From $O(n^2)$ to $O(n)$: An Efficient Deterministic Algorithm for Byzantine Agreement

Guanfeng Liang and Nitin Vaidya
Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Email: {gliang2,nhv}@illinois.edu

***Abstract*—In this paper, we introduce an efficient *deterministic* agreement algorithm that solves the multi-valued Byzantine agreement problem *deterministically* for networks of arbitrary size $n \geq 4$ and up to $t < n/3$ failures. This paper considers the "broadcast" version of the agreement problem, wherein the goal is for the nodes in the network to agree on the values that a certain source node wants to broadcast to them. The per-bit communication complexity of an agreement algorithm is defined as the worst case communication complexity for achieving agreement for l bits divided by the message length l. Our algorithm achieves per-bit complexity arbitrarily close to $n(n-1)/(n-t)$ for large value of l. For large l, by using a multi-valued approach, it not only breaks the quadratic lower bound $\Omega(n^2)$ for per-bit complexity from the prior work, it is also the most efficient among the known Byzantine agreement algorithms, including the ones that achieve agreement just probabilistically. Moreover, we believe that, besides being order-optimal, the proposed algorithm is in fact also optimal in the sense of minimizing the per-bit cost.**

I. Introduction

Distributed applications often require cooperation between multiple nodes in a network. In many environments, the distributed applications need to continue to operate correctly despite failures (such as link or node failures) or active attacks (such as a node compromise). Implementing such distributed applications is a difficult task. To simplify the task, *distributed primitives* for commonly used distributed operations are often developed as building blocks using which distributed applications can be built [1], [2]. Some examples of such primitives include ordered message delivery, consensus or agreement, clock synchronization, mutual exclusion, and leader election. Indeed, such distributed primitives have proved to be of great utility. For instance, the ISIS toolkit [3] has been used in several critical applications, including the New York Stock Exchange, and the French Air Traffic Control System. Similarly, the Paxos and multi-Paxos approaches, introduced by Lamport, and the Spread toolkit [4] have also found interesting applications.

Byzantine agreement (BA) is among the most important primitives in fault-tolerant distributed computing. Since initial solutions were presented in the seminal work of Pease, Shostak and Lamport [5], [6], many variations on the Byzantine *agreement* problem have been introduced in the past, with some of the variations also called *consensus*. We will use the following definition of Byzantine agreement (Byzantine generals problem [5]): Consider a network with one node designated as the *sender* or *source*, and the other nodes designated as the *peers*. The goal of Byzantine agreement is for all the fault-free peers to "agree on" the value being sent by the sender, despite the possibility that some of the nodes may be faulty. In particular, the following conditions must be satisfied:

- **Agreement:** All fault-free peers must agree on an identical value.
- **Validity:** If the sender is fault-free, then the agreed value must be identical to the sender's value.
- **Termination:** Agreement between fault-free peers is eventually achieved.

This version of the agreement problem is sometimes called the "broadcast" problem in related literature. We say that a specific algorithm solves the BA problem **deterministically** with up to t failures when the above requirements are always satisfied even in presence of up to t faulty nodes. An algorithm solves the BA problem **probabilistically** if, even with a vanishingly small probability, in some cases at least one of the first two requirements is unsatisfied when it terminates.

The Byzantine agreement (BA) problem was originally introduced for binary values (1 bit). However, in practice, agreement is usually required for long messages rather than on single bits. For example, in a fault-tolerant distributed storage system, the replicas must agree on the files updated at the master, which can be megabytes or even gigabytes of data.

