Error-Free Multi-Valued Consensus with Byzantine Failures

Guanfeng Liang
Dept. of Electrical and Computer Engineering,
and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
gliang2@illinois.edu

Nitin Vaidya
Dept. of Electrical and Computer Engineering,
and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
nhv@illinois.edu

ABSTRACT

In this paper, we present an efficient *deterministic* algorithm for consensus in presence of Byzantine failures. Our algorithm achieves consensus on an L-bit value with communication complexity $O(nL + n^4 L^{0.5} + n^6)$ bits, in a network consisting of n processors with up to t Byzantine failures, such that $t < n/3$. For large enough L, communication complexity of the proposed algorithm becomes $O(nL)$ bits, linear in the number of processors. To achieve this goal, the algorithm performs consensus on a long message (L bits), in multiple *generations*, each generation performing consensus on a part of the input message. The failure-free execution of each generation is made efficient by using a combination of two techniques: error detection coding, and processor clique formation based on matching input values proposed by the processors. By keeping track of faulty behavior over the different generations, the algorithm can ensure that most generations of the algorithm are failure-free. With parameterization, our algorithm is able to achieve a large class of validity conditions for consensus, while maintaining linear communication complexity. With a suitable choice of the error detection code, and using a clique of an appropriate size, the communication cost can be traded off with the strength of the validity condition. The proposed algorithm requires no cryptographic techniques.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity—*Nonnumerical Algorithms and Problems*

General Terms

Theory

Keywords

Byzantine agreement, consensus, distributed computing

1. INTRODUCTION

The Byzantine consensus problem considers n processors, namely $P_1, ..., P_n$, of which at most t processors may be *faulty* and deviate from the algorithm in arbitrary fashion. Each processor P_i is given an L-bit input value v_i. The basic version of the consensus problem considered here requires that the following properties to be satisfied.

- *Termination*: every fault-free P_i eventually decides on an output value v'_i.
- *Consistency*: the output values of all fault-free processors are equal, i.e., for every fault-free processor P_i, $v'_i = v'$ for some v'.
- *Validity*: if every fault-free P_i holds the same input $v_i = v$ for some v, then $v' = v$.

These properties have been used as the requirements for consensus in previous literature as well [9]. Other characterizations of the *validity* condition above are also of potential interest in practice. For instance, we may want the processors to agree on a value v if at least $\lceil \frac{n+1}{2} \rceil$ processors have input value equal to v. With suitable parameterization, our algorithm satisfies such more general validity conditions as well (Section 4). Algorithms that satisfy the desired properties in *all* executions are said to be *error-free.* We are interested in the communication complexity of error-free consensus algorithms. *Communication complexity* of an algorithm is defined as the maximum (over all permissible executions) of the total number of bits transmitted by all the processors according to the specification of the algorithm. This measure of complexity was first introduced by Yao [24], and has been used widely (e.g., [8, 9, 21]).

System Model: We assume a synchronous fully connected network of n processors. Every pair of processors is connected by a pair of directed point-to-point communication channels. Each processor correctly knows the identity of the processors at the other end of its channels. Whenever a processor receives a message on such a directed channel, it can correctly assume that the message is sent by the processor at the other end of the channel. We assume a Byzantine adversary that has complete knowledge of the state of the processors, including the L-bit input values. No secret is hidden from the adversary. The adversary can take over up to t processors ($t < n/3$) at any point during the algorithm. These processors are said to be *faulty.* The faulty processors can engage in any "*misbehavior*", i.e., deviations from the algorithm, including collusion. The remaining processors are *fault-free* and follow the algorithm.

It has been shown that error-free consensus is impossible if $t \geq n/3$ [20]. $\Omega(n^2)$ has been shown to be a lower bound on the number of messages needed to achieve error-free consensus [7]. Since any message must be of at least 1 bit, this gives a lower bound of $\Omega(n^2)$ bits on the communication complexity of any binary (1-bit) consensus algorithm.

The problem of achieving consensus on a single L-bit value may be solved using L instances of a 1-bit consensus algorithm. However, this approach will result in communication complexity of $\Omega(n^2L)$, since $\Omega(n^2)$ is a lower bound on communication complexity of 1-bit consensus. Fitzi and Hirt [9] proposed a probabilistically correct multi-valued consensus algorithm which improves the communication complexity to $O(nL)$ for sufficiently large L, at the cost of allowing a non-zero probability of error. Since $\Omega(nL)$ is a lower bound on the communication complexity of consensus on an L-bit value [9], this algorithm has optimal complexity. We present a deterministic error-free consensus algorithm with communication complexity of $O(nL)$ bits for sufficiently large L. For smaller L, the communication complexity of our algorithms is $O(nL + n^4L^{0.5} + n^6)$. The proposed algorithm is also able to satisfy more general *validity* conditions (with parameterization), while still achieving communication complexity linear in n.

2. ALGORITHM OVERVIEW

The goal of the proposed consensus algorithm is to achieve consensus on an L-bit value (or message). The algorithm is designed to perform efficiently for large L. Consequently, our discussion will assume that L is "sufficiently large" (how large is "sufficiently large" will become clearer later in the paper). We now briefly describe the salient features of the consensus algorithm, with the detailed algorithm presented later in Sections 3 and 4.

2.1 Execution in Multiple Generations

To improve the communication complexity, consensus on the L-bit value is performed "in parts". In particular, for a certain integer D, the L-bit value is divided into L/D parts, each consisting of D bits. For convenience of presentation, we will assume that L/D is an integer. A sub-algorithm is used to perform consensus on each of these D-bit values, and we will refer to each execution of the sub-algorithm as a "generation".

2.2 Memory Across Generations

If during any one generation, misbehavior by some faulty processor is detected, then additional (and expensive) diagnostic steps are performed to gain information on the potential identity of the misbehaving processor(s). This information is captured by means of a *diagnosis graph*, as elaborated later. As the sub-algorithm is performed for each new generation, the *diagnosis graph* is updated to incorporate any new information that may be learned regarding the location of the faulty processors. The execution of the sub-algorithm in each generation is adapted to the state of the diagnosis graph at the start of the generation.

2.3 Bounded Instances of Misbehavior

With Byzantine failures, it is not always possible to immediately determine the identity of a misbehaving processor. However, due to the manner in which the diagnosis graph is maintained, and the manner in which the sub-algorithm adapts to the diagnosis graph, the t (or fewer) faulty processors can collectively misbehave in at most $t(t+1)$ generations, before all the faulty processors are exactly identified. Once a faulty processor is identified, it is effectively isolated from the network, and cannot tamper with future generations. Thus, $t(t+1)$ is also an upper bound on the number of generations in which the expensive diagnostic steps referred above may need to be performed.

2.4 Low-Cost Failure-Free Execution

Due to the bounded number of generations in which the faulty processors can misbehave, it turns out that the faulty processors do not tamper with the execution in a majority of the generations. We use a low-cost mechanism to achieve consensus in failure-free generations, which helps to achieve low communication complexity. In particular, we use an *error detection code*-based strategy to reduce the amount of information the processors must exchange to be able to achieve consensus in the absence of any misbehavior (the strategy, in fact, also allows detection of potential misbehavior). The error detection code is used to efficiently identify a a large enough *clique* of processors that "propose" an identical value. Clique formation helps reduce communication cost by ensuring that the non-clique processors, whose input value is not going to correspond to the final consensus, do not interfere with the correct consensus. The algorithm can satisfy a large range of *validity* conditions, without changing the algorithm structure, simply by changing the level of redundancy in the error detection code used by the algorithm, while still maintaining low failure-free overhead.

