Multi-Valued Connected Consensus: A New Perspective on Crusader Agreement and Adopt-Commit

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

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

Abstract

Algorithms to solve fault-tolerant consensus in asynchronous systems often rely on primitives such as crusader agreement, adopt-commit, and graded broadcast, which provide weaker agreement properties than consensus. Although these primitives have a similar flavor, they have been defined and implemented separately in ad hoc ways. We propose a new problem called *connected consensus* that has as special cases crusader agreement, adopt-commit, and graded broadcast, and generalizes them to handle multi-valued inputs. The generalization is accomplished by relating the problem to approximate agreement on graphs.

We present three algorithms for multi-valued connected consensus in asynchronous message-passing systems, one tolerating crash failures and two tolerating malicious (unauthenticated Byzantine) failures. We extend the definition of *binding*, a desirable property recently identified as supporting binary consensus algorithms that are correct against adaptive adversaries, to the multi-valued input case and show that all our algorithms satisfy the property. Our crash-resilient algorithm has failure-resilience and time complexity that we show are optimal. When restricted to the case of binary inputs, the algorithm has improved time complexity over prior algorithms. Our two algorithms for malicious failures trade off failure resilience and time complexity. The first algorithm has time complexity that we prove is optimal but worse failure-resilience, while the second has failure-resilience that we prove is optimal but worse time complexity. When restricted to the case of binary inputs, the time complexity (as well as resilience) of the second algorithm matches that of prior algorithms.

The contributions of the paper are first, a deeper insight into the connections between primitives commonly used to solve the fundamental problem of fault-tolerant consensus, and second, implementations of these primitives that can contribute to improved consensus algorithms.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases graded broadcast, gradecast, binding, approximate agreement

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2023.6

Related Version *Full Version*: https://arxiv.org/abs/2308.04646

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

1 Introduction

Solving consensus in the presence of faults is a fundamental problem in distributed computing, yet it is impossible to solve deterministically in purely asynchronous systems [25]. One way to address this impossibility is to augment the system model with unreliable *failure detectors* [14]. Several algorithms in this class (e.g., [11, 27]) combine a failure detector with a mechanism for detecting whether processes have reached unanimity, in the form of an *adopt-commit* protocol [37]. In such a protocol, each process starts with a binary input value and returns a

pair (v, g) where v is one of the input values and g is either 1 or 2. The process is said to pick v as its output value; furthermore, if $g = 2$, then it *commits* to v, and if $g = 1$, then it *adopts* v. In addition to the standard validity property that the output value is the input of some correct process, an adopt-commit protocol ensures that processes commit to at most one value, and if any process commits to a value, then no process adopts the other value.

Another way to address the impossibility of consensus is to use randomization and provide only probabilistic termination. Some algorithms in this class (e.g., [36]) rely on a mechanism called *crusader agreement* [19]: Roughly, if all processes start with the same value v, they must decide on this value, and otherwise, they may pick an *undecided* value, denoted $\perp$. Other algorithms in this class (e.g., [17]) rely on *graded broadcast* [23], also called *graded crusader agreement*, *graded consensus*, or just *gradecast*. Graded broadcast can be viewed as a combination of adopt-commit with crusader agreement: the decisions are either (v, g), where v is a binary value and g is either 1 or 2, or $\perp$ (also denoted $(\perp, 0)$). As in adopt-commit, the requirement is that processes *commit* to at most one value, but in addition, if any process *adopts* a value, then no process *adopts* the other value. In a sense, the $\perp$ value allows a separation between adopting one value and adopting a different value.

Crusader agreement, adopt-commit and graded broadcast have a very similar flavor, yet it is hard to tell them apart or to pinpoint how they relate to each other. (For example, some agreement protocols, e.g., [10, 30], state that they use graded broadcast while, in fact, they rely on an adopt-commit protocol, without a $\perp$ value.) The relation between these primitives becomes apparent when they are pictorially represented, as in Figure 1, with the possible decisions represented by vertices on a chain. The different "convergence" requirements are all special cases of the requirement that processes should decide on the *same or adjacent vertices* in the relevant chain.

With binary inputs, this description of the problems resembles *approximate agreement on the* $[0, 1]$ *real interval with parameter* ϵ [20]: processes start at the two extreme points of the interval, 0 or 1, and must decide on values that are at most ϵ apart from each other. Decisions must also be valid, i.e., contained in the interval of the inputs. Approximate agreement is a way to sidestep the impossibility of solving consensus in asynchronous systems and there are many algorithms for approximate agreement (e.g., [2, 20–22]).

Indeed, crusader agreement reduces to approximate agreement with $\epsilon = \frac{1}{2}$: Run approximate agreement with your input (0 or 1) to get some output y, then choose the value in $\{0, \frac{1}{2}, 1\}$ that is closest to y (taking the smaller one if there are two such values, e.g., for $y = \frac{1}{4}$). Finally, return $\perp$ if $\frac{1}{2}$ is chosen. (A similar observation is noted in [21, 29].) Likewise, adopt-commit reduces to approximate agreement with $\epsilon = \frac{1}{3}$, and graded consensus to taking $\epsilon = \frac{1}{4}$. This connection makes it clear why *binary* crusader agreement, adopt-commit and graded broadcast can be solved in an asynchronous message-passing system, in the presence of crash and malicious (unauthenticated Byzantine) failures, within a small number of communication rounds.

In some situations, agreement must be reached on a *non-binary* value, e.g., the identity of a leader [9], or the next message to deliver in totally-ordered atomic broadcast [16]. This requires handling *multi-valued* inputs, where processes can start with an input from a set V with $|V| \geq 2$. We take inspiration from *approximate agreement on graphs* [13], in which each

Figure 1 Left: crusader agreement. Center: adopt-commit. Right: graded broadcast.

process starts with a vertex of a graph as its input and must decide on a vertex such that all decisions are *within distance one of each other* and *inside the convex hull of the inputs*. When all processes start with the same vertex, this implies they must decide on this vertex.

This paper defines a new problem, which we call *connected consensus*. Connected consensus elegantly unifies seemingly-diverse problems, including crusader agreement, graded broadcast, and adopt-commit, and generalizes them to accept *multi-valued* inputs. It can be viewed as approximate agreement on a restricted class of graphs. We primarily focus on *spider graphs* [28] consisting of a central vertex attached to which are $|V|$ paths ("branches") of length R, where R is the *refinement* parameter. We also consider a variation in which the central vertex is replaced with a clique of size $|V|$ and each branch is attached to a different vertex in the clique. (See Figures 2 and 3 in Section 3.)

Recently, the definition of binary (graded) crusader agreement was extended to include a *binding* property [3]: once the first correct process terminates, there exists a value $v \in \{0, 1\}$ such that no nonfaulty process can output v in any extension. That paper demonstrates that this property facilitates the modular design of randomized consensus algorithms that tolerate an *adaptive* adversary. We refer to [3] for an excellent description of the usage, and its pitfalls, of (graded) crusader agreement, together with a shared random coin, in randomized consensus; they show how faster (graded) crusader agreement algorithms lead to faster randomized consensus algorithms. We generalize the binding property to hold for multi-valued inputs: once the first process decides, one value is "locked", so that in all possible extensions, the decisions are on the same branch of the graph. Although the generalization is natural, we point out (in Section 3) that simply applying the original definition unchanged when inputs are multi-valued does not accomplish the desired goal when connected consensus is composed with a multi-valued shared random coin [15].

With these definitions at hand, we turn to designing algorithms for multi-valued connected consensus in asynchronous message-passing systems that tolerate crash or malicious failures and satisfy binding. There is an algorithm for approximate agreement on general graphs in the presence of malicious failures [35]. However, it requires exponential local computation and does not satisfy the binding property. We are interested in special-case graphs as described above; furthermore, we focus on the cases when the refinement parameter R equals either 1 or 2, which captures the applications of interest. Thus we exploit opportunities for optimizations to obtain better algorithms.

For crash failures, we present an algorithm for $R = 1$ and $R = 2$ with the binding property; it requires $n > 2f$, where n is the total number of processes and f is the maximum number of faulty processes, which we show is optimal. Its time complexity is R and its total message complexity is $O(n^2)$. We show that the time complexity is also optimal for reasonable resiliencies. The best previous algorithms, in [3], have slightly worse time complexity: 2 for $R = 1$ (crusader agreement) and 3 for $R = 2$ (graded crusader agreement). Moreover, both of these previous algorithms are for the binary case ($|V| = 2$) only, and cannot easily handle larger input sets.