Our goal in this work is to design algorithms that can achieve optimal communication complexity of agreement. When defining complexity, the "value" referred in the above definition of agreement is viewed as a sequence of l *information* bits, with every combination of these l bits representing one of the 2^l possible values. The **communication complexity** of an algorithm $C(l)$ is defined as the maximum of the total number of bits transmitted by all the nodes according to the algorithm until agreement is reached correctly, considering all 2^l possible values and all possible misbehaviors of the faulty nodes. This measure of complexity was first introduced by Yao

Lit.	Complexity	Authentication required?	Probabilistic agreement?
[11]	$\Theta(n^2 l)$	**No**	**Deterministic**
[9]	$\Omega(n^2 l + n^6\kappa)$	Yes	Probabilistic
[10]	$\Omega(n^2 l + n^3\kappa)$	Yes	Probabilistic
[8]	$2nl + \Theta(n^3(n+\kappa))$	**No**	Probabilistic
Ours	$< \frac{3}{2}nl + l^{1/2}\Theta(n^4)$	**No**	**Deterministic**

TABLE I
COMPLEXITY OF EXISTING BA ALGORITHMS. THE COMPLEXITY OF ALGORITHMS FROM[9], [10] ARE CITED FROM [8].

[7], and is then widely used by the distributed computing community [8], [9], [10]. The **per-bit communication complexity** of an algorithm is then defined as

$$\alpha(l) = C(l)/l. \tag{1}$$

II. RESULTS

All our results are about achieving agreement deterministically, which means that under our algorithm, it is impossible for the fault-free nodes to decide on different values. We make the following contributions

- We propose an algorithm which solves the BA problem deterministically for l bits in any network with n nodes and at most $t < n/3$ faulty nodes, and uses $C(l) = \frac{n(n-1)}{n-t}l + l^{1/2}O(n^4)$ bits of communication for large l. Hence, for large l, this algorithm achieves per-bit complexity $\alpha(l)$ approaching $\frac{n(n-1)}{n-t}$, which is **linear** in n. It was believed that it is impossible to achieve agreement deterministically with per-bit communication complexity of $o(n^2)$ [12], [11], [13], [8]. Our algorithm not only breaks the quadratic bound for deterministic algorithms, but for large l, also it is more efficient than previous probabilistic BA algorithms (see Table I).
- We develop a lower bound on the communication complexity of the BA problem as a function of the lower bound on the communication complexity of the multi-party equality function under the point-to-point communication model. Based on this lower bound, we develop a conjecture that $\frac{n(n-1)}{n-t}$ is a lower bound on the per-bit communication complexity of the BA problem.

III. RELATED WORK

Prior work on agreement or consensus: In the seminal work of Pease et al. [5], [6], it is proved that it is impossible to achieve agreement if no fewer than one third of nodes are faulty ($t \geq n/3$), even just for 1 bit. An algorithm that solves BA for 1 bit for all $t < n/3$ is also presented. However, this is a very inefficient algorithm since its complexity is exponential in the number of nodes n. Since then, a lot of work has been done on the BA problem [10], [12], [9], [14], [15], [16], [8]. In 1985, Dolev and Reischuk [12] proved that, without authentication, $\Theta(n^2)$ bits are necessary to be communicated, in order to achieve agreement on 1 bit, which results to a lower bound on the per-bit communication complexity of agreement $\Omega(n^2)$. Algorithms have been derived to achieve this quadratic lower bound [11], [13] for 1-bit agreement.

A lot of efforts have been dedicated into tolerating more than $(n-1)/3$ failures and overcoming the quadratic lower bound, by relaxing the requirements of agreement. For example, the agreement and validity requirements can be relaxed to be satisfied with high probability, rather than satisfied in all possible cases. The adversary model can also be relaxed. If a public-key infrastructure (PKI) is available, making it very unlikely that the faulty nodes can forge false messages without being detected, it is possible to tolerate more failures. In the model with an information-theoretic PKI, agreement on l bits is possible for $t < n/2$ with $\Omega(n^2 l + n^6\kappa)$ bits of communication [9], where κ denotes the security parameter (i.e., the error probability $\epsilon < 2^{-\kappa}l$). In the model with a cryptographic PKI, agreement is possible for $t < n$ with $\Omega(n^2 l + n^3\kappa)$ bits of communication [10] (here κ denotes the length of a signature). However, the per-bit complexity of all these algorithms is still the order of $\Omega(n^2)$. Moreover, all these algorithms only solve BA probabilistically.