2.5 Consistent Diagnosis Graph Maintenance

A copy of the diagnosis graph is maintained locally by each fault-free processor. To ensure consistent maintenance of this graph, the *diagnostic information* (elaborated later) needs to be distributed consistently to all the processors in the network. This operation is performed using an error-free 1-bit Byzantine broadcast algorithm that tolerates $t < n/3$ Byzantine failures with communication complexity of $O(n^2)$ bits [6, 3]. This 1-bit broadcast algorithm is referred as *Broadcast_1_Bit* in our discussion. While *Broadcast_1_Bit* is expensive, its cumulative overhead is kept low by invoking it a relatively small number of times.

The structure above is inspired by prior work on fault-tolerant computing and communications theory, which is discussed in Section 6. It turns out that the "dispute control" approach used in the work on multi-party computation (MPC) has also used this structure [1], and its variation called *player elimination* has been used to design an error-free linear complexity Byzantine *broadcast* algorithm [2].

We now elaborate on the error detection code used in our algorithm, and also describe the *diagnosis graph* in some more detail.

2.6 Error Detection Code

We will use Reed-Solomon codes in our algorithm (potentially, other codes may be used instead). Consider a (m, d) Reed-Solomon code in Galois Field GF(2^c), where c is chosen large enough (specifically, $m \leq 2^c - 1$). This code encodes d data symbols from GF(2^c) into a codeword consisting of m symbols from GF(2^c). Each symbol from GF(2^c) can be represented using c bits. Thus, a data vector of d symbols contains dc bits, and the corresponding codeword contains mc bits. Each symbol of the codeword is computed as a linear combination of the d data symbols, such that every subset of d coded symbols represents a set of linearly independent combinations of the d data symbols. This property implies that any subset of d symbols from the m symbols of a given codeword can be used to determine the corresponding data vector. Similarly, knowledge of any subset of d symbols from a codeword suffices to determine the remaining symbols of the codeword. So d is also called the *dimension* of the code. The (m, d) code has the Hamming distance of $m - d + 1$, and can always detect up to $m - d$ errors. We will denote a code with dimension d as C_d, and the encoding/decoding operations as $Z = C_d(v)$ and $v = C_d^{-1}(Z)$ for a data vector v and the corresponding codeword Z.

In our algorithm, we also assume the availability of a null ($\perp$) symbol that is distinguished from all other symbols. For an m-element vector V, we denote $V[j]$ as the j-th element of the vector, $1 \le j \le m$. Given a subset $A \subseteq \{1, \ldots, m\}$, denote $V|A$ as the ordered list of elements of V at the locations corresponding to elements of A. For instance, if $m = 5$ and $A = \{2, 4\}$, then $V|A$ is equal to $(V[2], V[4])$. We will say that $V|A \in C_d$ if $V|A$ contains at least d non-null ($\neq\perp$) elements, and there exists a codeword $Z = C_d(v)$ for some v such that the non-null elements of $V|A$ are equal to the corresponding elements of $Z|A$. When such Z and v exist, by generalizing the decoding operation, we define $C_d^{-1}(V|A) = v$. If no such Z and v exist, we will say that $V|A \notin C_d$.

For our algorithm, we use a $(n, q-t)$ distance-$(n-q+t+1)$ code C_{q-t}, for a suitable q such that $t+1 \le q \le n-t$, and ensure that there are at least $q - t$ non-null elements in the argument to C_d^{-1} (when the decoding function is used at a fault-free processor).

2.7 Diagnosis Graph

The fault-free processors' (potentially partial) knowledge of the identity of the faulty processors is captured by a diagnosis graph. A diagnosis graph is an undirected graph with n vertices, with vertex i corresponding to processor P_i. A pair of processors are said to "trust" each other if the corresponding pairs of vertices in the diagnosis graph is connected with an edge; otherwise they are said to "accuse" each other.

Before the start of the very first generation, the diagnosis graph is initialized as a fully connected graph, which implies that all the n processors initially trust each other. During the execution of the algorithm, whenever misbehavior by some faulty processor is detected, the diagnosis graph will be updated, and one or more edges will be removed from the graph, using the diagnostic information communicated using the *Broadcast_1_Bit* algorithm. The use of *Broadcast_1_Bit* ensures that the fault-free processors always have a consistent view of the diagnosis graph. The evolution of the diagnosis graph satisfies the following properties:

- If an edge is removed from the diagnosis graph, at least one of the processors corresponding to the two endpoints of the removed edge must be faulty.
- The fault-free processors always trust each other.
- If more than t edges at a vertex in the diagnosis graph are removed, then the processor corresponding to that vertex must be faulty.

The last two properties above follow from the first property, and the assumption that at most t processor are faulty.

3. MULTI-VALUED CONSENSUS

In this section, we describe a consensus algorithm that satisfies the properties listed in Section 1, and present a proof of correctness. Algorithm parameterization to satisfy more general *validity* requirements will be discussed in Section 4.

The L-bit input value v_i at each processor is divided into L/D parts of size D bits each, as noted earlier. These parts are denoted as $v_i(1), v_i(2), \cdots, v_i(L/D)$. The algorithm for achieving L-bit consensus consists of L/D sequential executions of Algorithm 1 presented in this section. Algorithm 1 is executed once for each generation. For the g-th generation ($1 \le g \le L/D$), each processor P_i uses $v_i(g)$ as its input in Algorithm 1. Each generation of the algorithm results in processor P_i deciding on g-th part (namely, $v_i'(g)$) of its final decision value v_i'.

The value $v_i(g)$ is represented by a vector of $n - 2t$ symbols, each symbol represented with $D/(n - 2t)$ bits. For convenience of presentation, assume that $D/(n - 2t)$ is an integer. We will refer to these $n - 2t$ symbols as the *data symbols.*

An $(n, n - 2t)$ distance-$(2t + 1)$ Reed-Solomon code, denoted as C_{n-2t}, is used to encode the $n-2t$ data symbols into n coded symbols. We assume that $D/(n-2t)$ is large enough to allow the above Reed-Solomon code to exist, specifically, $n \le 2^{D/(n-2t)} - 1$, which implies that $D = \Omega(n \log n)$. This condition is met only if L is large enough (since $L > D$). As we will see later, C_{n-2t} is used only for error detection. To detect t faults, a more efficient code of dimension $n - t$ suffices. We will discuss this improvement in Section 7.

Let the set of all the fault-free processors be denoted as P_{good}. Algorithm 1 for each generation g consists of three stages. We summarize the function of these three stages first, followed by a more detailed discussion:

1. Matching stage: Each processor P_i encodes its D-bit input $v_i(g)$ for generation g into n coded symbols, as noted above. Each processor P_i sends one of these n coded symbols to the other processors *that it trusts.* Processor P_i trusts processor P_j if and only if the corresponding vertices in the diagnosis graph are connected by an edge. Using the symbols thus received from each other, the processors attempt to identify a "matching set" of processors (denoted P_{match}) of size $n - t$ such that the fault-free processors in P_{match} are guaranteed to have an identical input value for the current generation. If such a P_{match} is not found, it can be determined with certainty that all the fault-free processors do not have the same input value – in this case, the fault-free processors decide on a default output value and terminate the algorithm.

