are in both sets of good input values G_1, G_2 it holds by construction that $G_1[i] = G_2[i - t]$. Assume that v is a value that satisfies the upper bound for G_1. Note that:

$$v < G_1[\left\lceil \frac{n-f}{2} \right\rceil - 1 + \left\lceil \frac{t}{2} \right\rceil] \quad \Longrightarrow \quad v \leq G_1[\left\lceil \frac{n-f}{2} \right\rceil - 1 + \left\lceil \frac{t}{2} \right\rceil - 1]$$

Due to the selection of the input values of the good nodes. Therefore it must hold that:

$$v \leq G_1[\left\lceil \frac{n-f}{2} \right\rceil - 1 + \left\lceil \frac{t}{2} \right\rceil - 1] = G_2[\left\lceil \frac{n-f}{2} \right\rceil - 2 - \left\lfloor \frac{t}{2} \right\rfloor] < G_2[\left\lceil \frac{n-f}{2} \right\rceil - 1 - \left\lfloor \frac{t}{2} \right\rfloor]$$

Hence, every value that satisfies the upper bound for the first scenario, does not satisfy the lower bound for the second scenario. ◀

▶ **Theorem 14.** *There is a deterministic algorithm that guarantees that the decision value x always satisfies*

$$G[\left\lceil \frac{n-f}{2} \right\rceil - 2 - \left\lfloor \frac{t}{2} \right\rfloor] \leq x \leq G[\left\lceil \frac{n-f}{2} \right\rceil + \left\lceil \frac{t}{2} \right\rceil].$$

Proof. The algorithm works in two stages: First, we use any algorithm that achieves interactive consistency (for example the one from [24]). With interactive consistency, every node agrees on the input values of *all* nodes; in particular, it knows all the input values of the good nodes, and for all the byzantine nodes, there is either agreement on a value, or the good nodes agree that the node is byzantine. The second stage of the algorithm is to pick the median of the agreement array.

The agreement array will consist of all values from G, and some byzantine values. Note that every node that is identified as a byzantine node has no effect on the resulting decision value; therefore, the distortion created by the byzantine nodes is maximal if all byzantine nodes succeed in adding a bad value to the agreement array. Observe that this distortion can be maximized, if all byzantine nodes add values that are either smaller than the median of the good nodes, or larger[2]. It is clear that in order to achieve the maximal effect on the value of the resulting decision, and not only to maximize the index shift on the median of the good nodes, all byzantine values must be either smaller than the smallest good value, or larger than the largest good value, and that $f = t$. As $n \geq 3t + 1$, it is not possible for the byzantine nodes to move the median of the agreement array beyond the value range spanned by the good values; thus, the minimal (maximal) decision value can be bounded by a value from G. Due to the definition of the median, the t byzantine nodes can shift the decision value at most by $\lceil \frac{t}{2} \rceil$ index positions. Hence, the upper bound follows directly, and the lower bound follows with the fact that $\lceil \frac{t}{2} \rceil \leq \lfloor \frac{t}{2} \rfloor + 1$. ◀

Comparing how close to the median of the good nodes the decision value of the Jack algorithm is in terms of index position difference within G yields the following corollary:

▶ **Corollary 15.** *The Jack algorithm is a 2-approximation with respect to the index distance in G in the worst case.*

Having shown that the Jack algorithm is not optimal, one might argue, that it is better to use an algorithm based on interactive consistency, as outlined before. However, we are not aware of any algorithm that achieves interactive consistency with real valued[3] input values in an *efficient* way. The algorithm in [24] uses messages of exponential size, and also local computation that requires exponential runtime in each step. Since the Jack algorithm only requires small messages (only one[4] value), and only simple local computations, we claim that our algorithm achieves a reasonable trade-off between precision and complexity.