The quadratic lower bound on the per-bit communication complexity of agreement remained unbroken until 2006. In [8], a multi-valued probabilistic BA algorithm is introduced, which achieves agreement for l bits with communication complexity $O(2nl + n^3(n+\kappa))$. The quadratic lower bound is overcome by reducing a BA problem with a long message of l bits to a BA problem with much shorter messages, using a universal hash function and allowing a small probability of error, which makes the solution in [8] probabilistic. In addition, the authors proved that their algorithm is order-optimal in the sense that, for large l, $\alpha = 2n = \Theta(n)$ is linear in n. This is order-optimal since at least $(n-1)l$ bits are necessary just for the $n-1$ peers to learn the value, which leads to the result that the per-bit complexity of any BA algorithm must be at least $\Omega(n)$. In contrast with this, our algorithm not only solves the BA problem deterministically, but also is more efficient since $\frac{n(n-1)}{n-t} < 3n/2$. So even our algorithm sticks to the original requirements of the BA problem, it is still at least 25% more efficient than the algorithm that solves the relaxed problem.

Some of the structure of our algorithm has similarities to the work on failure detection [15], [16], [14] and dispute control [17], [18]

Prior work on multicast using network coding: While the early work on fault tolerance typically relied on replication [19] or source coding [20] as mechanisms for tolerating packet tampering, *network coding* has been recently used with significant success as a mechanism for tolerating attacks or failures. In traditional routing algorithms, a node serving as a router, simply forwards packets on their way to a destination. With network coding, a node may “mix” (or *code*) packets from different neighbors [21], and forward the coded packets. This approach has been demonstrated to improve throughput, being of particular benefit in *multicast* scenarios [21], [22], [23]. The problem of *multicast* is related to *agreement*. There has been much research on multicast with network coding in presence of a Byzantine attacker (e.g., [24], [25], [26], [27]).

The significant difference between Byzantine agreement and multicasting is that the multicast problem formulation assumes that the source of the data is always fault-free. In addition, most of the existing work on fault-tolerant network coding assume a link-failure model, while Byzantine agreement considers the nodes to be faulty. In fact, the unicast/multicast problem with node-failure is an open problem in general, and only a few small networks have been solved [28], [29].

IV. Models

A. Network model

We assume a synchronous fully connected network of n nodes, the node IDs (identifiers) are common knowledge. Every pair of nodes is connected with a pair of directed point-to-point links. We assume that all communication links are private and that whenever a node sends a message directly to another node, the identity of the sender is known to the recipient, but we otherwise make no cryptographic assumptions.

We impose no constraint on the capacity of each individual link. In other words, arbitrary number of bits can be transmitted on a link. However, our goal is to minimize the *total number of bits* communicated over all links in the network. This can be viewed as a wired network in which the cost is the sum traffic.

B. Adversary model

We assume a strong adversary. That is, the adversary has complete knowledge on the BA algorithm, the l-bit value the source wants to send, and the packets being sent by every node. The adversary can take over up to $t < n/3$ nodes over the whole execution of the algorithm, possibly including the source. These nodes are said to be *faulty* or *compromised*. The faulty nodes can engage in any kind of deviations from the algorithm, including sending false messages, collusion, and crash failures.