2. Checking stage: If a set of processors P_{match} is identified in the above matching stage, each processor $P_j \notin P_{match}$ checks whether the symbols received from processors in P_{match} correspond to a valid codeword. If such a codeword exists, then the symbols received from P_{match} are said to be "consistent". If any processor finds that these symbols are not consistent, then misbehavior by some faulty processor is detected. Else all the processors are able to correctly compute the value to be agreed upon in the current generation.

3. Diagnosis stage: Whenever misbehavior is detected, the diagnosis stage is performed, to learn (possibly partial) information regarding the identity of the faulty processor(s). For fault diagnosis, the processors in P_{match} are required to *broadcast* the coded symbol they sent in the matching stage, using the *Broadcast_1_Bit* algorithm. Using the information received during these broadcasts, the fault-free processors are able to learn new information regarding the potential identity of the faulty processor(s). The *diagnosis graph* (called *Diag_Graph* in Algorithm 1) is updated to incorporate this new information.

In the rest of this section, we discuss each of the three stages in more detail. Note that whenever algorithm *Broadcast_1_Bit* is used, all the fault-free processors will receive the broadcast information identically. One instance of *Broadcast_1_Bit* is needed for each bit of information broadcast using *Broadcast_1_Bit* .

3.1 Matching Stage

The line numbers referred below correspond to the line numbers for the pseudo-code in Algorithm 1.

`Line 1(a):` In generation g, each processor P_i first encodes $v_i(g)$, represented by $n-2t$ symbols, into a codeword S_i from the code C_{n-2t}. The j-th symbol in the codeword is denoted as $S_i[j]$. Then processor P_i sends $S_i[i]$, the i-th symbol of its codeword, to all the other processors *that it trusts*. Recall that P_i trusts P_j if and only if there is an edge between the corresponding vertices in the diagnosis graph (referred as *Diag_Graph* in the pseudo-code).

`Line 1(b):` Let us denote by $R_i[j]$ the symbol that P_i receives from a trusted processor P_j. If a processor P_i does not trust some processor P_j, then P_i sets $R_i[j]$ equal to null ($\perp$). Messages received from untrusted processors are ignored.

`Line 1(c):` Flag $M_i[j]$ is used to record whether processor P_i finds processor P_j's symbol consistent with its own local value. Specifically, the pseudo-code in Line 1(c) is equivalent to the following:

- When P_i trusts P_j:
  If $R_i[j] = S_i[j]$, then $M_i[j] =$ **true** ;
  else $M_i[j] =$ **false** .
- When P_i does not trust P_j: $M_i[j] =$ **false** .

`Line 1(d):` As we will see later, if a fault-free processor P_i does not trust another processor, then the other processor must be faulty. Thus entry $M_i[j]$ in vector M_i is **false** if (i) P_i believes that processor P_j is faulty, or (ii) the input value at processor P_j appears to differ from the input value at P_i. Thus, entry $M_i[j]$ being **true** implies that, as of this time, P_i believes that P_j is fault-free, and that the value at P_j is possibly identical to the value at P_i. Processor P_i uses *Broadcast_1_Bit* to broadcast M_i to all the processors. One instance of *Broadcast_1_Bit* is needed for each bit of M_i.

`Lines 1(e) and 1(f):` Due to the use of *Broadcast_1_Bit* , all fault-free processors receive identical vector M_j from each processor P_j. Using these M vectors, each processor P_i attempts to find a set P_{match} containing $n-t$ processors such that, for every pair $P_j, P_k \in P_{match}$, $M_j[k] = M_k[j] =$ **true** . Since the M vectors are received identically by all the fault-free processors (using *Broadcast_1_Bit*), they can compute identical P_{match}. However, if such a set P_{match} does not exist, then the fault-free processors conclude that all the fault-free processors do not have identical input – in this case, they decide on a default value, and terminate the algorithm.

It is worth noting that finding P_{match} is, in fact, equivalent to identifying a clique of size $n-t$ in an undirected graph of size n, whose edges are defined by the M vectors. Specifically, an edge exists between j and k in this graph, if $M_j[k] = M_k[j] =$ **true** . Finding a clique of a certain size in general graphs is NP-Complete. It turns out that, with a slight modification, the algorithm can perform correctly even if P_{match} is not a clique in the graph induced by M vectors. Instead it suffices if P_{match} includes at least $n-2t$ fault-free processors with identical input for the current generation. In other words, the subgraph induced by M and P_{match} will contain a clique of size $n-2t$ corresponding to fault-free processors. Such a P_{match} can be found with polynomial computational complexity (please refer the Appendix). However, for simplicity, in our proofs, we will assume that P_{match} indeed corresponds to a clique of size $n-t$.

In the following discussion, we will show the correctness of the *Matching Stage*. In the proofs of Lemmas 1, 2, and 3, we assume that the fault-free processors (that is, the processors in set P_{good}) always trust each other – this assumption will be shown to be correct later in Lemma 4.

LEMMA 1. *If for each fault-free processor $P_i \in P_{good}$, $v_i(g) = v(g)$, for some value $v(g)$, then a set P_{match} necessarily exists (assuming that the fault-free processors trust each other).*

PROOF. Since all the fault-free processors have identical input $v(g)$ in generation g, $S_i = C_{n-2t}(v(g))$ for all $P_i \in P_{good}$. Since these processors are fault-free, and trust each other, they send each other correct messages in the matching stage. Thus, $R_i[j] = S_j[j] = S_i[j]$ for all $P_i, P_j \in P_{good}$. This fact implies that $M_i[j] =$ **true** for all $P_i, P_j \in P_{good}$. Since there are at least $n-t$ fault-free processors, it follows that a set P_{match} of size $n-t$ must exist. $\square$

Observe that, although the above proof shows that there exists a set P_{match} containing only fault-free processors, there may also be other such sets that contain some faulty processors as well. That is, all the processors in P_{match} cannot be assumed to be fault-free. The converse of Lemma 1 implies that, if a set P_{match} does not exist, it is certain that all the fault-free processors do not have the same input values. In this case, they can correctly agree on a default value and terminate the algorithm. Thus Line 1(f) is correct.

In the case when a set P_{match} is found, the following lemma is useful.

