Byzantine Agreement with Interval Validity

Darya Melnyk
Distributed Computing Group
ETH Zurich
Zurich, Switzerland
dmelnyk@ethz.ch

Roger Wattenhofer
Distributed Computing Group
ETH Zurich
Zurich, Switzerland
wattenhofer@ethz.ch

***Abstract*—To solve Byzantine agreement, n nodes with real input values, among which $t < n/3$ are Byzantine, have to agree on a common consensus value. Previous research has mainly focused on determining a consensus value equal to an input value of some arbitrary node. In this work we instead assume that the values of the nodes are ordered and introduce a novel validity condition which accepts consensus values that are close to the k^{th} smallest value of the correct nodes. We propose a deterministic algorithm that approximates the k^{th} smallest value and show that this approximation is the best possible for the synchronous message passing model. Our approach is furthermore extended to multiple dimensions, where the order is not well-defined, and we show that our algorithm can be applied to determine a value that lies within a box around all correct input vectors.**

***Index Terms*—distributed consensus, multi-valued, vector agreement, synchronous systems**

I. Introduction

Some of the key results in distributed computing promise to provide a distributed system that can tolerate *arbitrary ("Byzantine") failures*. The machines (nodes) of a distributed system regularly check if they are in the same state. Whenever the nodes disagree, they run a *Byzantine agreement* protocol to eliminate blunders by some nodes and thus enforce agreement. If nodes continually agree on their state, the distributed system as a whole is correct.

For Byzantine agreement to be applicable, it should be fast and ensure that all nodes agree on a common state. These criteria are known as *termination* and *agreement property*, respectively. However, the two properties are not enough. For instance, a protocol may just agree to delete all the information in the system, fulfilling both termination and agreement, but potentially also destroying valuable data. To prevent such absurd "solutions", we need a third property, known as the *validity property*. Informally, the validity property must make sure that the decision "makes sense". In particular, if all nodes of a distributed system propose the same state, they should settle for that state.

There are several situations where nodes of a distributed system do not propose the same state. For example, all nodes of a distributed stock market system may have seen a different transaction first, and therefore propose their own transaction as the next one to be included in the common ledger. Other situations could occur when all nodes of a distributed auction system offer a slightly different price, or the nodes of a distributed flight control system are equipped with a height sensor and all sensors report slightly different altitudes, e.g., represented as floating point numbers. With Byzantine nodes participating in the decision, it would be not advisable to simply agree on any value.

A majority decision will also not help in systems where each node might propose a different outcome. Luckily, many distributed systems seem to have in common that the inputs of the Byzantine agreement algorithm can be ordered: some transactions have an earlier timestamp than others and altitude sensors will likely report at least slightly different heights. If the inputs are ordered, we can try to decide on a value which is not an outlier, as outliers coming from Byzantine nodes should be avoided. Instead we want to decide on a value which makes sense: If we want our plane to operate safely, we can think of avoiding outliers by choosing the median value. If we are interested in the price of honest bidders for our apartment, we are looking for a high bid, but not a goofy outlier.

The median is in some sense the safest value in a Byzantine setting, as it is robust against Byzantine attacks from both sides, i.e., it does not matter whether the Byzantine nodes propose high or low values. Our paper presents the first algorithm that finds the optimal median in the Byzantine setting. The ability to choose the largest or smallest value in Byzantine environments also finds various applications. The need of a generalization to the k^{th} largest or smallest value is less obvious, but it is interesting in several cases as well. As an example, consider a distributed system with at most t Byzantine (arbitrarily malicious) nodes. In addition to these Byzantine nodes, it is assumed that there are nodes that are not Byzantine but also not correct. These nodes will generally follow the protocol, but they will not be completely honest about their input, e.g. agents who always bid a too high value. We do not want our result to be affected by these nodes. By going for the k^{th} smallest or largest value instead of the maximal/minimal/median value, we can adapt nicely to such situations. In this paper we will assume that the implementation is aware of such malfunctioning behavior of the system and chooses k accordingly before starting the algorithm.

Paper Overview: Given totally orderable inputs, we are looking for an output close to the k^{th} largest or smallest value, and depending on the number of Byzantine nodes we can tolerate a solution in an *interval* around the k^{th} value. In Section III, we formally define this validity condition. In Section V, we present an algorithm that will achieve this *interval validity*. Our

algorithm is optimal in that it tolerates the maximum possible number of $t < n/3$ Byzantine nodes. Our algorithm is also optimal in how close the chosen value is to the k^{th} value thanks to a matching lower bound. In Section VII we finally show that our algorithm also can handle multi-dimensional inputs.

II. Related Work

The problem of Byzantine agreement was first introduced as the "Byzantine Generals Problem" by Pease, Shostak and Lamport [12], [16]. They showed that it is not possible to reach agreement if at least one third of all nodes are Byzantine. Byzantine agreement has since then been studied in a range of different settings, synchronously and asynchronously in shared memory and message passing models. For the synchronous message passing model, Fisher and Lynch [9] proved that any deterministic algorithm can reach consensus in a minimum of $t+1$ rounds. Berman et al. later proposed the Phase Queen [2] and the Phase King Algorithms [3] that both match this lower bound for binary input values. While agreement is possible in the synchronous message passing model, no *deterministic* protocol can establish agreement in the presence of even one Byzantine node in the asynchronous model. This was shown by Fisher, Lynch and Paterson [10].

Byzantine agreement is well studied in the binary setting where each node has the input value 0 or 1. Many applications must however be able to handle real numbers $\mathbb{R}$ or natural numbers $\mathbb{N}$. This setting is referred to as *multivalued* Byzantine agreement [21] in the literature. The idea is to establish agreement on a value that was an input value of some correct node. Some proposed algorithms assume that the majority of nodes have the same input value [11], [20], [21]. If there is no clear majority, some leader node might decide on a value that all nodes will adapt, or the nodes choose a preselected value. These algorithms need $t+1$ rounds to establish agreement, which is optimal in the synchronous model. The algorithms will however agree on an arbitrary value if there is no majority among the input values.