V. The Proposed Byzantine Agreement Algorithm

The Byzantine agreement algorithm we are going to describe in this section distinguishes itself from the existing algorithms in two ways:

- Instead of trying to reach agreement for l bits all at once, our algorithm reaches agreement incrementally: The l bits value is divided into generations each of which has $(n-t)c$ bits (the choice of integer c will be elaborated later). To simplify the discussion, here we assume that l is an integer multiple[1] of $(n-t)c$ on the number of generations, and the proof and results are still correct. A *"basic"* algorithm is run to achieve agreement for every generation. If the faulty nodes misbehave in a particular generation, the misbehavior will be detected and some information of the locations of the faulty nodes will be learned by all fault-free nodes. Then the algorithm will be updated and made more efficient for the following generations. If the faulty nodes repeatedly keep misbehaving in multiple generations, they will be all identified eventually and *removed* from the network.
- In each generation, instead of sending the $(n-t)c$ bits of *raw* data to every peer individually, a linear network coding inspired strategy is used. Essentially, the $(n-t)c$ bits of each generation is encoded and transmitted such that any misbehavior by the faulty nodes will be detected by at least one fault-free peer.

[1]Otherwise, we only need to apply the ceiling function ($\lceil \cdot \rceil$) on the number of generations in determining the complexity in Section V-C.

Normally, if no failure is detected, the algorithm completes in 3 steps and achieves agreement for $(n-t)c$ bits in every generation. On the other hand, if failure is detected during the execution, one more step is added. The algorithm starts assuming no failure is yet detected.

A. Algorithm when no failure is yet detected

Without loss of generality, let us label the n nodes as $0, 1, 2, \ldots, n-1$, and let node 0 be the source. Since no failure is yet detected initially, nothing is known about the locations of the faulty-nodes. Our algorithm specifies what each node should send to every other node. So if a node i transmits nothing to node j when it should, node j detects a failure immediately. This can be considered as node i sending a all-zero message to j, and the following discussion is still valid.

a) Step 1: The source node 0 divides the $(n-t)c$ bits of the current generation into $n-t$ packets of c bits, each packet being a symbol from Galois field GF(2^c). Then node 0 encodes the $n-t$ packets of data into $2(n-1)$ coded packets, each of which is obtained as a linear combination of the $n-t$ packets of data. Let us denote the $n-t$ data packets as the data vector

$$\tilde{x} = [x_1, x_2, \ldots, x_{n-t}] \tag{2}$$

and the $2(n-1)$ coded packets as

$$y_1, y_2, \ldots, y_{2(n-1)}. \tag{3}$$

For the correctness of the algorithm, these $2(n-1)$ coded packets (or symbols) need to be computed such that any subset of $n-t$ encoded packets constitutes independent linear combinations of the $n-t$ data packets. As we know from the design of Reed-Solomon codes, if c is chosen large enough, this linear independence requirement can be satisfied. The weights or coefficients used to compute the linear combinations is part of the algorithm specification, and is assumed to be correctly known to all nodes a priori. Due to the above independence property, any $n-t$ of the $2(n-1)$ symbols, if they are not tampered, can be used to (uniquely) solve for the data vector $\tilde{x}$, i.e., the $n-t$ data packets.

Source node 0 transmits 2 packets y_i and y_{n-1+i} to each peer node i, where $i = 1, \ldots, n-1$. The peers do not transmit in step 1.

b) Step 2: Each peer node i sends packet y_i to all other peer nodes. So by the end of step 2, every peer node i has received n coded packets in total: $y_1, \ldots, y_{n-1}$ and y_{n-1+i} (2 packets directly from the source and $n-2$ packets from the other peers).

c) Step 3: Each fault-free peer finds the solution for each subset of $n - t$ packets from among the n packets received from source and the other peers in steps 1 and 2. If the solution to all these subsets is not unique, then the fault-free peer has detected faulty behavior by some node in the network. We are going to show below that the following two properties hold:

- Property 1: If source is faulty, among the packets all fault-free peers have received, there are at least $n - t$ common packets. In other words, the size of the intersection of the sets of packets the fault-free peers have received is at least $n - t$.

- Property 2: If source is fault-free, the packets sent by source are said to be "correct". Every fault-free peer receives at least $n-t$ correct packets, either directly from the source or via other fault-free peers.

The proof of the following theorem shows that our algorithm has the two properties described above.

Theorem 1: Misbehavior by up to t faulty nodes will either be detected by at least one fault-free peer, or all the fault-free peers will reach agreement correctly.