For malicious failures, we first present a simple algorithm with binding for $R = 1$ and $R = 2$, which assumes $n > 5f$. Like the crash-tolerant algorithm, its time complexity is R and its total message complexity is $O(n^2)$. We show that the time complexity is optimal for reasonable resiliencies. Both this algorithm and our crash-tolerant algorithm derive the binding property from the inputs of the processes. That is, the assignment of input values to the processes uniquely determines which non-$\perp$ value, if any, can be decided in any execution with that input assignment. The fact that the locked value for binding is determined solely by the inputs is conducive to the development of simple and efficient algorithms. However,

Table 1 Summary of connected consensus algorithms for $R = 1$ (crusader agreement) and $R = 2$ (graded broadcast) with input set V; n is the total number of processes, f is the maximum number of faulty processes. All algorithms satisfy Binding.

failure type	crash		malicious			
algorithm	Alg. 1	[3] ($\|V\| = 2$)	Alg. 2	Alg. 3	Alg. 3 + [34]	[3] ($\|V\| = 2$)
resilience	$n > 2f$	$n > 2f$	$n > 5f$	$n > 3f$	$n > 3f$	$n > 3f$
messages	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(\|V\| \cdot n^2)$	$O(n^2)$	$O(n^2)$
time ($R = 1$)	1	2	1	5	7	5
time ($R = 2$)	2	3	2	7	9	7

we show that in the presence of malicious failures, the locked value cannot be determined solely by the inputs when $n < 5f$, even if faulty processes do not equivocate and the input set is binary (see Appendix A).

Our main algorithmic contribution is a connected consensus algorithm for $R = 1$ and $R = 2$ with binding that tolerates malicious failures. Its failure resilience is $n > 3f$; a simple proof shows that this is the optimal resilience. Its time complexity is 5 for $R = 1$ and 7 for $R = 2$, and its total message complexity is $O(|V| \cdot n^2)$, where V is the set of input values. The message complexity can be reduced to $O(n^2)$, at the cost of increasing the time complexity by 2, using a reduction from [34].

Table 1 compares our algorithms with prior work. The best previous algorithms with optimal resilience are in [3] and are for the binary case only. In [3], the algorithms are evaluated in terms of communication "rounds", giving smaller numbers than our time complexity measure; we discuss the relationship between the two measures at the end of Section 5.

To summarize, our contributions are the following:

- We define the *connected consensus* problem for inputs from a set V, with a numeric *refinement* parameter R. The problem can be reduced to real-valued approximate agreement in the binary case ($|V| = \{0, 1\}$), and is equal to approximate agreement on a specific class of graphs in the multi-valued case.
- We define the *binding property* for the multi-valued case, which previously was only defined for the binary case.
- These insights lead us to design efficient message-passing algorithms for connected consensus with $R = 1$ or 2, in the presence of crash and malicious failures, for arbitrarily large input sets. The algorithms are modular in that the $R = 2$ case is obtained by appending more communication exchanges to the $R = 1$ case.
- For crash failures, our simple algorithm is optimal in resilience, time complexity (for reasonable resiliencies), and message complexity. Its time complexity improves on the best previously known algorithms, which only handle binary inputs.
- For malicious failures, we provide two algorithms that trade off resilience and time and message complexity. One algorithm has time complexity 1 or 2 (for $R = 1$ or $R = 2$), which is optimal for reasonable resiliencies, and sends $O(n^2)$ messages, but requires $n > 5f$. The other algorithm only requires $n > 3f$, but has time complexity 5 or 7 (for $R = 1$ or $R = 2$) and sends $O(|V| \cdot n^2)$ messages. This is the same performance as the algorithms in [3] which are only for the case when $|V| = 2$.

2 Model of Computation

We assume the standard asynchronous model for n processes, up to f of which can be faulty, in which processes communicate via reliable point-to-point messages. We consider two possible types of failures: *crash* failures, when a faulty process simply ceases taking steps, and *malicious* failures, when a faulty process can change state arbitrarily and send messages with arbitrary content.

In more detail, we assume a set of n processes, each modeled as a state machine. Each process has a subset of initial states, with one state corresponding to each element of V, denoting its input. The transitions of the state machine are triggered by events. There are two kinds of *events*: spontaneous wakeup of a process and receipt of a message by a process. Note that every event is a step by some process. A transition takes the current state of the process and incoming message and produces a new state of the process and a set of messages to be sent to any subset of the processes. The state set of a process contains a collection of disjoint subsets, each one modeling the fact that a particular decision has been taken; once a process enters the subset of states for a specific decision, the transition function ensures that it never leaves that subset.

A *configuration* of the system is a vector of process states, one for each process, and a set of in-transit messages. In an initial configuration, each process is in an initial state and no messages are in transit. Given a subset of at most f processes that are "faulty" with the rest being "correct", we define an *execution* as a sequence of alternating configurations and events $C_0, e_1, C_1, \ldots$ such that:

- C_0 is an initial configuration.
- The first event for each process is wakeup. A correct process experiences exactly one wakeup, a crash-faulty process experiences at most one wakeup, and a malicious-faulty process can experience any number of wakeups.
- Suppose e_i is a step by (correct)[1] process p and let s and M be the state and set of messages resulting from p's transition function applied to p's state in C_i and m, if e_i is the receipt of message m (or nothing if e_i is a wakeup event). Then the only differences between C_i and C_{i+1} are that m is no longer in transit, M is in transit, and p's state in C_{i+1} is s. If p is malicious, then s and M can be anything.
- Every message sent by a process to a correct process is eventually received and the receipt occurs after the recipient wakes up.

Since we consider all executions that satisfy the above properties, we are capturing an "adaptive adversary", which can control the inputs, choice of faulty processes, ordering of process steps, behavior of malicious processes, and the message delays, depending on anything that has happened so far in the execution.

We study worst-case complexity measures. For communication complexity, we count the maximum, over all executions, of the number of messages sent by all the (correct) processes.

We adopt the definition in [8] for time complexity in an asynchronous message-passing system. We start by defining a *timed execution* as an execution in which nondecreasing nonnegative integers ("times") are assigned to the events, with no two events by the same process having the same time. For each timed execution, we consider the prefix ending when the last correct process decides, and then scale the times so that the maximum time that elapses between the sending and receipt of any message between correct processes is 1. We

[1] Here and throughout, the restriction to correct processes is only for the case of malicious failures.

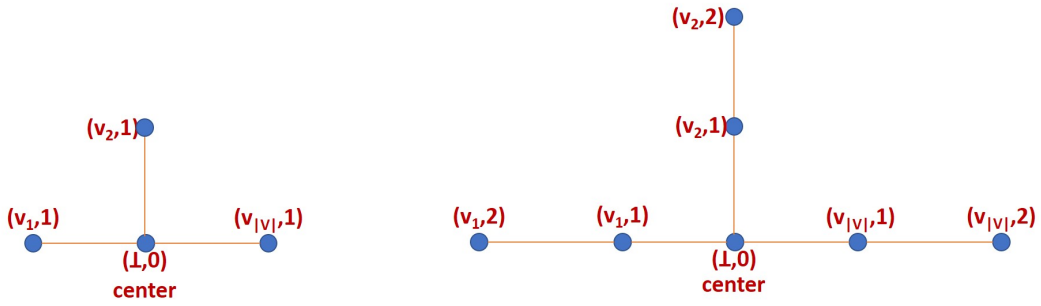

Figure 2 Spider graphs: $R = 1$ (left) and $R = 2$ (right).

define the *time complexity* as the maximum, over all such scaled timed execution prefixes, of the time assigned to the last event minus the latest time when any (correct) process wakes up. For simplicity, we assume that the first wakeup event of each process occurs at time 0. This definition of time complexity is analogous to that in [31, 34], which measures the length of the longest sequence of causally related messages.

The definition of time complexity just given applies to an algorithm with any communication structure. However, many algorithms, including several of ours, have a specific style of communication, in which each process repeatedly sends a message to all the processes, waits to receive a certain number of messages, and then sends another message. For algorithms with this structure, the sending of a message and waiting for the receipt of the specified number of messages, forms a *round*.

3 Definitions of Connected Consensus and Related Problems

Connected Consensus. Let V be a finite, totally-ordered set of values; assume $\perp \notin V$. Given a positive integer R, let $G_S(V, R)$ be the spider graph consisting of a central vertex labeled $(\perp, 0)$ that has $|V|$ paths extending from it, with one path ("branch") associated with each $v \in V$. The path for each v has R vertices on it, not counting $(\perp, 0)$, labeled $(v, 1)$ through (v, R), with (v, R) being the leaf. (See Figure 2.) Given a subset V' of V, we denote by $T(V, R, V')$ the minimal subtree of $G_S(V, R)$ that connects the set of leaves $\{(v, R)|v \in V'\}$; note that when V' is a singleton set $\{v\}$ then $T(V, R, \{v\})$ is the single (leaf) vertex (v, R).

In the *connected consensus problem for V and R*, each process has an input from V. The requirements are:

Termination: Each correct process must decide on a vertex of $G_S(V, R)$, namely, an element of $\{(v, r)|v \in V, 1 \leq r \leq R\} \cup \{(\perp, 0)\}$.

Validity: Let $I = \{(v, R)|v$ is the input of a (correct) process$\}$. The output of each (correct) process must be a vertex in $T(V, R, I)$. In particular, if all (correct) processes start with the same input v, then (v, R) must be decided.

Agreement: The distance between the vertices labeled by the decisions of all (correct) processes is at most one.