Dolev et al. [7] proposed a deterministic algorithm for *approximate* Byzantine agreement in the asynchronous message passing model. In this relaxed setting, nodes do not establish consensus on an exact value, since this is impossible [10]. They instead converge towards a consensus value in every round. In the proposed algorithm the values of all nodes converge towards the mean of the correct values. Fekete [8] later improved the running time of the algorithm and proposed an algorithm for approximation that solves consensus in the synchronous model exactly, if it iterates for $t+1$ rounds. Approximate agreement can guarantee that the values of correct nodes will be inside an arbitrarily small interval after sufficiently many iterations but cannot solve the exact Byzantine agreement problem unless the lower bounds for exact consensus are satisfied. Another relaxation of the Byzantine agreement problem is k-set agreement, where the nodes try to agree on values that are within some common set of size at most k [5], [13], [17].

In this paper we present a protocol which establishes agreement on a value that is an approximation to the k^{th} smallest of all correct input values. Similar approaches have been considered for the special case of the median: Doerr et al. [6] consider the Power of Two Choices to establish agreement on the value which is close to the median in the asynchronous message passing model. In their protocol, each node requests the values of two nodes chosen uniformly at random among all nodes and updates its value to the median of the two requested values and its own. The authors showed that for $t \in O(\sqrt{n})$ the system stabilizes with a consensus value that is between the $(n/2 - c\sqrt{n \log n})$-largest and the $(n/2 + c\sqrt{n \log n})$-largest value. The same problem was considered for the synchronous message passing model by Stolz et al. [19]. In this paper the authors proposed an algorithm which computes an approximation of the median within $t+1$ rounds. Their approximation is not optimal compared to the bounds of any deterministic algorithm for this model. We will use these ideas to derive an approximation to the k^{th} smallest value and show that our method can be adjusted to solve the median problem optimally.

In the spirit of [14], [15], [22] we will present how our method can be applied to Byzantine vector consensus. This is a generalization of multivalued agreement that allows multi-dimensional input values. Previous work mostly concentrated on finding a value which is within the convex hull of all correct values. While this method is efficient for approximate agreement in the asynchronous message passing model, the exact agreement in the synchronous message passing model requires exponential number of computations to determine the convex hull in the presence of Byzantine nodes. Xiang and Vaidya [23], [24] introduced two relaxations of the convex hull - the k-relaxed and the (δ, p)-relaxed Byzantine vector consensus. The former requires the consensus value to be inside the projections of the convex hull onto any k dimensions of the vectors and the latter requires the value to be within distance δ to the convex hull. They show that their relaxation cannot be used to improve the number of Byzantine nodes that can be tolerated by the system. We will relax the validity condition from the convex hull to a box and apply our proposed k^{th} smallest value algorithm. This adjustment allows us to achieve exact consensus in $O(d(t+1))$ rounds, where d represents the dimension. It can furthermore tolerate up to $\lceil n/3 \rceil - 1$ Byzantine nodes.

III. Model and Notation

In this work we consider Byzantine agreement for the synchronous message passing model in a distributed system with n nodes, where every node can directly communicate with every other node. In the beginning of the computation each node has an input value from a totally orderable domain, for example $\mathbb{R}$. The goal is to make all nodes agree on a common value that is close to the k^{th} smallest value of all correct input values by communicating in synchronous rounds. In each such round, a node sends one message to all other nodes, receives all messages sent by the other nodes and finally performs some local computation on the received input. We differentiate between two kinds of nodes. The correct nodes follow the

protocol at all times. If such a correct node sent a message to all other nodes, this message will be received by all nodes in the same round. In addition, we allow our system to contain at most t nodes that are Byzantine, where $t < n/3$. Byzantine nodes can behave arbitrarily, they can choose to send different messages to different nodes or not to send any message at all. The Byzantine nodes are assumed to be controlled by an omnipotent adversary. The adversary has knowledge about all message contents that the correct nodes send in the same round and is allowed to decide which messages the Byzantine nodes will send in this round upon accessing that information.

The Byzantine agreement protocol must generally satisfy the following standard conditions:

- (Agreement) All correct nodes agree on the same value upon termination.
- (Termination) Every correct node terminates with a valid value after a finite number of communication rounds.
- (All-Same Validity) If all correct nodes start with the same value, they eventually agree on that value.
- (Correct-Input Validity) The nodes agree on the value that at least one of the correct nodes has proposed.

Note that the Correct-Input validity is equivalent to the All-Same validity if the input values of nodes are binary. However, in the multi-valued setting where all nodes have different input values the Correct-Input validity cannot be satisfied. This is because any correct node is not differentiable from a Byzantine node which follows the protocol using its own input value. We therefore introduce a relaxed validity condition of the Correct-Input validity:

- (Any-Input Validity) The nodes agree on the value that at least one of the nodes has proposed. This value is not required to be proposed by a correct node.