Proof: Node 0 is said to misbehave only if it sends packets to the peers that are inconsistent – that is, all of them are not appropriate linear combinations of an identical data vector $\tilde{x}$. A peer node i is said to misbehave if it forwards tampered (incorrect) packets other than y_i to (some of) the fault-free peers.

- *Source is faulty:* Since the source node 0 is faulty, there are at least $n - t$ fault-free peers. Let G be the set of the fault-free peers. In this case, $|G| \geq n-t$. According to the way the peers relay their packets in step 2, each fault-free peer sends the same y_i packet to all other peers in G. As a result, all the peers in set G will have at least $|G| \geq n-t$ coded packets that are all identical (as Property 1 above). Consider any fault-free peer, if these $|G|$ packets have a unique solution and it is consistent with the other packets received by this fault-free peer, then the fault-free peer will agree on the unique solution. If these $|G|$ packets do not have a unique solution, or the unique solution is not consistent with some other packets received by the fault-free peer, it will detect a failure. It follows that either all fault-free peers agree on the same solution, or at least one fault-free peer detects a failure.

- *Source is fault-free:* When the source is fault-free, there are at least $n-t-1$ fault-free peers, i.e. $|G| \geq n-t-1$. Similar to the previous case, among the packets a fault-free peer has received, at least $|G| - 1 \geq n - t - 2$ are received from the other fault-free peers and must be correct. In addition, every peer has also received 2 correct packets from source node 0. Together, every fault-free peers has received at least $|G|+1 \geq n-t$ correct packets (as Property 2 above).

  Thus every fault-free peer has received at least $n - t$ correct packets, which have the unique solution $\tilde{x}$. Thus, a fault-free peer will agree on $\tilde{x}$ correctly if it is consistent with the other received packets. Otherwise, it will detect the failure. It follows that either all fault-free peers agree on $\tilde{x}$ correctly, or at least one detects a failure.

■

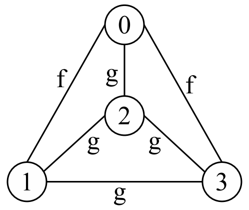


Fig. 1. An example of the diagnosis graph for $n = 4$ and $t = 1$. In this case, node 0 must be faulty.

In step 3, each peer broadcasts to the remaining $n - 1$ nodes a 1-bit notification indicating whether it has detected a failure or not – the agreement among all n nodes on these 1-bit indicators is achieved by using an efficient traditional Byzantine agreement algorithm (e.g. the algorithms from [11], [13]). Since less than 1/3 of the nodes can be faulty, using this traditional algorithm, all fault-free nodes obtain identical 1-bit notifications from all the peers. If none of the notifications indicates a detected failure, then each fault-free peer agrees on the unique solution obtained above, and the current generation is completed. However, if failure detection is indicated by any peer, then one more step is added to the execution.

d) Step 4: When failure detection is indicated by any peer in step 3, the fault-free peers may find different solutions from their received packets. In order to reach an agreement for the current generation and learn some information about the location of the faulty nodes, an extra step is added subsequent to the failure detection.

During step 4, every node (including the source) broadcasts all the packets it has sent to other nodes, or received from other nodes, during steps 1 and 2 – as with the failure notifications in step 3, agreements on these broadcast packets are achieved using the same traditional Byzantine agreement algorithm. Using the information, all fault-free nodes form identical *"diagnosis graphs"* as follows.

The diagnosis graph contains n vertices 0, 1, ..., $n - 1$, corresponding to the n nodes in our network; there is an undirected edge between each pair of vertices, with each edge being labeled as g at time 0 (with g denoting "good"). The labels may change to f (denoting "faulty") during step 4. Once a label changes to f, it is never changed back to g. Without loss of generality, consider the edge between vertices X and Y in the diagnosis graph. At each node, the label for this edge may be set to f in four ways:

- (i) After the broadcast, all nodes obtain identical information about what every node "claims" to have sent and received during steps 1 and 2. Then, each fault-free peer will compare the claims by nodes X and Y about packets sent and received on links XY and YX in steps 1 and 2. If the two claims mismatch, then the label for edge XY in the diagnosis graph is set to f.