If we set $R = 1$, we get *crusader agreement* [19], originally considered in the synchronous model. If we set $R = 2$ we get *graded broadcast* [24], originally considered in the synchronous model. In asynchronous shared-memory systems, graded broadcast is also called *adopt-commit-abort* [18, 32].

Defining the Binding Property. An additional condition of interest for the connected consensus problem is called *binding* [3]. It was originally proposed for the case of binary inputs, and defined as follows: "before the first non-faulty party terminates, there is a value $v \in \{0, 1\}$ such that no non-faulty party can output the value v in any continuation of the execution." Here we generalize this property for multi-valued inputs.

Binding: In every execution prefix that ends with the first (correct) process deciding, one value is "locked", meaning that in every extension of the execution prefix, the decision of every (correct) process must be on the same branch of the spider graph.

If the first decision is not $(\perp, 0)$, then this condition follows from Agreement. More interestingly, if the first decision is $(\perp, 0)$, then there are many choices as to which branch is locked but the choice must be the same in every extension. Note that when $|V| = 2$, our definition is equivalent to the original from [3], but for larger V, our definition is stronger – the original definition only excludes one value, leaving $|V| - 1$ possible decision values, while ours excludes $|V| - 1$ values, leaving only one possible decision value.

The original definition of binding is not strong enough to handle multi-valued inputs in some cases. For example, consider the framework for solving asynchronous Byzantine Agreement by alternating calls to a (black box) connected consensus with $R = 1$ (crusader agreement) subroutine with calls to a shared random coin subroutine (cf. [3, Section 3] and [7, Section 2]). Suppose there are three processes with inputs 0, 1 and 2, and the first process, say p, to return from connected consensus gets $(\perp, 0)$ (corresponding to $\perp$ in crusader agreement). According to the original definition, some value $v \in \{0, 1, 2\}$ is no longer a possible output of the connected consensus subroutine. Since p obtains $(\perp, 0)$ from crusader agreement, it calls the shared coin; let $v' \in \{0, 1, 2\}$ be the value p gets from the shared coin. However, the original binding property still allows the adversary to make the other processes return $(v'', 1)$, with $v'' \neq v'$. This would mean that processes start the next iteration in disagreement.

When $R = 1$, there are only two vertices on any given branch of the spider graph, $(v, 1)$ and $(\perp, 0)$. This implies:

▶ **Proposition 1.** *If $R = 1$, then the Binding property implies the Agreement property.*

If $R = 2$, though, the Binding property only restricts the branch of the spider graph on which decisions can be made; both $(\perp, 0)$ and $(v, 2)$ are on the same branch, but Agreement does not permit them to both be decided.

Centerless Variants. Recall that in the *adopt-commit* problem [27, 37], processes return a pair (v, g) where v is one of the input values and g is either 1 (adopt) or 2 (commit). Thus, there is no analog of the "center" vertex. We model this with a *centerless* variant of a spider graph (see left side of Figure 3). Here, $G_S(V, R)$ is the graph consisting of a clique on the vertices $(v, 1)$ for all $v \in V$, each with a path extending from it, with $R - 1$ vertices on it, not counting $(\perp, 0)$, labeled $(v, 2)$ through (v, R), with (v, R) being the leaf. Decisions must satisfy termination, validity and agreement as specified for the variant with a center. Since the graph has no center, binding cannot be defined; indeed, when a process returns $(v, 1)$, other processes might return $(v', 1)$, for $v \neq v'$.

Instead of developing algorithms directly for the centerless problem, we note that it can be reduced to the centered problem with the same refinement parameter: Call the algorithm for the centered problem with your input u. If the return value is (v, g) with $g > 0$, then decide this value for the centerless problem; when the return value is $(\perp, 0)$ *(i.e, the center)*, decide $(u, 1)$ for the centerless problem. (See right side of Figure 3). This reduction implies the following proposition:

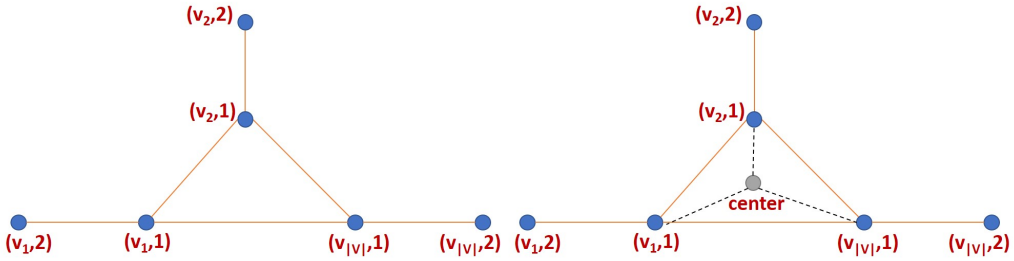

Figure 3 Centerless graph with $R = 2$ (left) and its reduction to a (centered) spider graph with $R = 2$ (right).

▶ **Proposition 2.** *If A is an algorithm that solves the (centered) connected consensus problem for R, then there is an algorithm A' that solves the centerless connected consensus problem for R.*

In the *vacillate-adopt-commit* (VAC) problem [4], the possible output values are (v,commit), (v,adopt), and (v,vacillate), where v is any value. If any output is (v,commit), then every other output is either (v,commit) or (v,adopt), for the same v. Furthermore, if there is no commit output and there is at least one (v,adopt) output, then every other output is either (v,adopt), with the same value v, or (w,vacillate), where w can be any value. At first glance, VAC seems to correspond to a centerless graph with refinement parameter $R = 3$. However, a closer look at the usage of VAC suggests that the return value of vacillate is irrelevant, in which case the problem could be represented as a centered spider graph with $R = 2$, like adopt-commit-abort and graded crusader agreement.

4 Tolerating Crash Failures

4.1 Lower Bounds

Resiliency. We first note that a standard partitioning argument shows that $n > 2f$ is required to solve connected consensus with crash failures for any $R \geq 1$, even without binding. Assume in contradiction that there is an algorithm for $n = 2f$ where $V = \{0, 1\}$ and consider execution α_0 in which all processes have input 0 and half of the processes crash initially; by Validity, the other half must decide $(0, R)$ by some time t_0. Consider execution α_1 in which all processes have input 1 and the other half of the processes crash initially; by Validity, the first half must decide $(1, R)$ by some time t_1. Finally consider execution α which is the "merger" of α_0 and α_1, in which the first half of the processes have input 0, the other half have input 1, and messages between the halves are delayed until after $\max\{t_0, t_1\}$. Since processes in the first half decide $(1, R)$ as they do in α_1 and processes in the second half decide $(0, R)$ as they do in α_0, Agreement is violated.

Round Complexity. First we note that processes cannot solve connected consensus without communicating, and thus at least one round is necessary.

For $R = 2$, we use a reduction from approximate agreement and a result in [22] to show that, as long as $n \leq 4f$, at least two rounds of message exchange are necessary to solve connected consensus.

Suppose we want to solve ϵ-approximate agreement on the interval $[0, 1]$. We show how to do so using any connected consensus algorithm A_{CC} with $V = \{0, 1\}$ and $R = \lceil \frac{1}{2\epsilon} \rceil$ that tolerates crash failures. Letting v be the approximate agreement input for process p, call A_{CC}

with input v. To obtain the approximate agreement output from the connected consensus output, map the $2R+1$ vertices of the connected consensus graph, which is a chain, in order to points in the real interval $[0,1]$ that are equally spaced, with vertex $(0,R)$ corresponding to point 0 and vertex $(1,R)$ corresponding to point 1. Since adjacent points in $[0,1]$ are distance $\frac{1}{2R}$ apart, they are within ϵ of each other.

For example, when $\epsilon = \frac{1}{4}$, we use connected consensus with $R = 2$.

This transformation uses no rounds other than those used by A_{CC}. Thus any lower bound on the number of rounds for approximate agreement is also a lower bound on the number of rounds for connected consensus.

For any round-based approximate agreement algorithm, the *convergence ratio* is the fraction by which the size of the interval of values reduces after one round. The number of rounds needed to achieve outputs that are at most ϵ apart is $\left\lceil \log_{\frac{1}{r}} \frac{U}{\epsilon} \right\rceil$, where r is the convergence ratio per round and U is the size of the interval of inputs. In our case, $U = 1$ and $\epsilon = 1/4$. Fekete proved in [22] that the best r can be is $\left\lceil \frac{n-f}{f} \right\rceil^{-1}$. Thus, as long as $n \leq 4f$, the number of rounds is at least 2.

In the full version we show that, if the resilience is higher than $4f$, a simple one-round connected consensus algorithm with Binding is possible for $R = 2$.

4.2 Algorithm

We next present an algorithm for connected consensus with binding for n processes that tolerates $f < n/2$ crash failures. (See Algorithm 1.) The algorithm is extremely simple and efficient, using one round of message exchanges for $R = 1$ and two rounds of message exchanges for $R = 2$, and thus using only $O(n^2)$ messages. Due to the simple communication structure, the number of rounds of message exchanges is equal to the time complexity. The lower bounds discussed in the previous subsection show that the time complexity is optimal for $R = 1$, and that for $R = 2$, it is optimal as long $n \leq 4f$.