In this paper, the goal of the protocol is to make all correct nodes agree on a particular value which is close to the k^{th} smallest value among the input values of all correct nodes. It is not necessarily a drawback if the Byzantine nodes propose values close to the k^{th} smallest value, while values that are far away should be omitted. In order to handle values that are too far away from the k^{th} smallest value we introduce a new validity condition called *Interval Validity*. Let S be a sorted array containing the $n - t$ correct input values and refer to $S[k]$ as the k^{th} smallest value in this array. In each round it is assumed that every node stores the values of all received messages in a sorted array R of size $n - t + f$. Note that $R[k]$ is not the same value as $S[k]$, since R also stores f Byzantine values. The validity condition for the k^{th} smallest value is defined as follows

Definition 1 (Interval Validity)**.** *Sort all input entries of correct nodes in an array S. A valid value is a value v that is close to the k^{th} smallest value of all correct nodes:*

$$S\big[k - \lceil t/2 \rceil\big] \leq v \leq S\big[k + \lfloor t/2 \rfloor\big]$$

This validity condition does not guarantee that v is an input value of a correct node. We only require it to not be further away than $\lceil t/2 \rceil$ positions from the actual k^{th} value in S. For $t = 0$ the validity condition holds only for $S[k] = v$, which is the exact k^{th} smallest value. Note that this is not the same definition of Interval Validity as used in [1].

For brevity we denote the subarray of S which starts with the a^{th} value and ends with the b^{th} $S[a, b]$. In contrast, $\big[S[a], S[b]\big]$ denotes the interval enclosed by the two values, i.e., correct and Byzantine values which lie inside the boundaries. The same notation is used for R, which also will be referred to as the local array of a node.

For the computation of the k^{th} smallest value we use the geometric median. It is defined as the central value of an array of ordered numbers. For an even number of nodes, this value usually corresponds to the mean of the two central values. We adjust the definition of the median to agree on the smaller of the two central values in the case where a node receives an even number of values:

Definition 2 (Median)**.** *Given an array A of n values, the median is defined as $A\big[\lfloor n/2 \rfloor\big]$, i.e., the value at position $\lfloor n/2 \rfloor$ in the array A.*

This alternative definition enables the nodes to choose a valid median value.

IV. Lower Bound for the Approximation of the k^{th} Smallest Value

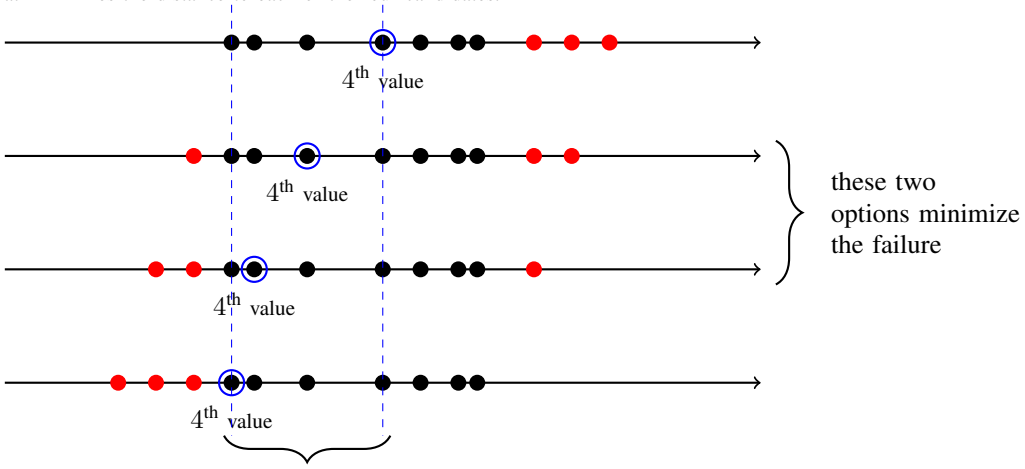
In this section we show that no deterministic algorithm can approximate the k^{th} smallest value better than by $\lceil t/2 \rceil$ positions in the presence of t Byzantine nodes. The quality of the approximation is calculated by the number of positions by which the approximate k^{th} smallest value is shifted from the actual k^{th} smallest value with respect to the array S. Two cases will be considered separately, the case where $k \in \big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$ and the case where k is outside of these bounds. In the first case the lower bound on the approximation value is $\lceil t/2 \rceil$. In the second case we need to guarantee that the approximation value is inside the interval of all correct nodes. Otherwise the Byzantine nodes might choose values that deviate arbitrarily from the values of the correct nodes. With this restriction the approximation of the k^{th} smallest value can be up to t positions away from the actual value.

Theorem 1. *Assume $k \in \big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$. Then, no deterministic algorithm can choose a value that is closer than $\lceil t/2 \rceil$ to the actual k^{th} smallest value when t Byzantine nodes are present in the system.*

Proof. We consider $t + 1$ cases for which the values that the correct nodes received only differ in t values that were sent by the Byzantine nodes. We will show that the correct nodes are not able to distinguish the given views, while the k^{th} smallest values in any two views differ by up to t positions. Figure 1 shows such an example for $t = 4$ with t different values that might potentially be the k^{th} smallest value.

Let $\max(S)$ and $\min(S)$ respectively denote the maximum and minimum value of the correct nodes. We assume that a Byzantine node always sends the same value to all other nodes,

Fig. 1. In this example we look for the 4^{th} smallest value in a system with $(n-t)=7$ correct nodes and $t=3$ Byzantine nodes. The correct nodes are shown in black, the Byzantine nodes in red. All values are ordered according to the axis. On the first axis, the Byzantine nodes choose their value to be larger than all correct values. On the last axis the Byzantine values are all smaller than the correct values. In between, the Byzantine nodes choose some values to be larger and some to be smaller. All four cases are not differentiable to the correct nodes, since all nodes just see n values. The correct nodes do not know the position of the Byzantine nodes. Therefore, any of the blue marked correct nodes might be a candidate for the actual 4^{th} smallest value. Thus, the nodes must agree on a value that minimizes the distance to each of the four candidates.



i.e., all nodes receive all t Byzantine values and values that were sent by the same node are equal. In the first case the t Byzantine nodes send values that are larger than $\max(S)$ to every correct node. In this case the new k^{th} smallest value has the same position as before, i.e., the k^{th} smallest value is $R[k]$.