- (ii) When X=0 (that is, X is the source), if the packets it claims (in step 4) to have sent (in step 1) do not have a unique solution, then edge 0Y in the diagnosis graph is set to f for all Y$\neq$0.
- (iii) If node X is a peer, and what it claims to have received from the source is inconsistent with any one of the packets it claims to have sent to other peers, then edge XY in diagnosis graph is set to f for all Y$\neq$X.
- (iv) If node X is a peer, and claims to have detected a misbehavior in step 3, but the packets it claims to have received in steps 1 and 2 are inconsistent with this claim, then edge XY in diagnosis graph is set to f for all Y$\neq$X.

In the last three cases, all edges associated with X are set to f. Since the broadcast content is guaranteed to be received identically at all fault-free nodes by the traditional Byzantine agreement algorithm, the diagnosis graph is the same at all fault-free nodes. An example diagnosis graph for $n = 4$ and $t = 1$ thus obtained is illustrated in Fig.1.

The following theorem states that every time a failure is detected, the fault-free nodes will learn some new information about the locations of the faulty nodes. In particular, every time a failure is detected, at least one new edge attached to the faulty nodes will be set to f after step 4.

Theorem 2: At least one edge will be marked as f after step 4 is performed. Every edge marked as f is associated with at least one faulty node.

Proof: Presented in Appendix A. ■

Let us say that nodes i and j accuse(trust) each other if edge ij is marked $f(g)$ in the diagnosis graph. According to Theorem 2, a fault-free node can only be accused by the faulty nodes. This implies that, in a network with at most t failures, if a node is accused by more than t other nodes, this node is identified as faulty.

As a result, if the source node 0 is identified as faulty, the fault-free peers can terminate the algorithm and all agree on some default value. On the other hand, if the source node is not identified as faulty, the packets it broadcast in step 4 must have a unique solution. So the fault-free peers agree on this unique solution as the data packets. Then the current generation completes.

B. Modified algorithm after failure detected

After a failure is detected, and step 4 is finished, a new generation of $(n - t)c$ bits of new data begins if the source node is accused by no more than t peers. The algorithm is modified such that no packet is scheduled between any pair of nodes that accuse each other. So if a peer is identified as faulty, it is *isolated* from the network, and no transmission is scheduled on the links attached to it. The following description of the modified algorithm considers only the nodes that have not been isolated.

e) Step 1: Without loss of generality, assume that node 0 is accused by $m \leq t < n/3$ peers. If node 0 is indeed fault-free, it encodes the $n - t$ packets of data into $2(n - 1 - m) > n - t$ coded packets[2], each of which is obtained as a linear combination of the $n - t$ packets of data. Similar to the case when no failure is yet detected, every subset of $n - t$ encoded packets consists linear independent combinations of the $n - t$ data packets. For convenience, we will index the two packets node 0 sends to node i as y_i and y_{n-1+i}, the same as before. For a node i that is accused by the source, y_i and y_{n-1+i} do not exist.

f) Step 2: Every fault-free peer i that is trusted by the source forwards y_i to every peer j that it trusts.