In the first round of the algorithm, processes exchange their inputs. After hearing from $n - f$ processes, each process chooses branch v of the spider graph, if all the received values equal v, or the center vertex otherwise. If $R = 1$, then the process decides on the branch. Otherwise, processes exchange branch values (v or $\perp$) in a second round in order to decide on a vertex on the v branch. After waiting for $n - f$ messages in the second round, if a process' branch is $\perp$, then it decides $(v, 1)$ if at least one second-round message is for v and $(\perp, 0)$ otherwise. If the process' branch is v, then it decides $(v, 2)$ if all the second-round messages are for v and $(v, 1)$ otherwise.

▶ **Theorem 3.** *If $n > 2f$, then Algorithm 1 solves binding connected consensus for $R = 1$ and $R = 2$ with n processes, up to f of which can crash. It takes 1 time unit and sends $O(n^2)$ messages for $R = 1$ and takes 2 time units and sends $O(n^2)$ messages for $R = 2$.*

The proof of this theorem appears in the full version. Here, we only outline why the algorithm is binding. In fact, we show a stronger property, that the branch along which decisions are made is determined solely by the inputs.

For any assignment of inputs to the processes, since $n > 2f$, there is at most one input value $v \in V$ that occurs at least $n - f$ times. Note that if p sets its branch variable to v, then all $n - f$ INPUT messages it receives are for v, and since processes fail only by crashing, the $n - f$ senders of these messages all have input v. Therefore, no process can set its branch variable to any value in V other than v. For $R = 1$, processes decide on their branch variables, implying that in any future extension, every process that sets its branch variable sets it to v

Algorithm 1 Connected Consensus algorithm with Binding for $R = 1, 2$ with n processes, $f < n/2$ of which may crash; code for process p.

```
1:  send ⟨INPUT,input⟩ to all                                   ▷ round 1
2:  wait for n − f INPUT messages
3:  let W be set of values received in INPUT messages
4:  if ∃v ∈ V s.t. W = {v} then
5:      branch := v
6:  else
7:      branch := ⊥
8:  if R = 1 then
9:      if branch = ⊥ then
10:         decide (⊥, 0)                                       ▷ center vertex
11:     else
12:         decide (branch,1)                                   ▷ leaf vertex for chosen branch
13: else                                                        ▷ R = 2; round 2
14:     send ⟨BRANCH,branch⟩ to all
15:     wait for n − f BRANCH messages
16:     if branch = ⊥ then
17:         if ∃v ∈ V s.t. at least one BRANCH message has value v then
18:             decide (v, 1)                                   ▷ middle vertex on branch for v
19:         else
20:             decide (⊥, 0)                                   ▷ center vertex
21:     else                                                    ▷ branch ≠ ⊥
22:         if ∃v ∈ V s.t. all BRANCH messages have value v then
23:             decide (v, 2)                                   ▷ leaf vertex for v
24:         else
25:             decide (branch,1)                               ▷ middle vertex on branch chosen in round 1
```

or $\perp$. For $R = 2$, processes exchange their branch variables in the second round. The only possible non-$\perp$ value that can be exchanged is v, so the only possible decisions are $(\perp, 0)$, $(v, 1)$ and $(v, 2)$, which proves binding for $R = 2$.

5 Tolerating Malicious Failures

5.1 Lower Bounds

Resiliency. We first note that $n > 3f$ is required to solve connected consensus for any $R \geq 1$ with malicious failures, even without binding. This simple lower bound can be derived from [35]; we provide a complete proof in the full version.

Round Complexity. To show a lower bound on the round complexity, we use the same reduction from approximate agreement as for crash failures in Section 4.1, except that A_{CC} tolerates malicious failures. We then appeal to a result in [20] to show that, as long as $n \leq 9f$, at least two rounds of message exchange are necessary to solve connected consensus when $R = 2$, in the presence of malicious failures.

The relevant result in [20] is that the best the convergence ratio r can be in the presence of malicious failures is $\left\lceil \frac{n-3f}{2f} \right\rceil^{-1}$. Plugging this value of r into the formula $\left\lceil \log_{\frac{1}{r}} \frac{U}{\epsilon} \right\rceil$, with $U = 1$ and $\epsilon = 1/4$, results in a number of rounds that is at least 2 as long as $n \leq 9f$.

If the resilience is sufficiently large, $n > 12f$, a simple one-round connected consensus algorithm is possible for $R = 2$. Furthermore, if $n > 13f$, then the algorithm also satisfies Binding. These algorithms are presented in the full version.

5.2 Algorithms

We next present two algorithms for connected consensus, tolerating malicious failures. We start with a simple algorithm for $n > 5f$ using ideas from [20]. Then we present a more complex algorithm for $n > 3f$, which is a modular extension of algorithms in [3] incorporating new ideas inspired by [34] to deal with multi-valued inputs. Although the $n > 3f$ algorithm has better resilience than the $n > 5f$ algorithm, it has worse time complexity. The round lower bounds discussed above show that it is optimal for $R = 1$ and for $R = 2$, it is optimal as long as $n \leq 9f$.

5.3 Algorithm for $n > 5f$

We now present an algorithm for connected consensus with binding for n processes that tolerates $f < n/5$ malicious failures. The algorithm is extremely simple and efficient, using one round of message exchanges for $R = 1$ and two rounds of message exchanges for $R = 2$, and thus using only $O(n^2)$ messages.

The pseudocode, which is similar to Algorithm 1, appears in Algorithm 2. In the first round of the algorithm, processes exchange their inputs and, after hearing from $n - f$ processes, each process drops the f smallest and f largest values received, an idea inspired by approximate agreement algorithms (e.g., [20]). Then each process chooses branch v of the spider graph, if all the remaining values equal v, or the center vertex otherwise. If $R = 1$, then the process decides on the branch. Otherwise, processes exchange branch values (v or $\perp$) in a second round in order to decide on a vertex on the v branch. This is done in a manner that is similar to the second round in our crash-resilient algorithm, Algorithm 1. After waiting for $n - f$ messages in the second round, if a process' branch is $\perp$, then it decides $(v, 1)$ if at least $f + 1$ second-round messages are for v and $(\perp, 0)$ otherwise. If the process' branch is v, then it decides $(v, 2)$ if at least $n - 2f$ second-round messages are for v and $(v, 1)$ otherwise.

▶ **Theorem 4.** *If $n > 5f$, then Algorithm 2 solves binding connected consensus for $R = 1$ and $R = 2$ with n processes, up to f of which can be malicious. It takes 1 time unit and sends $O(n^2)$ messages for $R = 1$ and takes 2 time units and sends $O(n^2)$ messages for $R = 2$.*

The proof of this theorem appears in the full version. Here, we only outline why the algorithm is binding, which similarly to Algorithm 1 follows from a stronger property, that the branch along which decisions are made is determined solely by the inputs.

Given an assignment of inputs to the processes, suppose there is one execution in which a correct process decides $u \in V$ and another execution in which a correct process decides $v \in V$ with $u < v$. It can be shown that at least $n - 3f$ correct processes have input at most u, and at least $n - 3f$ correct processes have inputs at least v. Thus the total number of processes n is at least $2(n - 3f)$ plus the f faulty processes. That is, $n \geq 2(n - 3f) + f$, which implies $n \leq 5f$, a contradiction. This implies binding for $R = 1$. For $R = 2$, this argument shows that there is only one possible $v \in V$ that can appear in correct processes' branch variables at the end of round 1 in any execution. Thus no correct process can get more than f BRANCH messages for any $u \in V$ other than v and thus it cannot decide $(u, 1)$ or $(u, 2)$ in any execution.

Algorithm 2 Connected Consensus algorithm with Binding for $R = 1, 2$ with n processes, $f < n/5$ of which may be malicious; code for process p.

```
1:  send ⟨INPUT,input⟩ to all                                          ▷ round 1
2:  wait for n − f INPUT messages
3:  let W be multiset of values received in INPUT messages, dropping f smallest and f largest
4:  if ∃v ∈ V s.t. every element in W is v then
5:      branch := v
6:  else
7:      branch := ⊥
8:  if R = 1 then
9:      if branch = ⊥ then
10:         decide (⊥, 0)                                              ▷ center vertex
11:     else
12:         decide (branch,1)                                          ▷ leaf vertex for chosen branch
13: else                                                               ▷ R = 2; round 2
14:     send ⟨BRANCH,branch⟩ to all
15:     wait for n − f BRANCH messages
16:     if branch = ⊥ then
17:         if ∃v ∈ V s.t. at least f + 1 BRANCH messages have value v then
18:             decide (v, 1)                                          ▷ middle vertex on branch for v
19:         else
20:             decide (⊥, 0)                                          ▷ center vertex
21:     else                                                           ▷ branch ≠ ⊥
22:         if ∃v ∈ V s.t. at least n − 2f BRANCH messages have value v then
23:             decide (v, 2)                                          ▷ leaf vertex for v
24:         else
25:             decide (branch,1)                                      ▷ middle vertex on branch chosen in round 1
```

5.4 Algorithm for $n > 3f$

We now present an algorithm for connected consensus with binding for n processes that tolerates $f < n/3$ malicious failures. The time complexity for $R = 1$ is 5 and for $R = 2$ is 7, while the message complexity is $O(|V| \cdot n^2)$ in both cases. The failure-resiliency is optimal, per the discussion at the beginning of this section. The $|V|$ factor in the message complexity can be reduced to a constant, resulting in $O(n^2)$ message complexity, by first employing the RD-broadcast primitive in [34], which reduces the number of values under consideration to 6, at the cost of $O(n^2)$ additional messages and two additional time units.