LEMMA 2. *All processors in $P_{match} \cap P_{good}$ have identical input in generation g.*

PROOF. $|P_{match} \cap P_{good}| \geq n-2t$ because $|P_{match}| = n-t$ and there are at most t faulty processors. Consider any two processors $P_i, P_j \in P_{match} \cap P_{good}$. Since $M_i[j] = M_j[i] =$ **true** , it follows that $S_i[i] = S_j[i]$ and $S_j[j] = S_i[j]$. Since there are $n-2t$ fault-free processors in $P_{match} \cap P_{good}$, this implies that the codewords computed by these fault-free processors (in Line 1(a)) contain at least $n-2t$ identical symbols. Since the code C_{n-2t} has dimension $(n-2t)$, this implies that the fault-free processors in $P_{match} \cap P_{good}$ must have identical input in generation g. $\square$

Algorithm 1 Multi-Valued Consensus (generation g)

```
1. Matching Stage:
   Each processor $P_i$ performs the matching stage as follows:
   (a) Compute $(S_i[1], \ldots, S_i[n]) = C_{n-2t}(v_i(g))$, and send $S_i[i]$ to every trusted processor $P_j$
   (b) $R_i[j] \leftarrow \begin{cases} \text{symbol that } P_i \text{ receives from } P_j \text{, if } P_i \text{ trusts } P_j; \\ \perp, \text{otherwise} \end{cases}$
   (c) If $S_i[j] = R_i[j]$ then $M_i[j] \leftarrow$ true ; else $M_i[j] \leftarrow$ false
   (d) $P_i$ broadcasts the vector $M_i$ using Broadcast_1_Bit
   Using the received $M$ vectors:
   (e) Find a set of processors $P_{match}$ of size $n-t$ such that
           $M_j[k] = M_k[j] =$ true for every pair of $P_j, P_k \in P_{match}$
   (f) If $P_{match}$ does not exist, then decide on a default value and terminate;
       else enter the Checking Stage
2. Checking Stage:
   Each processor $P_j \notin P_{match}$ performs steps 2(a) and 2(b):
   (a) If $R_j|P_{match} \in C_{n-2t}$ then $Detected_j \leftarrow$ false ; else $Detected_j \leftarrow$ true .
   (b) Broadcast $Detected_j$ using Broadcast_1_Bit
   Each processor $P_i$ performs step 2(c):
   (c) Receive $Detected_j$ from each processor $P_j \notin P_{match}$ (broadcast in step 2(b)).
       If $Detected_j =$ false for all $P_j \notin P_{match}$, then decide on $v'_i(g) = C^{-1}_{n-2t}(R_i|P_{match})$;
       else enter Diagnosis Stage
3. Diagnosis Stage:
   Each processor $P_j \in P_{match}$ performs step 3(a):
   (a) Broadcast $S_j[j]$ using Broadcast_1_Bit
   Each processor $P_i$ performs the following steps:
   (b) $R^{\#}[j] \leftarrow$ symbol received from $P_j \in P_{match}$ as a result of broadcast in step 3(a)
   (c) For all $P_j \in P_{match}$,
           if $P_i$ trusts $P_j$ and $R_i[j] = R^{\#}[j]$ then $Trust_i[j] \leftarrow$ true ;
           else $Trust_i[j] \leftarrow$ false
   (d) Broadcast $Trust_i|P_{match}$ using Broadcast_1_Bit
   (e) For each edge $(j,k)$ in Diag_Graph , such that $P_j \in P_{match}$
           remove edge $(j,k)$ if $Trust_j[k] =$ false or $Trust_k[j] =$ false
   (f) If $R^{\#}|P_{match} \in C_{n-2t}$ then
           if for any $P_j \notin P_{match}$,
               $Detected_j =$ true , but no edge at vertex $j$ was removed in step 3(e)
           then remove all edges at vertex $j$ in Diag_Graph
   (g) If at least $t+1$ edges at any vertex $j$ have been removed so far,
       then processor $P_j$ must be faulty, and all edges at $j$ are removed.
   (h) Find a set of processors $P_{decide} \subset P_{match}$ of size $n-2t$ in the updated Diag_Graph,
       such that every pair of $P_j, P_k \in P_{decide}$ still trust each other
   (i) Decide on $v'_i(g) = C^{-1}_{n-2t}(R^{\#}|P_{decide})$

       NOTE: Instead of performing steps 3(h) and 3(i), Algorithm 1 may be repeated for generation $g$ again
       after the diagnosis graph has been updated in step 3(g). This alternative does not affect algorithm correctness.
```

3.2 Checking Stage

When P_{match} is found during the matching stage, the checking stage is entered.

`Lines 2(a), 2(b):` Each fault-free processor $P_j \notin P_{match}$ checks whether the symbols received from the trusted processors in P_{match} are consistent with a valid codeword: that is, check whether $R_j|P_{match} \in C_{n-2t}$. The result of this test is broadcast as a 1-bit notification $Detected_j$, using *Broadcast_1_Bit* . If $R_j|P_{match} \notin C_{n-2t}$, then processor P_j is said to have detected an *inconsistency*.

`Line 2(c):` If no processor announces in Line 2(b) that it has detected an inconsistency, each fault-free processor P_i chooses $C^{-1}_{n-2t}(R_i|P_{match})$ as its output for generation g.

The following lemma argues correctness of Line 2(c).

LEMMA 3. *If no processor detects inconsistency in Line 2(a), all fault-free processors $P_i \in P_{good}$ decide on the identical output value $v'(g)$ such that $v'(g) = v_j(g)$ for all $P_j \in P_{match} \cap P_{good}$.*

PROOF. We assume that the fault-free nodes trust each other. Observe that size of set $P_{match} \cap P_{good}$ is at least $n-2t$, and hence the decoding operations $C^{-1}_{n-2t}(R_i|P_{match})$

and $C^{-1}_{n-2t}(R_i|P_{match} \cap P_{good})$ are both defined at fault-free processors.

Since fault-free processors send correct messages, and trust each other, for all processors $P_i \in P_{good}$, $R_i|P_{match} \cap P_{good}$ are identical. Since no inconsistency has been detected by any processor, every $P_i \in P_{good}$ decides on $C^{-1}_{n-2t}(R_i|P_{match})$ as its output. Also, $C^{-1}_{n-2t}(R_i|P_{match}) = C^{-1}_{n-2t}(R_i|P_{match} \cap P_{good})$, since C_{n-2t} has dimension $(n-2t)$. It then follows that all the fault-free processors P_i decide on the identical value $v'(g) = C^{-1}_{n-2t}(R_i|P_{match} \cap P_{good})$ in Line 2(c). Since $R_j|P_{match} \cap P_{good} = S_j|P_{match} \cap P_{good}$ for all processors $P_j \in P_{match} \cap P_{good}$, $v'(g) = v_j(g)$ for all $P_j \in P_{match} \cap P_{good}$. □

3.3 Diagnosis Stage

When any processor that is not in P_{match} announces that it has detected an inconsistency, the diagnosis stage is entered. The algorithm allows for the possibility that a faulty processor may erroneously announce that it has detected an inconsistency. The purpose of the diagnosis stage is to learn new information regarding the potential identity of a faulty processor. The new information is used to remove one or more edges from the diagnosis graph *Diag_Graph* – as we will soon show, when an edge (j,k) is removed from the diagnosis graph, at least one of P_j and P_k must be faulty. We now describe the steps in the Diagnosis Stage.

`Lines 3(a), 3(b):` Every fault-free processor $P_j \in P_{match}$ uses *Broadcast_1_Bit* to broadcast $S_j[j]$ to all processors. Let us denote by $R^{\#}[j]$ the result of the broadcast from P_j. Due to the use of *Broadcast_1_Bit* , all fault-free processors receive identical $R^{\#}[j]$ for each processor $P_j \in P_{match}$. This information will be used for diagnostic purposes.

`Lines 3(c), 3(d):` Every fault-free processor P_i uses flag $Trust_i[j]$ to record whether it "*believes*", as of this time, that each processor $P_j \in P_{match}$ is fault-free or not. Then P_i broadcasts $Trust_i|P_{match}$ to all processors using *Broadcast_1_Bit* . Specifically,

- If P_i trusts P_j **and** $R_i[j] = R^{\#}[j]$, then set $Trust_i[j]$ =**true** ;
- If P_i does not trust P_j **or** $R_i[j] \neq R^{\#}[j]$, then set $Trust_i[j]$ =**false** .