In the second case we assume that one Byzantine node sends a value that is smaller than $\min(S)$ and the other $t-1$ Byzantine nodes send values that are larger than $\max(S)$. This way, the k^{th} smallest value is at position $k+1$ in the array R.

In the third case we assume that two Byzantine values are smaller than $\min(S)$. This will shift the k^{th} smallest value by 2 positions in R etc.

In the last case all t Byzantine nodes broadcast values that are smaller than $\min(S)$. Here, the Byzantine nodes shift the k^{th} smallest value to position $k+t$ in the new array, i.e., it is $R[k+t]$.

In any of the cases a node knows that its array contains exactly t Byzantine values, but the cases are indistinguishable to the node. The k^{th} smallest value can therefore be any value from the subarray $R[k,k+t]$. Choosing a value closer to k would decrease the mistake in the first case and increase it in the last. A value closer to $k+t$ does the opposite. The value that minimizes the mistake is the median value which is at most $\lceil t/2 \rceil$ positions away from all solutions. Note that for odd values of t the value must be rounded up since the median of $R[k,k+t]$ lies between two values. It is thus not possible for a deterministic algorithm to be better than $\lceil t/2 \rceil$ positions away from the optimal solution. □

Theorem 2. *For k outside $\left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$, any deterministic algorithm can be forced to choose a value further away than $\lceil t/2 \rceil$ but at most t positions away from the actual k^{th} smallest value.*

Proof. We consider the same $t+1$ cases as in the proof of Lemma 1. The median of the subarray $R[k,k+t]$ minimizes the mistake of approximating $S[k]$. If $k \leq \lceil t/2 \rceil$ the median of this subarray may be a smaller value than $S[1]$ which is the smallest correct value. The median does not satisfy the requirements for the approximation of the k^{th} value in this case since the values outside of $\left[S[1], S[n-t]\right]$ can be arbitrarily small or large. The closest value inside the interval of correct values is at position $t+1$. Therefore, the value at position $t+1$ is the one that minimizes the mistake to any $S[k]$ with $k \leq \lceil t/2 \rceil$. Analogously, the closest correct value for $k > n - \lfloor 3t/2 \rfloor$ is at position $n-t$. The guess of the k^{th} smallest value may deviate from the actual value by more than $\lceil t/2 \rceil$. If we are looking for the minimal or maximal values, i.e., $S[1]$ or $S[n-t]$, this value may deviate by t, since all Byzantine nodes may choose values smaller than the smallest correct value or larger than the largest correct value. □

V. Algorithm for the k^{th} Smallest Value

In this section we present an algorithm that selects an approximation of the k^{th} smallest value in the presence of Byzantine nodes. We will show that this algorithm gives the best approximation to $S[k]$ for all values of $k \in [0, n-t]$. As in the previous section, two cases are distinguished for which the bounds of the approximation differ: In case where $k \in \left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$, the algorithm finds a value that satisfies Definition 1. For k outside of $\left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$ the solution can not be guaranteed to satisfy Definition 1. In this case the solution will instead be at most t positions away from the actual k^{th} smallest value. We will also prove that our algorithm produces an optimal solution by showing that it matches the bounds from Section IV.

Algorithm 1. The k^{th} Smallest Value Algorithm

Phase 1 Choosing values close to the k^{th} smallest value

Input: input value x of node v
Output: new input value x^* in the vicinity of the k^{th} smallest value
every node v executes the following commands :
Broadcast x
Receive input values from every other node, store all values in the sorted array R
$x^* :=$ median of the subarray $R[k, k+f]$
if $x^* \leq R[f]$ **then** ▷ k is too small and $R[k]$ can be an arbitrarily small Byzantine value
    $x^* := R[f+1]$
else if $x^* > R[n-t]$ **then** ▷ k is too large and $R[k]$ can be an arbitrarily large Byzantine value
    $x^* := R[n-t]$
end if
return x^*

Phase 2 Determining an interval where the actual k^{th} smallest value is suspected to be

Input: x^* from Phase 1
Output: trusted interval T for every node and a guess for the k^{th} value s_k
every node v executes the following commands :
Broadcast x^*
Receive new input values from all other nodes and store them in the sorted array R
Broadcast($R[f+1], R[n-t]$)
Receive bounds $(R[f+1], R[n-t])$ from all other nodes
for every new input value x^* that is in at least $n-t$ intervals $\big[R[f+1], R[n-t]\big]$ **do**
    add x^* to the sorted array T and call it the trusted array
end for
Guess for the k^{th} value s_k := median(T)
return trusted array T, guess for the k^{th} value s_k

Phase 3 King algorithm for the k^{th} smallest value

Input: guess for the k^{th} value s_k, trusted array T
Output: consensusValue
for $i = 1$ to $t+1$ **do**
    Communication Phase:
        Broadcast(guess for the k^{th} value s_k)
        receive guesses x from all other nodes
        if some value x is received $\geq n-t$ times **then** ▷ satisfied if all correct nodes have the same input value
            Broadcast("propose x")
        end if
        if some "propose x" received $> t$ times **then** ▷ at least one correct node broadcast "propose x"
            guess for the k^{th} value $s_k := x$
        end if
    Phase King (only the King node executes this phase):
        kingValue = guess for the k^{th} value s_k
        Broadcast("suggest kingValue")
    Decision Phase:
        if $s_k ==$ kingValue or kingValue $\in \big[T[min], T[max]\big]$ **then**
            Broadcast("support kingValue")
        end if
        if "propose x" received $< n-t$ times and "support kingValue" received $> t$ times **then**
            s_k = kingValue
        end if
end for

The main idea of the algorithm is to perform a step at the beginning where each node selects a new input value that is close to the actual k^{th} smallest value that we are looking for. Denoting this value the new input value of the node, we reduce the problem of establishing agreement with a special result to a multivalued agreement where nodes can agree on any value inside the interval of all correct values, i.e., inside $\big[S[1], S[n-t]\big]$. We use the ideas of the Phase King Algorithm proposed by Berman et al. [3] to establish agreement on any value. One distinguished correct node, the King, can decide on the value that all correct nodes have to adapt. The same authors showed that $t+1$ rounds suffice to establish agreement on any input value in the presence of Byzantine nodes. A similar idea was used in [19], where a distinguished node, the Jack, proposed the value that all nodes should adapt.