The algorithm is a modular combination of Algorithms 4 (for $R = 1$) and 6 (for $R = 2$) in [3], which work when $|V| = 2$, with the addition of a mechanism from the MV-broadcast in [34] to handle $|V| > 2$. MV-broadcast is a primitive to reduce the number of input values under consideration to two in the context of solving consensus.

Processes exchange input values in increasing "levels" of ECHO messages. Initially, processes exchange their inputs via ECHO messages and also use ECHO messages to amplify values that have been received at least $f + 1$ times; this threshold ensures that at least one correct process has that value as its input. To handle the situation when $|V| > 2$ and there may not be any message that is sent in at least $f + 1$ ECHO messages, an ECHO message for $\perp$ is initiated if a process receives at least $f + 1$ ECHO messages in addition to those for the most commonly received value; this condition only holds if there are at least two different input values at the correct processes. This early appearance of $\perp$ requires some later modifications, mentioned below, to the original algorithm.

Whenever a process receives $n - f$ ECHO messages for some value v, it stores v in its local set variable "approved"; the first time this happens, it sends the value in an ECHO2 message. The $n - f$ threshold ensures some level of uniformity in processes' "approved" sets. Each process sends one ECHO3 message, either for $\perp$ if it collects more than one approved value, or for value v if it receives $n - f$ ECHO2 messages for v. The ECHO3 messages have the desirable property that only one non-$\perp$ value is sent in them by the correct processes. When $R = 1$, processes decide once at least $n - f$ ECHO3 messages have been received: if there are at least $n - f$ for the same value v, then $(v, 1)$ is decided, otherwise if there are at least two approved values or if $\perp$ is approved, then $(\perp, 0)$ is decided. (Checking if $\perp$ has been approved here and later are modifications needed because of the possibility that $\perp$ is sent in ECHO messages.)

When $R = 2$, processes continue for two more levels in order to refine the values obtained so far on the chosen branch. The value chosen as the decision in the $R = 1$ case is sent in an ECHO4 message; these message inherit the property that at most one non-$\perp$ appears in those sent by correct processes. Each process waits for ECHO4 messages. If eventually it collects $n - f$ for a common value, then it sends an ECHO5 message for that value; if eventually it has more than one approved value or if $\perp$ is approved, it sends an ECHO5 message for $\perp$. ECHO5 messages also inherit the property that at most one non-$\perp$ appears in those sent by correct processes.

The decision is based on ECHO4 and ECHO5 messages received. If a process receives at least $n - f$ ECHO5 messages for the same non-$\perp$ value, then it decides $(v, 2)$. If it receives at least $n - f$ ECHO5 messages for $\perp$, then it decides $(\perp, 0)$. Otherwise, if it has approved either $\perp$ or at least two values, receives at least one ECHO5 message for some non-$\perp$ value w, and has at least $f + 1$ ECHO4 messages for w, it decides $(w, 1)$.

The pseudocode is in Algorithm 3. The presentation differs from that of our Algorithms 1 and 2 and of the algorithms in [3]. Instead of using syntactic constructs such as "wait until" and "upon" receiving certain messages, our code is purely interrupt-driven in order to clarify the interactions between the receipts of different messages and the conditions triggering various actions. The technique inspired by [34] appears in Lines 14 through 15. We denote by sum(A), where A is an array of integers, the sum of all the entries in A. A correct process sends at most one ECHO message for any $v \in V \cup \{\perp\}$, and at most one ECHO2, ECHO3, ECHO4, and ECHO5 message. This allows us to assume there is some mechanism for eliminating duplicate messages that arrive from the same (faulty) sender.

▶ **Theorem 5.** *If $n > 3f$, then Algorithm 3 solves binding connected consensus for $R = 1$ and $R = 2$ with n processes, up to f of which can be malicious. It takes 5 time units and sends $O(|V| \cdot n^2)$ messages for $R = 1$ and takes 7 time units and sends $O(|V| \cdot n^2)$ messages for $R = 2$.*

The complete proof of Theorem 5 appears in the full version and is sketched below.

Proof (Sketch). First, we argue that the values sent in ECHO messages by correct processes are "valid": if the value is in V, then some correct process has input v, whereas if the value is $\perp$, then not all the correct processes have the same input. The latter property is ensured by lines 14 through 15 for the following reason. The first correct process to send ECHO for $\perp$ does so because it receives at least $f + 1$ ECHO messages for values other than the most frequently occurring value m of the ECHO messages it has received so far. Letting x be the number of ECHO messages received for m, it follows that at least $x + 1$ of the ECHO messages for values other than m are from correct processes. But they cannot all be for the same value since no value occurs more frequently than m.

Algorithm 3 Connected Consensus algorithm with Binding for $R = 1, 2$ with n processes, $f < n/3$ of which may be malicious; code for process p.

initially:
approved := $\emptyset$ (subset of $V \cup \{\perp\}$) ▷ *set of approved values*
num_echo[v] := 0 for $v \in V \cup \{\perp\}$ ▷ *# received* ECHO *messages for* v
num_echo$i$[v] := 0 for $2 \le i \le 5$, $v \in V \cup \{\perp\}$ ▷ *# received* ECHO-i *messages for* v
sent_echo[v] := false for $v \in V \cup \{\perp\}$ ▷ *has* p *sent an* ECHO *message for* v *yet?*
sent_echo$i$:= false for $2 \le i \le 5$ ▷ *has* p *sent an* ECHO-i *message yet?*
decided := false ▷ *has* p *decided yet?*

wakeup:
send ⟨ECHO,input⟩ to all; sent_echo[input] := true ▷ *initiate* ECHO *for* p*'s input*

receive ⟨ECHO, v⟩:
num_echo[v] + +
if (num_echo[v] = $f + 1$) and (!sent_echo[v]) **then**
  send ⟨ECHO, v⟩ to all; sent_echo[v] := true ▷ *echo* v *if enough support but only once*
else if (sum(num_echo) − num_echo[m] ≥ $f + 1$) and (!sent_echo[$\perp$]),
where m is s.t. num_echo[m] ≥ num_echo[u] for all $u \in V \cup \{\perp\}$) **then**
  ▷ *if evidence for multiple correct inputs then initiate* ECHO *for* $\perp$
  send ⟨ECHO, $\perp$⟩ to all; sent_echo[$\perp$] := true
else if num_echo[v] = $n - f$ **then**
  if !sent_echo2 **then** ▷ *send only one* ECHO2
    send ⟨ECHO2, v⟩ to all; sent_echo2 := true
  add v to approved
  if (|approved| > 1) and (!sent_echo3) **then** ▷ *send only one* ECHO3
    send ⟨ECHO3, $\perp$⟩ to all; sent_echo3 := true

receive ⟨ECHO2, v⟩:
num_echo2[v] + +
if (num_echo2[v] = $n - f$) and (!sent_echo3) **then** ▷ *send only one* ECHO3
  send ⟨ECHO3, v⟩ to all; sent_echo3 := true

receive ⟨ECHO3, v⟩:
num_echo3[v] + +
if (sum(num_echo3) ≥ $n - f$) and ((|approved| > 1) or ($\perp \in$ approved)) **then**
  if ($R = 1$) and (!decided) **then** ▷ *decide only once*
    decide ($\perp$, 0); decided := true ▷ *center vertex*
  else if ($R = 2$) and (!sent_echo4) **then** ▷ *send only one* ECHO4
    send ⟨ECHO4, $\perp$⟩ to all; sent_echo4 := true
else if num_echo3[v] ≥ $n - f$ **then**
  if ($R = 1$) and (!decided) **then** ▷ *decide only once*
    decide (v, 1); decided := true ▷ *leaf vertex for* v
  else if ($R = 2$) and (!sent_echo4) **then** ▷ *send only one* ECHO4
    send ⟨ECHO4,v⟩ to all; sent_echo4 := true

▷ *continued.....*

▷ .....*Continuation of Algorithm 3*

38: receive $\langle$ECHO4$, v\rangle$:
39: num_echo4$[v]++$
40: **if** (num_echo4$[v] = n - f$) and (!sent_echo5) **then** ▷ *send only one* ECHO5
41: send $\langle$ECHO5$, v\rangle$ to all; sent_echo5 := true
42: **else if** (sum(num_echo4) $\geq n - f$) and ((|approved| > 1) or ($\perp \in$ approved)) and (!sent_echo5) **then**
43: send $\langle$ECHO5$, \perp\rangle$ to all; sent_echo5 := true

44: receive $\langle$ECHO5$, v\rangle$:
45: num_echo5$[v]++$
46: **if** !decided **then** ▷ *decide only once*
47: **if** ($v \in V$) and (num_echo5$[v] \geq n - f$) **then**
48: decide $(v, 2)$; decided := true ▷ *leaf vertex for* v
49: **else if** (sum(num_echo5) $\geq n - f$) and ((|approved| > 1) or ($\perp \in$ approved)) and there exists $w \in V$ s.t. (num_echo5$[w] \geq 1$) and (num_echo4$[w] \geq f + 1$) **then**
50: decide $(w, 1)$; decided := true ▷ *middle vertex on branch for* w
51: **else if** num_echo5$[\perp] \geq n - f$ **then**
52: decide $(\perp, 0)$ ▷ *center vertex*

Since a correct process approves a value when it gets $n - f$ ECHO messages for it, the validity of the ECHO messages implies validity of the approved values. In addition, the approved sets of all the correct processes are eventually the same. The ECHO2 messages preserve the validity properties of the ECHO messages and also ensure that the values sent in them end up being approved. The ECHO3 messages add a "uniqueness" property, ensuring at most one non-$\perp$ value is sent by correct processes. They also satisfy a modified approval property: if v is sent by a correct process and v is not $\perp$, then eventually every correct process approves v, otherwise (if $v = \perp$) eventually every correct process either approves $\perp$ or approves at least two values.