`Line 3(e):` Using the $Trust$ vectors received above, each fault-free processor P_i then removes any edge (j,k) from the diagnosis graph such that $Trust_j[k]$ = **false** or $Trust_k[j]$ = **false** . Due to the use of *Broadcast_1_Bit* for distributing $Trust$ vectors, all fault-free processors will maintain an identical view of the updated *Diag_Graph* . Note that edges may only be removed from *Diag_Graph* .[1]

`Line 3(f):` As we will soon show, in the case $R^{\#}|P_{match} \in C_{n-2t}$, a processor $P_j \notin P_{match}$ that announces that it has detected an inconsistency, i.e., $Detected_j$ =**true** , must be faulty if no edge attached to vertex j was removed in Line 3(e). Such a processor P_j is "*isolated*", by having all edges attached to vertex j removed from *Diag_Graph* , and the fault-free processors will not communicate with it anymore in subsequent generations.

`Line 3(g):` As we will soon show, a processor P_j must be faulty if at least $t+1$ edges at vertex j have been removed. The identified faulty processor P_j is then isolated.

[1]If the system allows "repair" of faulty processors, then edges will need to be added back to *Diag_Graph* .

`Lines 3(h) and 3(i):` Since *Diag_Graph* is updated only with information broadcast with *Broadcast_1_Bit* (*Detected*, $R^{\#}$ and $Trust$), all fault-free processors maintain an identical view of the updated *Diag_Graph* . Then they can compute an identical set $P_{decide} \subset P_{match}$ containing exactly $n-2t$ processors such that every pair $P_j, P_k \in P_{decide}$ trust each other. Finally, every fault-free processor chooses $C^{-1}_{n-2t}(R^{\#}|P_{decide})$ as its decision value for generation g.

Lemma 4. *Every time the diagnosis stage is performed, at least one edge attached to a vertex corresponding to a faulty processor will be removed from Diag_Graph, and only such edges will be removed.*

Proof. We prove this lemma by induction. For the convenience of discussion, let us say that an edge (j,k) is "*bad*" if at least one of P_j and P_k is faulty.

Consider a generation g starting with any instance of the *Diag_Graph* in which only bad edges have been removed. Suppose that a processor $P_i \notin P_{match}$ "claims" that it detects a failure by broadcasting $Detect_i$ =**true** in Line 2(b). This implies that, if P_i is fault-free, $R_i|P_{match}$ cannot be a valid codeword, i.e., $R_i|P_{match} \notin C_{n-2t}$.

The symbols broadcast by processors of P_{match} in Line 3(a), i.e., $R^{\#}|P_{match}$, can either be a valid codeword of C_{n-2t} or not. We consider the two possibilities separately:

- $R^{\#}|P_{match} \in C_{n-2t}$: In the case, if P_i is actually fault-free, $R^{\#}[k] \neq R_i[k]$ must be true for some faulty processor $P_k \in P_{match}$, which is trusted by P_i. Thus, $Trust_i[k]$ = **false** and the bad edge (i,k) will be removed in Line 3(e). The converse of this argument implies that if $Detected_i$ =**true** but no edge attached to vertex i is removed in Line 3(e), then P_i must be faulty. As a result, all bad edges at vertex i are removed in Line 3(f).
- $R^{\#}|P_{match} \notin C_{n-2t}$: There are always at least $n-2t$ fault-free processors in $P_{match} \cap P_{good}$, and $R_j|P_{match} \in C_{n-2t}$ is **true** for every $P_j \in P_{match} \cap P_{good}$. Thus, if $R^{\#}|P_{match} \notin C_{n-2t}$, then $R^{\#}|P_{match} \neq R_j|P_{match}$ must be true for every $P_j \in P_{match} \cap P_{good}$. Similar to the previous case, $R^{\#}[k] \neq R_j[k]$ must be true for some faulty processor $P_k \in P_{match}$ which is trusted by every processor $P_j \in P_{match} \cap P_{good}$. As a result, $Trust_j[k]$ = **false** and the bad edge (j,k) will be removed in Line 3(e), for all $P_j \in P_{match} \cap P_{good}$.

At this point, we can conclude that by the end of Line 3(f), at least one new bad edge has been removed. Moreover, since $R_i[k] = R^{\#}[k]$ for every pair of fault-free processors $P_i, P_k \in P_{good}$, $Trust_i[k]$ remains **true** , which implies that the vertices corresponding to the fault-free processors will remain fully connected, and each will always have at least $n-t-1$ edges. This follows that a processor P_j must be faulty if at least $t+1$ edges at vertex j have been removed. So all edges at j are bad and will be removed in Line 3(g).

Now we have proved that for every generation that begins with a *Diag_Graph* in which only bad edges have been removed, at least one new bad edge, and only bad edges, will be removed in the updated *Diag_Graph* by the end of the diagnosis stage. Together with the fact that *Diag_Graph* is initialized as a complete graph, we finish the proof. □

The above proof of Lemma 4 shows that all fault-free processors will trust each other throughout the execution of the algorithm, which justifies the assumption made in the proofs of the previous lemmas. The following lemma shows the correctness of Lines 3(h) and 3(i).

LEMMA 5. *By the end of diagnosis stage, all fault-free processors $P_i \in P_{good}$ decide on the same output value $v'(g)$, such that $v'(g) = v_j(g)$ for all $P_j \in P_{match} \cap P_{good}$.*

PROOF. First of all, the set P_{decide} necessarily exists since there are at least $n - 2t \geq t + 1$ fault-free processors in $P_{match} \cap P_{good}$ that always trust each other. Secondly, since the size of P_{decide} is $n - 2t \geq t + 1$, it must contain at least one fault-free processor $P_k \in P_{decide} \cap P_{good}$. Since P_k still trusts all processors of P_{decide} in the updated *Diag_Graph*, $R^{\#}|P_{decide} = R_k|P_{decide} = S_k|P_{decide}$. The second equality is due to the fact that $P_k \in P_{match}$. Finally, since the size of set P_{decide} is $n - 2t$, the decoding operation of $C^{-1}_{n-2t}(R^{\#}|P_{decide})$ is defined, and it equals to $v_k(g) = v_j(g)$ for all $P_j \in P_{match} \cap P_{good}$, as per Lemma 2. □

We can now conclude the correctness of the Algorithm 1.

THEOREM 1. *Given n processors with at most $t < n/3$ are faulty, each given an input value of L bits, Algorithm 1 achieves consensus correctly in L/D generations , with the diagnosis stage performed for at most $t(t + 1)$ times.*

PROOF. Lemmas 1 to 5 imply that consensus is achieved correctly for each generation g of D bits. So the termination and consistency properties are satisfied for the L-bit outputs after L/D generations. Moreover, in the case all fault-free processors are given an identical L-bit input v, the D bits output $v'(g)$ in each generation g equals to $v(g)$ as per Lemmas 1, 3 and 5. So the L-bit output $v' = v$ and the validity property is also satisfied.