Algorithm 1 presents our method in pseudo code. It is divided into three phases: In the first phase, every node has an input that is broadcast to every other node. Each node sorts the received messages in increasing order and stores them in a local array R. It picks a local approximation of the k^{th} smallest value from R and sets it to be the new input value. In the second phase, every node broadcasts its selected k^{th} smallest value and stores the received values in a sorted array. Then, all nodes exchange their interval bounds to determine the interval in which the k^{th} smallest value should lie from their perspective. The nodes also pick the median of the corresponding array to be their local guess for $S[k]$. The third phase is where the consensus is established. We use the Phase King Algorithm to make the nodes agree on one of the local guesses from Phase 2. Hereby we assume that there are $(t+1)$ predetermined King nodes known to each of the correct nodes in the system, and each such King is assigned to exactly one round of Phase 3 in the algorithm.

Throughout the algorithm we assume that the correct nodes know the total number of nodes n and the upper bound on the number of Byzantine nodes t present in the system. Since Byzantine behavior is arbitrary, we also have to consider the case where some Byzantine nodes decide not to send any value to a correct node in the first phase of the algorithm. Such a correct node would have to choose the approximation of the k^{th} smallest value from a smaller set with fewer Byzantine nodes, thus not satisfying the required approximation for the k^{th} smallest value. One possibility to prevent this case is to fill up the array R with dummy values which are assumed to be worst-case input values, i.e. all smaller than $S[1]$. We can however reach a better local approximation of the k^{th} smallest value by adjusting the number of Byzantine nodes and choosing the k^{th} smallest value directly from the smaller interval. We therefore define $f \leq t$, which denotes the number of values that are suspected to be Byzantine in the array of received values R. We assume that each correct node receives $n-t+f$ values in a round and can calculate f since it knows n and t. Note that this number f depends on the node and the communication round, since Byzantine nodes can deviate arbitrarily from the protocol.

A. Correctness of the Algorithm

In this section we will prove the correctness of Algorithm 1 and show that the algorithm performs optimally in the proposed model.

Theorem 3 (Correctness and Validity)**.** *Algorithm 1 achieves Byzantine agreement in the presence of $t < n/3$ Byzantine nodes with a valid consensus value according to Definition 1.*

Theorem 4 (Termination and Optimality)**.** *Algorithm 1 terminates in $O(t+1)$ rounds with message complexity $O((t+1)n^2)$ and achieves an optimal approximation to the k^{th} smallest value.*

B. Proof of Theorem 3

We start by considering Phase 1 and 2 of the algorithm. These are preprocessing steps that force all correct nodes to choose their new input values such that they satisfy the desired validity conditions of the algorithm. Recall that the values in all arrays and subarrays are sorted.

Lemma 1. *After the first phase of the algorithm, every node has chosen a new input value that is inside the interval $\Big[S\big[k - \lceil f/2 \rceil\big], S\big[k + \lfloor f/2 \rfloor\big]\Big] \subseteq \Big[S\big[k - \lceil t/2 \rceil\big], S\big[k + \lfloor t/2 \rfloor\big]\Big]$.*

Proof. Every node selects its input value as the median of its local subarray $R[k, k+f]$. It is sufficient to show that $S[k]$ is inside any such interval. This is because the algorithm chooses the median of this interval as a guess for $S[k]$. Therefore, it ensures any value from the interval to be at most $\lceil f/2 \rceil \leq \lceil t/2 \rceil$ positions away from $S[k]$.

To prove that $S[k]$ is inside the interval we assume that the Byzantine nodes choose values smaller than $S[k]$. Values larger than $S[k]$ do not influence the position of the k^{th} value. Every value smaller than $S[k]$ shifts the k^{th} value by one position and f Byzantine nodes can shift the k^{th} value by at most f positions. This way, $S[k]$ will always be inside the chosen interval $\big[S[k], S[k+f]\big] \subseteq \big[S[k], S[k+t]\big]$. By the observation above the chosen median of the interval will be at most $\lceil f/2 \rceil$ positions away from $S[k]$. □

This lemma shows that the new input values x^* which are generated by Phase 1 are valid values according to Definition 1 for $k \in \big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$. Other values of k will be considered separately in Section V-C.

Lemma 2. *In Phase 2, the nodes decide on a trusted interval T which is a subinterval of $\Big[S\big[k - \lceil t/2 \rceil\big], S\big[k + \lfloor t/2 \rfloor\big]\Big]$.*

Proof. Every correct node cuts off the t rightmost and leftmost values of R. The received values from correct nodes are valid values according to Lemma 1. There are at most t Byzantine nodes, and therefore also at most t values that are either too large or too small. Step 3 of Phase 2 would remove such values. A value is added to T if it is contained in at least $n-t$

intervals, at least $n - 2t$ of which came from correct nodes. Thus, all values in T are valid according to Definition 1. □

Lemma 3. *The interval T is non-empty for each correct node.*

Proof. We know that all correct nodes will receive the same $n - t$ correct values. Some of the nodes might remove at most t largest and smallest entries after sorting the values. Therefore, at least $n - 3t > 0$ central values are left inside the interval T for each correct node. Moreover, this implies that the median of all correct values x^* from Phase 2 will be inside every correct interval T. □

The analysis of Phase 3 is similar to the analysis of the Jack algorithm proposed in [19]. We emphasize the important points of the analysis in the following part.