We can now prove correctness and complexity when $R = 1$. *Validity* is immediate from the validity properties ensured by the ECHO* messages. *Agreement* follows from *Binding* (cf. Proposition 1), which we discuss next. The first correct process to decide receives $n - f$ ECHO3 messages, at least $n - 2f$ of which are from correct processes. If any of these messages are for a value in V, then by uniqueness of ECHO3 messages, no correct process can send an ECHO3 message for a different non-$\perp$ value, and thus no such value can be decided subsequently. If all of these messages are for $\perp$, then other processes can receive at most $2f$ ECHO3 messages for any $v \in V$ (f from the correct processes that did not send ECHO3 for $\perp$ and f from the faulty processes), which is not enough support for deciding v subsequently.

We next address *Termination and time complexity.* In the full correctness proof, we first show that the algorithm terminates, before we scale all the message delays by the duration of the longest one. We first show that every correct process sends an ECHO2 message by time 2. If at least $f + 1$ correct inputs are the same, say v, then by time 1, every correct process receives the initial ECHO messages for v, and sends an ECHO message for v if it has not already done so. Thus every correct process receives at least $n - f$ ECHO messages by time 2, and sends ECHO2.

The more interesting case, which only arises when $|V| > 2$, is when no value occurs at least $f + 1$ times among the inputs of the correct processes. Let x_i (resp., y_i) be the number of ECHO messages for v_i received by a correct process p from correct (resp., faulty) processes

by time 1, $1 \leq i \leq |V|$. Note that $x_i \leq f$ and that $x_1 + \ldots + x_{|V|} \geq n - f$ since p has received all the ECHO messages sent by the correct processes initially. Let v_m be the value that occurs most frequently among all the ECHO messages received by time 1 (breaking ties arbitrarily). Then the total number of ECHO messages minus the number of those for v_m is

$$\left(\sum_{i=1}^{|V|}(x_i + y_i)\right) - (x_m + y_m) = \sum_{i=1}^{|V|} x_i + \sum_{\substack{i=1 \\ i \neq m}}^{|V|} y_i - x_m \geq (n - f) - f \geq f + 1,$$

since $n > 3f$. Thus p sends an ECHO message for $\perp$ by time 1 if it hasn't already done so, and so by time 2, every correct process receives $n - f$ ECHO messages for $\perp$ and sends ECHO2 for $\perp$ if it has not already sent an ECHO2 message.

We next show that every correct process p sends an ECHO3 message by time 4. We rely on the fact that if v is in a correct process' approved set at time t, then every correct process approved set by time $t + 2$, which implies that if v is sent by a correct process in an ECHO2 message, then every correct process approves v by time 4. Since ECHO2 messages are sent by correct processes by time 2, at least $n - f$ arrive at p by time 3. If they are all for a common value v, then p sends ECHO3 for v by time 3. Otherwise, p waits until it has at least two approved values. Process p must have received an ECHO2 for value v_1 from a correct process and an ECHO2 for a different value v_2 from a different correct process. Thus p approves v_1 and v_2 by time 4 and sends ECHO3 by time 4.

We use a similar argument to the previous paragraph to show that every correct process p decides by time 5. It relies on the key fact that the approval of values sent in ECHO3 messages by the correct processes takes place by time 5. Thus p either receives at least $n - f$ ECHO3 messages for a common value by time 5 or has approved either $\perp$ or at least two values by time 5, and thus decides by time 5.

The *message complexity* is $O(|V| \cdot n^2)$ since each correct process sends to all the processes at most one ECHO message for each $v \in V$, one ECHO2 message, and one ECHO3 message.

We continue to prove correctness and complexity when $R = 2$. First note that when $R = 2$, a process sends an ECHO4 message for v under exactly the same circumstances that it decides $(v, 1)$ (if $v \in V$) or $(v, 0)$ (if $v = \perp$) when $R = 1$. Thus we have the analogous properties for ECHO4 messages that we had for ECHO3 messages (validity, uniqueness, and approval). These properties also carry over to ECHO5 messages.

For *Agreement*, we first recall that all the non-$\perp$ values sent in ECHO4 and ECHO5 messages are the same, call it v. It remains to show that if a correct process p decides $(v, 2)$ then no correct process can decide $(\perp, 0)$. Since p decides $(v, 2)$, it receives $n - f$ ECHO5 message for v. But since $n > 3f$, it's not possible for another process to receive $n - f$ ECHO5 messages for $\perp$, which is required for a decision of $(\perp, 0)$.

Validity follows from the validity properties of ECHO4 and ECHO5 messages.

Binding holds since the binding property for $R = 1$ implies that there is only one possible non-$\perp$ value that can be decided in any extension after the first correct process sends its ECHO4 message.

For *Termination and time complexity*, we prove every correct process decides by time 7.

Since ECHO4 messages are sent for $R = 2$ exactly when decisions are made for $R = 1$, we know that correct processes send ECHO4 by time 5. Fortunately, the approval time of 5 for values in ECHO3 messages carries over to ECHO4 messages. Thus p either receives at least $n - f$ ECHO4 messages for a common value by time 6 or has approved either $\perp$ or at least two values by time 5, and thus sends ECHO5 by time 6.

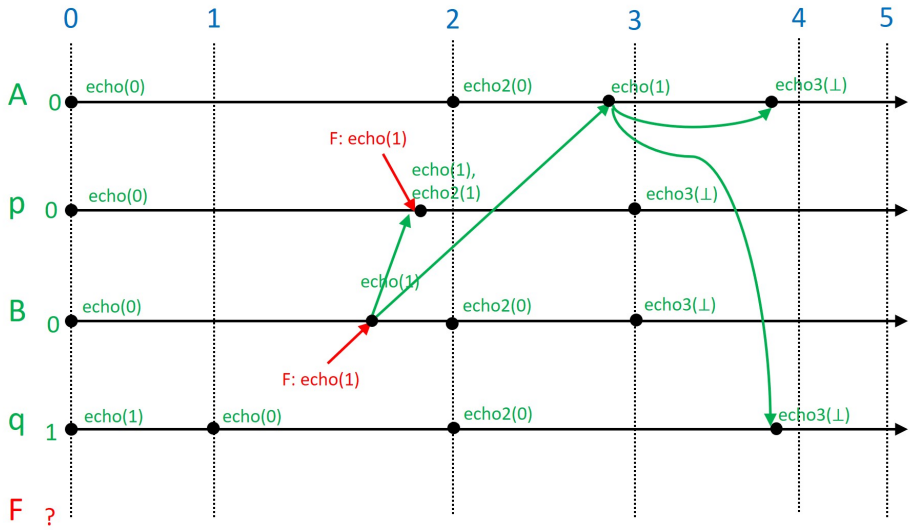

Figure 4 Diagram illustrating scenario in which decision is delayed until almost time 5.

Similarly, the approval time of 5 also holds for values in ECHO5 messages. Thus p either receives at least $n - f$ ECHO5 messages for a common value by time 7 and decides, or has approved either $\perp$ or at least two values by time 5. In the latter case, since p receives less than $n - f$ ECHO5 messages for $\perp$, it receives an ECHO5 message for some $w \in V$ from a correct process q. In turn, q received at least $n - f$ ECHO4 messages for w, at least $n - 2f \geq f + 1$ of which are from correct processes. Since these correct processes send their ECHO4 messages for w by time 5, p receives them by time 6. Thus p decides by time 7.

The *message complexity* is still $O(|V| \cdot n^2)$ since in addition to the messages sent when $R = 1$, each process sends to all processes one ECHO4 message and one ECHO5 message. ◀

Time complexity versus round complexity. The upper bounds of 5 and 7 on the *time complexities* for Algorithm 3 are tight, as shown by an execution described below; see more details in the full version. The execution uses $V = \{0, 1\}$ and thus it is also an execution of Algorithm 4 in [3], implying that the tight time complexity of the latter algorithm is also 5, and that of Algorithm 6 in [3] is 7. This is in contrast to the *round complexities* of 4 and 6 calculated in [3] for their Algorithms 4 and 6.