According to Lemma 4 and the fact that a faulty processor P_j will be removed once more than t edges at vertex j have been removed, it takes at most $t(t+1)$ instance of the diagnosis stage before all faulty processors are identified. After that, the fault-free processors will not communicate with the faulty processors. Thus, the diagnosis stage will not be performed any more. So it will be performed for at most $t(t + 1)$ times in all cases. □

3.4 Communication Complexity

Let us denote by B the complexity of broadcasting 1 bit with one instance of *Broadcast_1_Bit* . In every generation, the complexity of each stage is as follows:

- Matching stage: every processor P_i sends at most $n-1$ symbols, each of $D/(n-2t)$ bits, to the processors that it trusts, and broadcasts $n - 1$ bits for M_i. So at most $\frac{n(n-1)}{n-2t}D + n(n - 1)B$ bits in total are transmitted by all n processors.

- Checking stage: every processor $P_j \notin P_{match}$ broadcasts one bit $Detected_j$ with *Broadcast_1_Bit* , and there are t such processors. So tB bits are transmitted.

- Diagnosis stage: every processor $P_j \in P_{match}$ broadcasts one symbol $S_j[j]$ of $D/(n - 2t)$ bits with *Broadcast_1_Bit* , and every processor P_i broadcasts $n - t$ bits of $Trust_i|P_{match}$ with *Broadcast_1_Bit* . So the complexity is $\frac{n-t}{n-2t}DB + n(n - t)B$ bits.

According to Theorem 1, there are L/D generations in total. In the worst case, P_{match} can be found in every generation, so the matching and checking stages will be performed for L/D times. In addition, the diagnosis stage will be performed for at most $t(t + 1)$ time. Hence the communication complexity of the proposed consensus algorithm, denoted as $C_{con}(L)$, is then computed as

$$\begin{aligned} C_{con}(L) &= \left(\frac{n(n-1)}{n-2t}D + n(n-1)B + tB\right)\frac{L}{D} \\ &\quad +t(t+1)\left(\frac{n-t}{n-2t}D + n(n-t)\right)B. \end{aligned} \quad (1)$$

For a large enough value of L, with a suitable choice of $D = \sqrt{\frac{(n^2-n+t)(n-2t)L}{t(t+1)(n-t)}}$, we have

$$\begin{aligned} C_{con}(L) &= \frac{n(n-1)}{n-2t}L + t(t+1)n(n-t)B \\ &\quad +2BL^{0.5}\sqrt{\frac{(n^2-n+t)t(t+1)(n-t)}{n-2t}}. \end{aligned} \quad (2)$$

Error-free algorithms that broadcast 1 bit with communication complexity $\Theta(n^2)$ bits are known [3, 6]. So we assume $B = \Theta(n^2)$. Then the complexity of our algorithm for $t < n/3$ becomes

$$\begin{aligned} C_{con}(L) &= \frac{n(n-1)}{n-2t}L + O(n^4L^{0.5} + n^6) \\ &= O(nL + n^4L^{0.5} + n^6). \end{aligned} \quad (3)$$

For $L = \Omega(n^6)$, the communication complexity becomes $O(nL)$. (This requirement can be improved to $L = \Omega(n^5)$ if we use a technique from [1] in the Diagnosis stage.[2])

4. OTHER VALIDITY CONDITIONS

Algorithm 1 satisfies the validity conditions stated in Section 1. As noted earlier, other validity conditions may also be desirable in practice. Algorithm 1 is flexible in the sense that, with proper parameterization, it can achieve other (reasonable) validity properties. In particular, q-validity property defined below can be achieved for $t+1 \leq q \leq n-t$:

- q-Validity: If at least q fault-free processors hold an identical input v, then the output v' agreed by the fault-free processors equals input v_j for some fault-free processor P_j. Furthermore, if $q \geq \lceil \frac{n+1}{2} \rceil$, then $v' = v$.

In order to achieve q-validity, we need to change the error detection code and the size of P_{match} in Algorithm 1 as follows (q is said to be the parameter of the algorithm):

- Throughout the algorithm, we replace the $(n, n - 2t)$ distance-$(2t+1)$ code C_{n-2t} with a $(n, q-t)$ distance-$(n - q + t + 1)$ code, denoted as C_{q-t}. Since $q \geq t + 1$, $q - t \geq 1$. Thus C_{q-t} always exists.

- `Line 1(e):` Choose a set of q processors P_{match} such that $M_j[k] = M_k[j] =$ **true** for every pair of $P_j, P_k \in P_{match}$. [3]

[2]We would like to thank Martin Hirt for suggesting this improvement.

[3]The validity condition achieved can be made stronger, particularly for $q \leq n/2$, by choosing the largest possible set P_{match} with size at least q.

- `Lines 3(h) and 3(i):` For the more general validity conditions, instead of performing steps 3(h) and 3(i), as stated in the NOTE at the end of Algorithm 1, the algorithm can be repeated for generation g with the updated diagnosis graph. For $q > 2t$, we also have the alternative of retaining steps 3(h) and 3(i), but the size of the chosen P_{decide} must be $q - t$.

Similar to Lemmas 1 through 5, we can prove that

1. If at least q fault-free processors $P_i \in P_{good}$ hold the same input $v_i(g) = v(g)$ for some $v(g)$, then a set P_{match} of size q necessarily exists.

2. There are at least $q - t \geq 1$ fault-free processors in P_{match} and all the fault-free processors in P_{match} have the same input for generation g.

3. If no processor detects inconsistency in Line 2(a), all fault-free processors $P_i \in P_{good}$ decide on the identical output value $v'(g)$ such that $v'(g) = v_j(g)$ for all $P_j \in P_{match} \cap P_{good}$. Furthermore, when $q \geq \lceil \frac{n+1}{2} \rceil$, and at least q fault-free processors have identical input v, at least one of these fault-free processors is bound to be in P_{match}, and therefore, $v'(g) = v$.

4. In case (for $q > 2t$) steps 3(h) and 3(i) are used, by the end of diagnosis stage, all fault-free processors $P_i \in P_{good}$ decide on the same output value $v'(g)$, such that $v'(g) = v_j(g)$ for all $P_j \in P_{match} \cap P_{good}$. Otherwise, the algorithm is repeated for generation g with the updated diagnosis graph.

Then it can be concluded that Algorithm 1 achieves q-validity for $t + 1 \leq q \leq n - t$ with the aforementioned parameterization. The communication complexity for achieving q-validity is

$$\begin{aligned} C^q_{con}(L) &= \frac{n(n-1)}{q-t} L + O(n^4 L^{0.5} + n^6) \\ &= O(nL + n^4 L^{0.5} + n^6), \text{ if } q - t = \Omega(n). \end{aligned}$$

When $q < \lceil \frac{n+1}{2} \rceil$, there may be more than one choice for P_{match} in Line 1(e). For example, there can be two disjoint cliques, one containing q fault-free processors with the same input v, and the other containing $q - t$ fault-free processors with input u ($v \neq u$) and t faulty processors that pretend to have input u. It is possible that the second clique is picked to be P_{match} and the consensus value ends up being u. So in this case, even though at least q fault-free processors hold the same input, we can only guarantee agreement on the input of *some* fault-free processors, but not necessarily equal to the q identical inputs. However, when $q \geq \lceil \frac{n+1}{2} \rceil$, it is guaranteed that, if at least q inputs at the fault-free processors equals to v, then the agreed output equals v.