6 Conclusion

We introduced the median validity property which addresses a shortcoming in the specification of the byzantine agreement problem for many practical scenarios. Algorithms satisfying the

[2] If one byzantine node proposes a value that is smaller than the median, and one proposes a value that is larger, the median does not change.
[3] Note that many of the existing algorithms assume binary input values.
[4] In the setup stage two values are sent in one round, which could of course be split into two rounds in which only one value is sent.

median validity property can be used in environments with orderable input values, and such algorithms are particularly suitable if one cannot expect the input values of the nodes to be exactly equal. We presented an algorithm that achieves byzantine agreement satisfying this stronger validity property within an asymptotically optimal runtime of $t + 1$ phases, requiring only small messages. To analyze the quality of the validity property, we established a lower bound on the index distance to the good median within the input values of the good nodes, and we showed that our algorithm achieves a 2-approximation; i.e., if the input values are tightly clustered around a certain value v, the Jack algorithm guarantees to decide for a value that is very close to v.

References

1 Atul Adya, William J Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R Douceur, Jon Howell, Jacob R Lorch, Marvin Theimer, and Roger P Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14, 2002.

2 Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30. ACM, 1983.

3 Piotr Berman and Juan A Garay. *Asymptotically optimal distributed consensus.* Springer, 1989.

4 Piotr Berman, Juan A Garay, and Kenneth J Perry. Towards optimal distributed consensus. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 410–415. IEEE, 1989.

5 Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 335–350. USENIX Association, 2006.

6 Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

7 Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

8 Danny Dolev, Ruediger Reischuk, and H Raymond Strong. Early stopping in byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720–741, 1990.

9 Michael J Fischer and Nancy A Lynch. A lower bound for the time to assure interactive consistency. *Information processing letters*, 14(4):183–186, 1982.

10 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

11 Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220. ACM, 2003.

12 Oded Goldreich and Erez Petrank. The best of both worlds: Guaranteeing termination in fast randomized byzantine agreement protocols. *Information Processing Letters*, 36(1):45–49, 1990.

13 Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European conference on Computer systems*, pages 363–376. ACM, 2010.

14 Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. The case for byzantine fault detection. In *HotDep*, 2006.

15 Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX Annual Technical Conference*, volume 8, page 9, 2010.

16 Soummya Kar and José MF Moura. Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise. *Signal Processing, IEEE Transactions on*, 57(1):355–369, 2009.

17 Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 45–58. ACM, 2007.

18 Leslie Lamport and P Michael Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM (JACM)*, 32(1):52–78, 1985.

19 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

20 Jacob R Lorch, Atul Adya, William J Bolosky, Ronnie Chaiken, John R Douceur, and Jon Howell. The smart way to migrate replicated stateful services. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 103–115. ACM, 2006.

21 Gil Neiger. Distributed consensus revisited. *Information Processing Letters*, 49(4):195–201, 1994.

22 Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proc. USENIX Annual Technical Conference*, pages 305–320, 2014.

23 Stacy Patterson, Bassam Bamieh, and Amr El Abbadi. Distributed average consensus with stochastic communication failures. In *Decision and Control, 2007 46th IEEE Conference on*, pages 4215–4220. IEEE, 2007.

24 Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

25 Wei Ren and Randal W Beard. *Distributed consensus in multi-vehicle cooperative control*. Springer, 2008.

26 Lili Su and Nitin Vaidya. Byzantine multi-agent optimization: Part I. *arXiv preprint arXiv:1506.04681*, 2015.

27 Timothy Wood, Rahul Singh, Arun Venkataramani, Prashant Shenoy, and Emmanuel Cecchet. Zz and the art of practical bft execution. In *Proceedings of the sixth conference on Computer systems*, pages 123–138. ACM, 2011.

28 Lin Xiao, Stephen Boyd, and Seung-Jean Kim. Distributed average consensus with least-mean-square deviation. *Journal of Parallel and Distributed Computing*, 67(1):33–46, 2007.

29 Lin Xiao, Stephen Boyd, and Sanjay Lall. A scheme for robust distributed sensor fusion based on average consensus. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 63–70. IEEE, 2005.