The discrepancy between round complexity and time complexity is caused by the waiting conditions imposed before performing the next broadcast. If the condition is simply to receive enough messages from the previous broadcast, then at most one time unit elapses per broadcast. But if there is an additional condition, for instance, waiting to approve at least two values, then the condition may take more than one time unit to become true. We next sketch the execution to illustrate this point; see Figure 4.

Assume $n = 3f + 1$, where $f \geq 2$, and partition the correct processes into sets A, B, $\{p\}$, and $\{q\}$, where $|A| = f - 1$, $|B| = f$, and let F be the set of f faulty processes. Initially all correct processes have input 0 except q has input 1. At time 0, processes send ECHO messages with their inputs which arrive at time 1. At time 1, q sends an ECHO message for 0, which arrives at time 2. Just before time 2, every process in B receives f ECHO messages for 1 from the processes in F, which causes it to send ECHO for 1. Those messages take 1 time unit to arrive at all processes except p, which receives them immediately. This then causes p to send an ECHO message for 1. Immediately thereafter, p receives f ECHO messages for 1

from the processes in F, which causes it to send its ECHO2 message, for 1. These messages from p take 1 time unit to arrive. Then at time 2, all the correct processes send ECHO2 message for 0, except for p, which has already sent its ECHO2 message (for 1). The difficulty is that by time 3, no process has $n - f$ ECHO2 messages for a common value, due to the ECHO2 message for 1 from p. The processes in $B \cup \{p\}$ are not blocked from sending ECHO3 because they have approved both 0 and 1, but the processes in $A \cup \{q\}$ have only approved 0; they are unable to approve 1 until they get the ECHO messages for 1 sent by the processes in $A \cup \{q\}$, which does not happen until just before time 4. Letting all the ECHO3 messages have delay 1 means that processes cannot decide until shortly before time 5.

6 Discussion

We have proposed a new problem called connected consensus which generalizes a number of primitives used to solve consensus, including crusader agreement, graded broadcast, and adopt-commit, using a numeric parameter R. The problem can be reduced to real-valued approximate agreement when the input set is binary and and to approximate agreement on graphs for multi-valued input sets (two or more inputs). We extended the definition of the binding property for such primitives to the multi-valued case.

We presented efficient message-passing algorithms for connected consensus when R is 1 (corresponding to crusader agreement) or 2 (corresponding to graded broadcast), in the presence of crash and malicious failures, for multi-valued input sets.

Our algorithm for crash failures has optimal resilience and message complexity; its time complexity is optimal for reasonable resiliencies and improves on the best previously known algorithms, which only handled binary inputs.

For malicious failures, we provide two algorithms that trade off resilience against time and message complexity. One algorithm has time complexity 1 or 2 (for $R = 1$ or $R = 2$) and sends $O(n^2)$ messages, but requires $n > 5f$. The other algorithm only requires $n > 3f$, but has time complexity 5 or 7 (for $R = 1$ or $R = 2$) and sends $O(|V| \cdot n^2)$ messages. This is the same performance as the algorithms in [3] which are only for the case when $|V| = 2$.

The techniques used in our (simple) algorithms for crash failures and for malicious failures with $n > 5f$ are familiar from prior work. For example, algorithms in [12, 26, 33] rely on similar mechanisms to solve (standard) consensus using various kinds of oracles. The novelty in our work is the focus on the binding property for a key subproblem of consensus, extracted as connected consensus.

There is a message-passing algorithm for adopt-commit with multi-valued inputs that works in the presence of malicious failures as long as $n > 3f$ [11]. However, the number of possible inputs must be smaller than $\lfloor \frac{n-(f+1)}{f} \rfloor$, while our algorithm works for any size input set. The message complexity is $O(n^3)$ as compared to our $O(|V| \cdot n^2)$. Furthermore, this algorithm avoids the challenge of ensuring the binding property (which anyway is not well-defined for adopt-commit) as it is combined with an oracle in order to solve consensus.

Concurrent work by Abraham, Ben-David, Stern and Yandamuri appearing in these proceedings [1] studies the round complexity of binary (graded) crusader agreement both with and without binding. A key difference is in the definition of the adversary: In [1] it is assumed that the adversary can adaptively choose the inputs of the processes after the start of the execution when the processes take their first steps. In contrast, in our model, the adversary does not have that power, and the inputs are fixed at the beginning of each execution. As a result, some of the lower bounds in [1] are larger than some of our upper bounds. It is argued in [1] that tolerating such a strong adversary can be advantageous for developing simple and efficient randomized consensus algorithms.

An intriguing open question is whether there is an inherent cost for satisfying the binding property: is there some measure, perhaps time, in which solving connected consensus without binding is more efficient than solving it with binding?

Adopt-commit and related primitives have been implemented also in shared-memory systems, e.g., [6, 18, 27, 32]. The connection we have made between connected consensus and approximate agreement (on graphs) may contribute to finding improved algorithms for these primitives in shared memory.

This connection might also be a fruitful direction for future work on connected consensus in other timing models. For instance, [5] uses adopt-commit (and variants) to solve consensus in eventually synchronous systems. Another interesting direction is to study connected consensus in other fault models, such as the authenticated setting; authenticated algorithms appear in [3] but they are for binary inputs.

References

1 I. Abraham, N. Ben-David, G. Stern, and S. Yandamuri. On the round complexity of asynchronous crusader agreement. In *OPODIS*, 2023.

2 Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, pages 229–239. Springer, 2004. doi:10.1007/11516798_17.

3 Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In *41st ACM Symposium on Principles of Distributed Computing*, pages 381–391, 2022. doi:10.1145/3519270.3538426.

4 Yehuda Afek, James Aspnes, Edo Cohen, and Danny Vainstein. Brief announcement: Object oriented consensus. In *36th ACM Symposium on Principles of Distributed Computing*, pages 367–369, 2017. Full version in https://www.cs.yale.edu/homes/aspnes/papers/vac-abstract.html. doi:10.1145/3087801.3087867.

5 Karolos Antoniadis, Julien Benhaim, Antoine Desjardins, Elias Poroma, Vincent Gramoli, Rachid Guerraoui, Gauthier Voron, and Igor Zablotchi. Leaderless consensus. *Journal of Parallel and Distributed Computing*, 176:95–113, 2023. doi:10.1016/J.JPDC.2023.01.009.

6 James Aspnes. Faster randomized consensus with an oblivious adversary. In *31st ACM Symposium on Principles of Distributed Computing*, pages 1–8, 2012. doi:10.1145/2332432.2332434.

7 Hagit Attiya, Constantin Enea, and Shafik Nassar. Faithful simulation of randomized BFT protocols on block DAGs. In *Concur*, 2023. URL: https://eprint.iacr.org/2023/192.

8 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. McGraw-Hill Publishing Company, 1st edition, 1998.

9 Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16:249–262, 2003. doi:10.1007/S00446-002-0083-3.

10 Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous Byzantine agreement with subquadratic communication. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 353–380. Springer, 2020. doi:10.1007/978-3-030-64375-1_13.

11 Zohir Bouzid, Achour Mostefaoui, and Michel Raynal. Minimal synchrony for Byzantine consensus. In *34th ACM Symposium on Principles of Distributed Computing*, pages 461–470, 2015. doi:10.1145/2767386.2767418.

12 Francisco Brasileiro, Fabíola Greve, Achour Mostéfaoui, and Michel Raynal. Consensus in one communication step. In *6th International Conference on Parallel Computing Technologies*, volume 2127, pages 42–50, 2001. doi:10.1007/3-540-44743-1_4.

13 Armando Castañeda, Sergio Rajsbaum, and Matthieu Roy. Convergence and covering on graphs for wait-free robots. *Journal of the Brazilian Computer Society*, 24:1–15, 2018. doi:10.1186/S13173-017-0065-8.

14 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996. doi:10.1145/226643.226647.

15 Ran Cohen, Pouyan Forghani, Juan Garay, Rutvik Patel, and Vassilis Zikas. Concurrent asynchronous Byzantine agreement in expected-constant rounds, revisited. Cryptology ePrint Archive, Paper 2023/1003, 2023. URL: https://eprint.iacr.org/2023/1003.

16 Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006. doi:10.1093/COMJNL/BXH145.

17 Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient Byzantine agreement and multi-party computation with asynchronous fallback. In *19th International Conference on Theory of Cryptography, TCC*, pages 623–653, 2021. doi:10.1007/978-3-030-90459-3_21.

18 Carole Delporte-Gallet, Hugues Fauconnier, and Michel Raynal. On the weakest information on failures to solve mutual exclusion and consensus in asynchronous crash-prone read/write systems. *Journal of Parallel and Distributed Computing*, 153:110–118, 2021. doi:10.1016/J.JPDC.2021.03.015.