Lemma 4. *If the King adapts the proposed value from Step 5, it will be accepted by all nodes.*

Proof. Only one value x can be proposed simultaneously in Phase 3 by correct nodes. Assume there is another value y that is proposed by some correct node. From any $n - t$ messages that a node received in Step 2, at least $n - t - f \geq n - 2t$ values were broadcast by correct nodes. If two correct nodes propose two different values x and y in Step 5, there must have been $2(n - 2t) = 2n - 4t > n - t$ correct nodes which broadcast either of the values in Step 2. This is a contradiction since each correct node broadcasts only one value. If the King adapts the proposed solution, it has received the proposals from more than t nodes, i.e., at least one correct node. Each such correct node saw other guesses for the k^{th} smallest value x at least $n - t$ times. This means that at least $n - 2t > t$ correct nodes broadcast the value and will support the kingValue in Step 13 of Phase 3. □

Note that the trimming procedure in Phase 2 may also cut off correct input values. Nevertheless, the chosen median guess is not cut off in sufficiently many correct intervals. We will show that the median of the trusted array T is inside the trusted interval of at least $(t + 1)$ correct nodes.

Lemma 5. *If the correct King proposes its own value, all correct nodes will agree on this value.*

Proof. In Phase 2, any correct node decides on a value that is the median of the array of values that it has seen in at least $n - t$ bounds. f of the received bounds could have been malicious, while $n - t - f > t$ of the bounds came from correct nodes. This way, also the value of the King was inside at least $t + 1$ correct bounds. The corresponding nodes will support the kingValue in step 13 of Phase 3. □

In the next part we will show that the decision value is valid according to Definition 1 and also satisfies the standard validity conditions from Section III.

Lemma 6 (Interval Validity)**.** *The decision value of all nodes is a valid value in the sense of Definition 1.*

Proof. By Lemma 5, the correct kingValue will be accepted by all nodes. The nodes might however have established agreement in one of the previous rounds. We need to show that any value that was accepted by all nodes is a value that was inside at least one trusted interval of a correct node. Any value that was adapted in Step 16 of Phase 3 has been supported by more than t nodes in Step 13, i.e., by at least one correct node and thus was inside its trusted interval. Since any value in the trusted interval of correct nodes is correct, the accepted value must have been correct as well. □

Lemma 7 (Any-Input Validity)**.** *The algorithm satisfies Any-Input Validity.*

Proof. To show that Any-Input Validity holds, we need to consider the first phase of the algorithm. There, the correct nodes choose their new input value. This new input value is a median of values that a node received from all other nodes, i.e., a valid value according to the definition of Any-Input Validity. In the next two rounds, the nodes establish agreement on the new input values, where they choose a value that is inside an array of some of the nodes. This value must therefore have been suggested by at least one, possibly Byzantine, node, which proves the statement. □

Lemma 8 (All-Same Validity)**.** *Algorithm 1 satisfies All-Same Validity.*

Proof. For All-Same Validity assume that all correct nodes have the same input value x. In the first phase, the Byzantine values will be either equal to x, or they will be not considered in Step 3. Therefore, the new input values x^* in Phase 2 must be equal to the value x, i.e., $x^* = x$. If a Byzantine node chooses a value unequal to x^* in Phase 2, it will land on the right or the left side of the sorted array, and will thus be outside any interval bounds $(R[f + 1], R[n - t])$ making the nodes set $s_k = x^* = x$. All correct nodes will propose x in the third phase, which will immediately lead to agreement. □

C. Proof of Theorem 4

In this section we prove that the algorithm terminates after a finite number of rounds with an optimal approximation for the k^{th} smallest value.

Lemma 9 (Termination)**.** *For $t < n/3$, Algorithm 1 requires $O(t+1)$ rounds of communication and terminates with a valid value.*

Proof. The algorithm terminates after sufficiently many nodes have accepted the King's value. Since there are $(t + 1)$ predetermined Kings at the beginning of the algorithm, there will be one King that is not Byzantine in at least one of the rounds of the algorithm. By Lemma 5, all correct nodes will accept the King's value because it is inside the interval of every correct node. This way, all correct nodes will decide on a valid value after at most $(t + 1)$ rounds. In the case when a Byzantine node proposes a valid value to sufficiently many correct nodes, the algorithm might establish agreement in one of the earlier rounds. □

The next lemma shows that Algorithm 1 achieves the best possible approximation for the k^{th} smallest value.

Lemma 10 (Optimality)**.** *Algorithm 1 finds the best possible approximation for the k^{th} smallest value.*

Proof. For $k \in \big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$, each correct node chooses in Phase 1 of Algorithm 1 the best possible approximation to the k^{th} smallest value according to the proof of Theorem 1. In the next steps, the decision value is chosen as a value inside the bounds of all correct node values. Therefore, the decision value is between the smallest and the largest approximation of the k^{th} value, and gives a value that is at most $\lceil t/2 \rceil$ positions away from $S[k]$.

In the case where k is outside of the interval $\big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$, Algorithm 1 performs differently. For $k \leq \lceil t/2 \rceil$, each node in the algorithm needs to choose its guess s_k as the $(f+1)$-st value of each node. For $k \in [n-t-f+1, n-t]$, each node chooses the $(n-t)$-th value. According to Theorem 2, the local values of the nodes are chosen optimally. With these values, the algorithm will achieve the best possible approximation. □

Thus, Algorithm 1 also finds the best approximation for the k^{th} smallest values outside the interval $\big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$.