5. MULTIPLE CONSENSUS

In the above discussion, we considered consensus on a single long L-bit value. An alternate view of the problem is likely to be more relevant in practice. In particular, let us consider the problem of performing g instances of consensus, with each processor receiving a D-bit input for each instance (in particular, the input for P_i is $v_i(g)$ for instance g). Then the consensus properties need to be satisfied for each instance *separately.* We assume that the identity of faulty processors remains fixed across the different instances. Then it should not be difficult to see that Algorithm 1 solves the multiple consensus problem, with the algorithm for g-th generation essentially performing the g-th instance of the consensus problem for D-bit values.

Let us denote the *average* complexity for performing g instances of consensus on D-bit values as $\overline{C}_{con}(g, D)$. Similar to the analysis in Section 3.4, for $g \geq t(t + 1)$, we have

$$\begin{aligned} \overline{C}_{con}(g, D) &= \frac{n(n-1)}{n-2t} D + n(n-1)B + tB \\ &\quad + \frac{t(t+1)}{g} \left(\frac{n-t}{n-2t} D + n(n-t) \right) B \quad (4) \\ &= O\left(\left(n + \frac{n^4}{g} \right) D + n^4 + \frac{n^6}{g} \right). \quad (5) \end{aligned}$$

From Eq.5, we can conclude that we only need $D = \Omega(n^3)$ and $g = \Omega(n^3)$ instances of D-bit consensus for the per consensus complexity to reduce to $O(nD)$. In other words, when the goal is to sequentially perform a large number ($\Omega(n^3)$) of instances of consensus, the input size for each instance only needs to be $\Omega(n^3)$ in order to achieve complexity linear in n, rather than $\Omega(n^6)$ as discussed in Section 3.4.

6. RELATED WORK

Binary agreement: Binary agreement corresponds to $L = 1$ in our notation. For binary agreement, optimal error-free algorithms with $O(n^2)$ communication complexity have been proposed [3, 6]. King and Saia [11] introduced a randomized *consensus* algorithm with communication complexity of $O(n^{1.5})$, allowing a non-zero probability of error.

Multi-valued agreement: Fitzi and Hirt [9] proposed a multi-valued *consensus* algorithm in which an L-bit value (or message) is first reduced to a much shorter message, using a universal hash function. Byzantine consensus is then performed for the shorter hashed values. Given the result of consensus on the hashed values, consensus on L bits is then achieved by requiring processors whose L-bit input value matches the agreed hashed value deliver the L bits to the other processors jointly. By performing initial consensus only for the smaller hashed values, this algorithm is able to achieve linear communication complexity for large L and up to $t < n/2$ failures, with a non-zero error probability. Our algorithm can also probabilistically tolerate more than $n/3$ failures if *Broadcast_1_Bit* is replaced by any 1-bit broadcast algorithm that is probabilistically correct up to the desired number of faults.

Beerliova-Trubiniova and Hirt have presented an error-free linear communication complexity multi-party computation algorithm, which uses a linear complexity Byzantine *broadcast* algorithm as a sub-algorithm [2]. This algorithm achieves linear complexity using a *player elimination* framework, which is motivated by the *dispute control* framework proposed in [1]. When two processors disagree with each other (or, *do not trust* each other, in our terminology), one of the two processors must be fault-free. In player elimination, both the processors are removed from the system, and the underlying algorithm is performed on the smaller system, which must now tolerate one fewer faulty processor, with two fewer processors. This approach has also been adopted by asynchronous Byzantine agreement algorithms (e.g. [19]). While it may be possible to also use *player elimination* to achieve consensus, we believe that our approach, in general, can more efficiently achieve stronger validity properties than approaches that may be designed using player elimination.

In designing the proposed algorithm, we drew inspiration from well-known ideas in prior work, as summarized next.

System-level diagnosis: Preparata, Metze and Chien [23] introduced the system *diagnosability* problem in their 1967 paper. Since then there has been a large body of work exploring different variations of the problem (e.g., [18]). The work on system-level diagnosis considers a (un)directed *diagnosis graph* (or a *test graph*), wherein each (un)directed edge represents a *test*: in essence, when node X tests node Y, it may declare Y as *faulty* or *fault-free*, with a faulty tester providing potentially erroneous test outcomes. The goal then is to use the results of the tests to either exactly identify the faulty nodes, or identify a small set of nodes that contains the faulty nodes. The past work differ in the nature of tests being performed, and the nature of the faults being diagnosed. In the system-level diagnosis jargon, our faults are *intermittent* [17], and the tests are *comparison-based* [4]. We interpret the test outcome *fault-free* (*faulty*) as equivalent to the corresponding two processors trusting (not trusting) each other. In our work, we strengthen the comparison-based system-level diagnosis approach by incorporating an error detection code, which provides additional structure to our "comparison test" outcomes. This structure can be exploited for computational efficiency as well (see Appendix).

Linear coding and block coding: A standard mechanism for improving efficiency of information transmission is to use *block codes*, meaning that a "block" (or multiple bits) of data is encoded together in a single codeword. Our specific approach for using linear error detection (block) codes for consensus is motivated by the rich literature on network coding, particularly, multicasting in the presence of a Byzantine attacker (e.g., [25, 5, 10]). Application of such an approach to Byzantine consensus or broadcast in an arbitrary point-to-point networks under per-link capacity constraints is non-trivial [14, 15]. However, under the *communication complexity* model, the problem is simpler, as the algorithm in this paper demonstrates. Essentially, the simplification arises from the ability to treat each point-to-point link identically, resulting in a solution that has a certain symmetry (such a symmetric solution is generally not optimal when the different links have different capacities).

Make the common case fast: In fault-tolerant systems, a common trick to improve average system performance (or reduce average overhead of fault-tolerance) is to make the "common case", namely, the failure-free execution, efficient, with the possibility of much higher overhead when a failure does occur. This approach works well when failure rates are low. There are many instances of the application of this idea, but some examples include error detection followed by retransmission for link reliability, and checkpointing and rollback or roll-forward recovery after failure detection [22].

7. FURTHER RESEARCH

Algorithm 1 has complexity $\frac{n(n-1)}{n-2t}L$ (ignoring the terms sub-linear in L). We have recently developed another algorithm with communication complexity $\frac{n(n-1)}{n-t}L$ [16] (when $q = n - t$), which can be twice as efficient as Algorithm 1 when t gets close to $n/3$. (A Byzantine broadcast algorithm with the same communication complexity is introduced in our earlier report [12].) While the new algorithm may not be better in terms the *order* of the communication complexity, in practice, a factor of 2 reduction in communication overhead is quite significant. Whether this algorithm is optimal for arbitrary t and n ($t < n/3$) remains an open question. What we do know, however, is that the degenerate version of the algorithm for $t = 0$ with complexity $(n-1)L$ is not optimal for all n (when $t = 0$, the consensus problem with $q = n - t$ reduces to the problem of checking *equality* of the inputs at all the processors, which are necessarily fault-free [13]). In [16], we also introduce a consensus algorithm that achieves q-validity with $O(nL)$ communication complexity for all $t+1 \le q \le n-t$, not just when $q-t = \Omega(n)$ (as is the case for the solution in Section 4). In particular, while the solution in Section 4 has $O(nL)$ complexity for $q = t+\Omega(n)$, the complexity is quadratic in n for $q = t+1$. The algorithm in [16] achieves linear complexity even for $q = t+1$ (and can also achieve lower complexity for some other values of q).