19 Danny Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982. doi:10.1016/0196-6774(82)90004-9.

20 Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, 1986. doi:10.1145/5925.5931.

21 Alan David Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4:9–29, 1990. doi:10.1007/BF01783662.

22 Alan David Fekete. Asynchronous approximate agreement. *Information and Computation*, 115(1):95–124, 1994. doi:10.1006/INCO.1994.1094.

23 Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *12th Annual ACM Symposium on Theory of Computing*, pages 148–161, 1988. doi:10.1145/62212.62225.

24 Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997. doi:10.1137/S0097539790187084.

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

26 Roy Friedman, Achour Mostéfaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Trans. Dependable Secur. Comput.*, 2(1):46–56, 2005. doi:10.1109/TDSC.2005.13.

27 Eli Gafni. Round-by-round fault detectors: unifying synchrony and asynchrony. In *17th ACM Symposium on Principles of Distributed Computing*, pages 143–152, 1998. doi:10.1145/277697.277724.

28 Manfred Koebe. On a new class of intersection graphs. In *Annals of Discrete Mathematics*, volume 51, pages 141–143. Elsevier, 1992. doi:10.1016/S0167-5060(08)70618-6.

29 Stephen R Mahaney and Fred B Schneider. Inexact agreement: Accuracy, precision, and graceful degradation. In *4th ACM Symposium on Principles of Distributed Computing*, pages 237–249, 1985. doi:10.1145/323596.323618.

30 Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2295–2308, 2022. doi:10.1145/3548606.3559347.

31 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, 2015. doi:10.1145/2785953.

32 Achour Mostefaoui, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. The combined power of conditions and information on failures to solve asynchronous set agreement. *SIAM J. on Computing*, 38(4):1574–1601, 2008. doi:10.1137/050645580.

33 Achour Mostéfaoui and Michel Raynal. Leader-based consensus. *Parallel Process. Lett.*, 11(1):95–107, 2001. doi:10.1142/S0129626401000452.

34 Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with $t < n/3$, $O(n^2)$ messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017. doi:10.1007/s00236-016-0269-y.

35 Thomas Nowak and Joel Rybicki. Byzantine approximate agreement on graphs. In *33rd International Symposium on Distributed Computing*, pages 29:1–29:17, 2019. doi:10.4230/LIPICS.DISC.2019.29.

36 Sam Toueg. Randomized Byzantine agreements. In *3rd ACM Symposium on Principles of Distributed Computing*, pages 163–178, 1984. doi:10.1145/800222.806744.

37 Jiong Yang, Gil Neiger, and Eli Gafni. Structured derivations of consensus algorithms for failure detectors. In *17th ACM Symposium on Principles of Distributed Computing*, pages 297–306, 1998. doi:10.1145/277697.277755.

A No Built-in Binding, for $n < 5f$

In this section we show that the locked value for the Binding property cannot be predetermined from the correct processes' inputs if $n \leq 5f$, even if faulty processes cannot equivocate and the input set is binary. Our approach is to consider any algorithm that works by having processes obtain $n - f$ values, at most one from each process, such that if two processes p and q both get values corresponding to process r, then the values are the same and if r is correct then that value is r's input. Then the algorithm must decide based only on the multiset of values it has received. We call such an algorithm *uniform*.

Consider a uniform algorithm with $n \leq 5f$. We show that if a process receives all but f 0's, then it must decide 0 and if it receives all but f 1's, then it must decide 1. Then we show that if a process receives about half 0's and half 1's, then it must decide $(\perp, 0)$ (this relies on the fact that two configurations with different non-$\perp$ values must have more than f values that are different). Finally we rely on the fact that the number of values a process receives is at most $4f$ to show that if the first process to decide gets about half 0's and half 1's and decides $(\perp, 0)$, then in one extension we can replace f of the 0's with 1's to get a configuration that requires a decision of $(1, 1)$, and in another extension we can replace f of the 1's with 0's to get a configuration that requires a decision of $(0, 1)$, which violates Binding.

▶ **Theorem 6.** *If a uniform connected consensus algorithm for $R = 1$ with n processes, up to f of which can be malicious, satifies the Binding property, then $n > 5f$.*

Proof. Consider for contradiction such an algorithm with $n \leq 5f$. Without loss of generality, assume $V = \{0, 1\}$. For simplicity, we refer to the possible decisions as 0, 1, and $\perp$, instead of $(0, 1)$, $(1, 1)$, and $(\perp, 0)$. Let $D(z) \in \{0, 1, \perp\}$ denote the decision made when the multiset of $n - f$ values received has z 0's.

We first show that if there is an "overwhelming" number of 0's received, then the decision must be 0, and similarly for 1. The threshold is $n - 2f$, which is at least $f + 1$ since the resilience lower bound discussed in Section 5.1 shows that n must be at least $3f + 1$.

▶ **Lemma 7.** *Let z be any integer in $\{0, \ldots, n - 2f\}$.*
(a) *If $z \geq n - 2f$, then $D(z) = 0$.*
(b) *If $z \leq f$, then $D(z) = 1$.*

Proof.
(a) Consider any execution in which correct process p receives $z \geq n-2f$ 0's and $n-f-z$ 1's. This execution is indistinguishable from one in which all $n - f$ of the correct processes have input 0, and p receives $z \geq n - 2f$ messages for 0 from the correct processes and $n - f - z \leq f$ messages for 1 from faulty processes. By the Validity condition, p must decide 0 in the second execution. Thus $D(z) = 0$.

(b) Consider any execution in which correct process p receives $z \leq f$ 0's and $n - f - z$ 1's. This execution is indistinguishable from one in which all $n - f$ of the correct processes have input 1, and p receives $n - f - z$ messages for 1 from the correct processes and $z \leq f$ messages for 0 from faulty processes. By the Validity condition, p must decide 1 in the second execution. Thus $D(z) = 1$. ◀

▶ **Lemma 8.** *The number of processes n must be at least $3f + 2$.*

Proof. Suppose in contradiction that $n = 3f + 1$ and consider any correct process p. If the majority of the $n - f = 2f + 1$ values received by p is 0, then $z \geq f + 1$. Since $n = 3f + 1$, $f + 1 = n - 2f$, and thus Lemma 7(a) implies that p must decide 0. If the majority value is not 0, then the majority value must be 1, and Lemma 7(b) implies that p must decide 1. Thus there is no possibility of a process deciding $\perp$, and hence the algorithm actually solves consensus, which is impossible [25]. ◀

The next lemma shows that the range of input values requiring a decision of 0 and the range of input values requiring a decision of 1 must be sufficiently separated from each other.

▶ **Lemma 9.** *Let x and y be integers in $\{0, \ldots, n - 2f\}$. If $D(x) = 0$ and $D(y) = 1$, then $|x - y| > f$.*

Proof. Suppose in contradiction there exist x and y in $\{0, \ldots, n - 2f\}$ such that $D(x) = 0$, $D(y) = 1$, and $|x - y| \leq f$.

Without loss of generality, suppose $x > y$. Consider the execution in which the correct inputs are x 0's and $n - f - x$ 1's. Suppose that correct process p hears from all the correct processes, so it gets x 0's and $n - f - x$ 1's, and decides 0. Suppose that another correct process q hears from only y of the correct processes with input 0, all $n - f - x$ of the correct processes with input 1, and $x - y \leq f$ faulty processes, who pretend to have input 1. Thus q gets y 0's and $(n - f - x) + (x - y) = n - f - y$ 1's, and decides 1. This is possible because of the asynchrony of the message deliveries. But this violates the Agreement property. ◀

The next claim states that when the received values are about half 0's and half 1's, the decision must be $\perp$. It also shows that this situation is not that far from a situation requiring a decision of 0 and also not that far from a situation requiring a decision of 1.

▷ Claim 10. Let $m = \left\lceil \frac{n-f}{2} \right\rceil$.
(a) $m + f \geq n - 2f$,
(b) $m - f \leq f$, and
(c) $D(m) = \perp$.

We show that Binding is not guaranteed.

Consider an execution α in which $m = \left\lceil \frac{n-f}{2} \right\rceil$ of the correct processes have input 0 and the remaining $n - f - m$ correct processes have input 1. Let correct process p be the first to receive $n - f$ messages, which are all from the correct processes. So p gets m 0's and $n - f - m$ 1's. By Claim 10(c), it decides $\perp$.

Also suppose that all messages to another correct process q are delayed until after p decides and that the faulty processes do nothing until after p decides.

Let α_0 be an extension of α in which q receives m 0's from correct processes, $n - 2f - m$ 1's from correct processes, and f 0's from faulty processes. So q receives $m + f$ 0's and $n - 2f - m$ 1's. Since $m + f \geq n - 2f$ by Claim 10(a), Lemma 7(a) implies that q decides 0.

Let α_1 be an extension of α in which q receives $m-f$ 0's from correct processes, $n-f-m$ 1's from correct processes, and f 1's from faulty processes. So q receives $m-f$ 0's and $n-m$ 1's. Since $m-f \leq f$ by Claim 10(b), Lemma 7(b) implies that q decides 1.

The existence of extensions α_1 and α_0 violates the Binding property. ◀