Lemma 11 (Message Complexity)**.** *The message complexity of Algorithm 1 is $O\big((t+1) \cdot n^2\big)$.*

Proof. In the first two phases of the algorithm the nodes exchange a constant amount of values with each other node which gives an upper bound of $O(n^2)$ messages for the first part. In Phase 3, all nodes exchange their messages with all other nodes in each of the $t+1$ rounds. This gives a message complexity of $O\big((t+1) \cdot n^2\big)$ for the last rounds and also the total message complexity. □

It should be noted that the problem of finding the k^{th} smallest value can also be solved using Interactive Consistency (IC) [9], [16]. In this problem, all nodes have to agree on the same vector of n values among which $n-t$ have to be equal to the input values of each of the correct nodes. Each node can then choose the k^{th} smallest value in this vector as its decision value and thus find an optimal approximation to the k^{th} smallest value. IC protocols either need to rely on witness techniques, e.g., the Reliable Broadcast [4], [18], or require parallel execution of the Byzantine Agreement Protocols for each of the input values in order to guarantee that Byzantine nodes cannot send around different values to different nodes. The witnessing technique requires exponential message complexity [9]. The parallel execution of an Agreement Protocol increases the message complexity of a given algorithm by a factor of n, for the King algorithm this leads to message complexity in the order of $O(n^4)$. In contrast to this method our algorithm shows that it is possible to agree on a value with special requirements, such as the k^{th} smallest value, without increasing the time or message complexity of the standard multivalued Byzantine agreement protocols.

This concludes the analysis of Algorithm 1 and shows that the algorithm performs best possible in the distributed setting. In the next part we will apply the idea of Algorithm 1 to find the median of all correct nodes in the distributed setting.

VI. From the k^{th} Smallest Value to the Median

The median can be computed similarly to the k^{th} smallest value. The main difference is that the Byzantine nodes can shift the k^{th} smallest value by broadcasting values which are smaller than $S[k]$. In contrast, the median can be shifted in any direction by broadcasting larger and smaller values than the median itself. As will be shown in the next theorem, each node therefore has to search for the median inside a symmetric interval $R\big[k - \lceil f/2 \rceil, k + \lfloor f/2 \rfloor\big]$. As a validity condition we require the consensus value to be inside the interval $\Big[S\big[m - \lceil t/2 \rceil\big], S\big[m + \lfloor t/2 \rfloor\big]\Big]$, where m is the position of the median of S according to Definition 2. This validity condition was first proposed in [19]. Only the first phase of Algorithm 1 needs to be adjusted as presented in Algorithm 2.

Algorithm 2. Distributed Median Algorithm

Phase 1 Choosing values close to the median

Input: input value x of node v

Output: new input value x^* in the vicinity of the k^{th} smallest value

every node v executes the following commands :

1: Broadcast x
2: Receive input values from every other node, store all values in the sorted array R
3: $x^* :=$ median of R
4: **if** $x^* \leq R[f]$ **then**
5: $\quad x^* := R[f+1]$
6: **else if** $x^* > R[n-f]$ **then**
7: $\quad x^* := R[n-f]$
8: **end if**
9: **return** x^*

Theorem 5. *In Algorithm 2 the nodes agree on a value that lies within the interval $\Big[S\big[m - \lceil t/2 \rceil\big], S\big[m + \lfloor t/2 \rfloor\big]\Big]$, where m is the median of S.*

Proof. As before, we only need to show that every correct node will decide on a median that is within the interval $\Big[S\big[m - \lceil f/2 \rceil\big], S\big[m + \lfloor f/2 \rfloor\big]\Big]$ of all correct nodes after the first phase of the algorithm. The Byzantine nodes can shift the median in both directions. Note that placing one value at a position before the median and another one after does not shift the median in the array. The worst case is when all f Byzantine values lie on one side of the actual median. In this case the median is shifted $\lceil f/2 \rceil \leq \lceil t/2 \rceil$ positions away from its actual position. It is guaranteed that the guess of the k^{th} smallest value s_k is not further away than $\lceil f/2 \rceil$ positions from the actual median since every node picks the median of R as its new

input value. Thus, all correct nodes will choose their new input value inside the interval $\Big[S\big[m - \lceil t/2 \rceil\big], S\big[m + \lfloor t/2 \rfloor\big]\Big]$. □

With Phase 2 and 3 as in Algorithm 1, the presented Distributed Median Algorithm computes an approximation for the median which is optimal.

VII. Vector Consensus

A. Motivation

Vector consensus is a generalization of the one-dimensional consensus where the nodes have more than one input value and the values of single components are comparable. The idea is to determine a representative vector which is close to the vectors of all correct nodes. There are a number of applications that requires vector consensus. One example is the distributed facility location problem, where nodes need to minimize their distance to a median location. Another example are voting protocols, where the voters need to determine a representative median voter. Using the definition of the geometric median in multiple dimensions does however seem difficult since there is no explicit formula to compute a median and only iterative solutions provide an approximation.

In this section, we use the Distributed Median Algorithm to generalize consensus to multiple dimensions. Similar approaches can be found in [14], [15], [22]. In these papers the authors generalize the idea of removing the t leftmost and rightmost values of the sorted array to several dimensions. They therefore compute the convex hull of every $n - t$ nodes, show that the intersection of all such convex hulls is non-empty and determine a central point inside the intersection. The number of possible convex hulls does however become exponential for large t and computing a central point inside the intersection is therefore costly [22]. The papers instead propose approximate algorithms that can also be applied to asynchronous consensus. In this section we want to derive an exact version of the vector agreement, while restricting our computation to a linear number of rounds in t. We relax the condition of the consensus value from being inside the convex hull to a value that is inside the range of all correct values in each component. In addition, we have to drop the Any-Value Validity condition from Section III, while the All-Same Validity condition still holds.