Another related research direction of interest is multiple agreements under the constraints of the communication network capacity. In our related work [14, 15], we have studied the Byzantine broadcast and consensus problems in networks where each communication channel has a finite capacity. We proved upper bounds for the throughput of agreement in such networks, and showed their tightness in some (substantially) restricted classes of topologies. The problem is still open in general networks.

8. CONCLUSION

In this paper, we present an efficient error-free Byzantine consensus algorithm for long messages. The algorithm requires $O(nL)$ total bits of communication for messages of L bits for sufficiently large L. The algorithm makes no cryptographic assumptions. With proper parameterization, the proposed algorithm also satisfies a range of validity properties, while still achieving complexity linear in n. The choice of the parameter (called q in the paper) affects the choice of error detection code used to achieve consensus, and also the size of a processor *clique* that the algorithm attempts to identify. With a suitable choice of the error detection code, and using a clique of an appropriate size, the algorithm can trade-off communication cost with the strength of the validity condition.

9. ACKNOWLEDGMENTS

We thank the referees for their insightful comments and asking interesting questions. In particular, the referees asked whether strong forms of validity conditions can be satisfied, and whether the clique required in our algorithm can be determined efficiently. Section 4 and the Appendix address these questions. We thank Martin Hirt (ETH) for his encouraging feedback on our prior related work, and for pointing us to some of the relevant literature on multi-party computation. Thanks also to Manoj Prabhakaran (UIUC) and Ashish Chaudhury (ISI-Kolkata) for their feedback, and to Jennifer Welch for answering our many questions on distributed algorithms. This research is supported in part by Army Research Office grant W-911-NF-0710287 and National Science Foundation award 1059540. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. government.

10. REFERENCES

[1] Z. Beerliova-Trubiniova and M. Hirt. Efficient multi-party computation with dispute control. In *TCC*, 2006.

[2] Z. Beerliova-Trubiniova and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, 2008.

[3] P. Berman, J. A. Garay, and K. J. Perry. Bit optimal distributed consensus. *Computer science: research and applications*, 1992.

[4] D. Blough and A. Pelc. Complexity of fault diagnosis in comparison models. *IEEE Trans. Comp.*, 1992.

[5] N. Cai and R. W. Yeung. Network error correction, part ii: Lower bounds. *Communications in Information and Systems*, 2006.

[6] B. A. Coan and J. L. Welch. Modular construction of a byzantine agreement protocol with optimal message bit complexity. *Inf. Comput.*, 97(1):61–85, 1992.

[7] D. Dolev and R. Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 1985.

[8] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. on Comp.*, 1983.

[9] M. Fitzi and M. Hirt. Optimally efficient multi-valued byzantine agreement. In *ACM PODC*, 2006.

[10] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of byzantine adversaries. In *IEEE INFOCOM*, 2007.

[11] V. King and J. Saia. Breaking the $o(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In *ACM SIGACT-SIGOPS PODC*, 2010.

[12] G. Liang and N. Vaidya. Complexity of multi-valued byzantine agreement. *Tech-Report, UIUC*, June 2010.

[13] G. Liang and N. Vaidya. Multiparty equality function computation in networks with point-to-point links. *Tech-Report, UIUC*, October 2010.

[14] G. Liang and N. Vaidya. Capacity of byzantine agreement with finite link capacity. In *IEEE INFOCOM*, 2011.

[15] G. Liang and N. Vaidya. Capacity of byzantine consensus with capacity limited point-to-point links. *Tech-Report, UIUC*, March 2011.

[16] G. Liang and N. Vaidya. New efficient error-free multi-valued consensus with byzantine failures. *Tech-Report, UIUC*, under preparation (as of March 2011).

[17] S. Mallela and G. Masson. Diagnosable systems for intermittent faults. *IEEE Trans. Comp.*, 1978.

[18] G. M. Masson, D. M. Blough, and G. F. Sullivan. *System diagnosis*. Fault-Tolerant Computer System Design. Prentice Hall, 1996.

[19] A. Patra and C. P. Rangan. Communication optimal multi-valued asynchronous byzantine agreement with optimal resilience. Cryptology ePrint Archive, 2009.

[20] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 1980.

[21] B. Pfitzmann and M. Waidner. Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. *Technical Report, IBM Research*, 1996.

[22] D. Pradhan and N. Vaidya. Roll-forward and rollback recovery: performance-reliability trade-off. *IEEE Trans. Comp.*, 1997.

[23] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Trans. Electr. Comput.*, 1967.

[24] A. Yao. Some complexity questions related to distributive computing. In *STOC*, 1979.

[25] R. W. Yeung and N. Cai. Network error correction, part i: Basic concepts and upper bounds. *Communications in Information and Systems*, 2006.

APPENDIX

To make the algorithm computationally more efficient, we need to modify Algorithm 1 slightly, as elaborated later in this appendix. With this change, the algorithm only looks for a set P_{match} of size $n-t$ such that all the fault-free processors in $P_{match} \cap P_{good}$ have the same input in generation g. The algorithm's response when such a P_{match} is not found is now somewhat different, as sketched below. A complete description and the proof of correctness of the modified algorithm is omitted for brevity.

P_{match} is found as follows. We maintain a set Q that contains the largest set of processors that appear to have identical input up to the previous generation. Initially, Q is the set of all n processors. The matching stage is performed as it is in Algorithm 1, up to Line 3(d). The subsequent steps of the matching stage are different.

(i) Determine the largest set $Q' \subseteq Q$ such that all the processors in set Q' have M vectors that contain at least $n-t$ **true** entries. If $|Q'| < n-t$, then the fault-free processors must have different L-bit inputs, and the algorithm terminates with the decision being a default value. If $|Q'| \geq n-t$, the proceed to the following steps.

(ii) For every pair $P_i, P_j \in Q'$ that trusts each other, if there are more than t distinct processors not trusted by P_i or P_j, then one of P_i and P_j must be faulty. Remove edge (i,j) in the diagnosis graph, set $M_i[j] =$ **false** and $M_j[i] =$ **false** , and go back to step (i) above.

(iii) For every pair $P_i, P_j \in Q'$ (that now trust the same set of at least $n-t$ processors), check whether $M_i[k] = M_j[k]$ for each P_k that is trusted by both P_i and P_j. If this check fails, then either $v_i(g)$ and $v_j(g)$ are different (or, pretending to be different, in case one of these processors is faulty), or P_k has sent different symbols to P_i and P_j. In this case, go to step (iv); otherwise it can be proved that $v_i(g) = v_j(g)$ if P_i and P_j are both fault-free. If all these checks pass, then P_{match} can be chosen as any subset of Q' of size $n-t$, and it always contains a clique of at least $n-2t$ fault-free processors that have the same input for the current generation. Then proceed to the Checking stage as in Algorithm 1.

(iv) If misbehavior, or difference in processor inputs, is detected in step (iii) above, some additional steps are needed: All processors in Q' broadcast their inputs for generation g. Q is then updated as the largest subset of Q' that broadcast the same value. If $|Q| < n-t$, then terminate and decide on a default output. If $|Q| \geq n-t$, then decide on the value broadcast by processors in Q. Additionally, diagnosis is also performed to remove an edge from the diagnosis graph, if misbehavior has indeed occurred.