B. Generalization to d Dimensions

First, we introduce a new validity condition for the general case:

Definition 3 (Box Validity)**.** *Let $v^1, \ldots, v^{n-t} \in \mathbb{R}^d$ be the input values of all correct nodes. A vector $w \in \mathbb{R}^d$ satisfies box validity, if for each component $i \in [d]$ holds*

$$\min(v_i^1, \ldots, v_i^{n-t}) \leq w_i \leq \max(v_i^1, \ldots, v_i^{n-t})$$

We generalize the one-dimensional case by applying Algorithm 2 to compute the median of each coordinate separately.

Lemma 12. *The algorithm terminates in the presence of $t < n/3$ Byzantine nodes with a value that is valid according to Definition 3.*

Algorithm 3. Vector Consensus

Consensus in d Dimensions

Input: n input vectors $v^1, \ldots, v^n \in \mathbb{R}^d$
Output: consensusValue $m \in \mathbb{R}^d$

1: **for** $i = 1$ *to* d **do**
2: use Algorithm 2 to compute the median m_i of the values $v_i^1, \ldots, v_i^n$
3: **end for**
4: **return** $(m_1, \ldots, m_d)$

Proof. The algorithm can tolerate up to $(n-1)/3$ Byzantine values, since every component is considered separately. In every component the t leftmost and rightmost values are cut off, and the median is computed according to Algorithm 2. Assume some Byzantine value was outside of the interval in some other dimension of the vector. It is not possible for the algorithm to determine whether the node that is an outlier in some component is actually Byzantine. Therefore, we do not need to remove this component of the vector from the set of all nodes. This way we can compute the medians in each component without restricting the number of Byzantine nodes. Observe that the computed median is at most $\lceil t/2 \rceil$ positions away from the correct median in each component and is in the range of all correct values in this component. Thus, the value is guaranteed to satisfy box validity. □

Lemma 13. *Algorithm 3 satisfies All-Same Validity.*

Proof. Assume all correct nodes decided for the same vector. Then, the correct nodes will propose the same value to agree on in each iteration of Algorithm 3. By Lemma 8 the nodes will agree on the same values in each component, and therefore on the same vector in the end. □

With this algorithm it is possible to make the nodes agree on a vector that is not too far away from all vectors that were proposed by the correct nodes. The number of Byzantine nodes does not change with the dimension of the input vectors, and the number of rounds is bounded by the number of Byzantine nodes.

VIII. Conclusion

In this paper we presented a variation of the multivalued agreement problem, where the nodes are required to agree on a particular value from the interval of all correct nodes. We showed that no deterministic algorithm can solve this problem in the presence of Byzantine nodes, but can only approximate the value with an accuracy of $\lceil t/2 \rceil$ positions away from the actual value. We proposed an algorithm for consensus on the k^{th} smallest value in the synchronous message passing model that matches this bound. Using the same algorithm we were able to improve the result of [19] and find the best possible approximation of the median of all correct nodes. While the algorithm performs optimally in the one-dimensional case,

the idea seems not to be applicable to find a representative value for multi-dimensional consensus. In a vector space the ordering of the input vectors is not well-defined and it becomes computationally expensive to find a representative vector in the presence of Byzantine nodes [14], [15], [22]. Relaxing the validity condition gives a result that is independent of the dimension and can tolerate up to a third of Byzantine nodes.

REFERENCES

[1] Mayank Bawa, Aristides Gionis, Hector Garcia-Molina, and Rajeev Motwani. The price of validity in dynamic networks. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD, June 2004.

[2] Piotr Berman and Juan A. Garay. *Asymptotically optimal distributed consensus*. ICALP. July 1989.

[3] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards Optimal Distributed Consensus. In *30th Annual Symposium on Foundations of Computer Science*, FOCS, October 1989.

[4] Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.

[5] Soma Chaudhuri. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems . *Information and Computation* , 105(1):132 – 158, 1993.

[6] Benjamin Doerr, Leslie Ann Goldberg, Lorenz Minder, Thomas Sauerwald, and Christian Scheideler. Stabilizing Consensus with the Power of Two Choices. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA, June 2011.

[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*, 33(3):499–516, July 1986.

[8] Alan D. Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4(1):9–29, 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*, 32(2):374–382, April 1985.

[11] Matthias Fitzi and Martin Hirt. Optimally Efficient Multi-valued Byzantine Agreement. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC, July 2006.

[12] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[13] Hammurabi Mendes and Maurice Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. In *31st International Symposium on Distributed Computing (DISC 2017)*.

[14] Hammurabi Mendes and Maurice Herlihy. Multidimensional Approximate Agreement in Byzantine Asynchronous Systems. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC, June 2013.

[15] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, 28(6):423–441, January 2015.

[16] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[17] Roberto De Prisco, Dahlia Malkhi, and Michael Reiter. On k-Set Consensus Problems in Asynchronous Systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(1), 2001.

[18] T.K. Srikanth and Sam Toueg. Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms. *Distributed Computing*, 2(2):80–94, June 1987.

[19] David Stolz and Roger Wattenhofer. Byzantine Agreement with Median Validity. In *19th International Conference on Priniciples of Distributed Systems*, OPODIS, December 2015.

[20] Sam Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC, August 1984.

[21] Russell Turpin and Brian A. Coan. Extending binary Byzantine agreement to multivalued Byzantine agreement . *Information Processing Letters* , 18(2):73 – 76, February 1984.

[22] Nitin H. Vaidya and Vijay K. Garg. Byzantine Vector Consensus in Complete Graphs. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC, July 2013.

[23] Zhuolun Xiang and Nitin H. Vaidya. Relaxed Byzantine Vector Consensus. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*.

[24] Zhuolun Xiang and Nitin H. Vaidya. Brief Announcement: Relaxed Byzantine Vector Consensus. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA, July 2016.