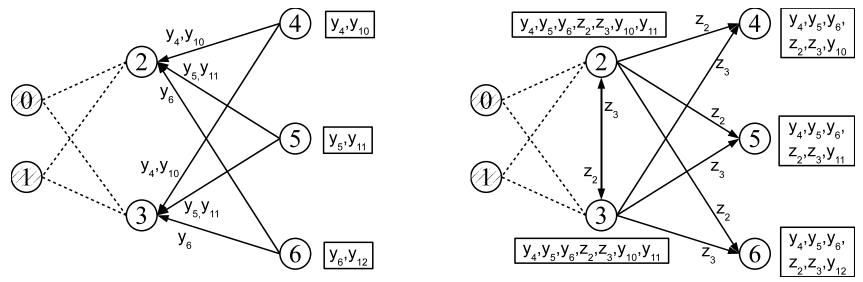
g) Step 3: Every fault-free peer i that is accused by the source node 0 only receives packets from the peers that nodes 0 and i both trust in step 2. If node i receives at least $n - t$ packets, it first checks these packets for consistency, by trying to find the unique solution of every subset of $n - t$ received packets in the same way as before. If these packets are inconsistent, node i detects a failure. Otherwise node i generates one packet z_i as a linear combination of the packets it receives from its trusted peers.

It is possible that peer i receives fewer than $n-t$ packets in step 2. If this is the case, the algorithm will schedule some of the peers trusted by nodes 0 and i both to send a second packet (namely y_{n-1+j}) to node i, until node i receives $n-t$ packets in total. There are always enough packets from the peers both nodes 0 and i trust for this requirement to be satisfied, as shown next. Since we are only considering the nodes that are not isolated (accused by no more than t other nodes), node 0 is accused by at most t peers, and i is accused by node 0 and at most $t - 1$ peers. Thus there are at least $n - 2t$ peers that are trusted by both node 0 and i, and there are at least $2(n - 2t) = (n - t) + (n - 3t) > n - t$ packets that these peers can send to node i (including the packet sent in step 2). Then node i generates a packet z_i as a linear combination of the $n - t$ packets it has received, similar to the previous case.

An example for the latter case in a 7-node network with 2 faulty nodes is shown in Fig.2. In this example, nodes 0 and 1 are faulty, and they both accuse nodes 2 and 3. As a result, nodes 2 and 3 only receives 3 packets (y_4, y_5, y_6) in step 2, from nodes 4, 5 and 6. In step 3, the algorithm schedules nodes 4 and 5 to send the second packets (y_{10}, y_{11}) to nodes 2 and 3. Then nodes 2 and 3 generate z_2, z_3 from $y_4, y_5, y_6, y_{10}, y_{11}$, and send them to their trusted nodes. By the end of step 3, all fault-free peers share $n - t = 5$ packets: z_2, z_3, y_4, y_5, y_6.

For the correctness of the algorithm, any subset of $n - t$ of the union of the y and z packets must be linearly independent. Similar to the case of generating y packets, if c is chosen large enough, this requirement can be satisfied. Since every peer i accused by the source receives at least $n-t$ packets, it can first solve for the unique solution of these packets. If the unique solution exists, then node i generates z_i in the same way as packet y_i is generated from $\tilde{x}$. Otherwise, peer i detects a failure.

[2]Since $m \leq t$, $2(n-1-m) \geq 2(n-1-t) = n-t+(n-t-2)$. From $t \geq 1$ and $n \geq 3t+1$, we have $n-t-2 > 0$, then $2(n-1-m) > n-t$.



(a) Transmissions by nodes 4, 5 and 6 to nodes 2 and 3. Transmissions among nodes 4, 5 and 6 are not shown.

(b) Transmissions by nodes 2 and 3 in step 3. By the end of step 3, all fault-free peers have packets y_4, y_5, y_6, z_2, z_3.

Fig. 2. Examples of some peers accused by the source with $n = 7$ and $t = 2$. Nodes 0 and 1 are faulty. The dotted lines indicates pairs of nodes that accuse each other. The boxes near the fault-free peers indicate the coded packets available at the peers by the end of step 1 and step 3, respectively.

After z_i is generated, every node i accused by the source sends z_i to all the peers it trusts. Then every fault-free peer finds the solution for each subset of $n-t$ packets from among the packets received from the other nodes in steps 1, 2 and 3, and detects a failure if no unique solution is found. Then a 1-bit notification is broadcast as before. Similar to Theorem 1, the following theorem shows that the modified algorithm here also has the two properties described in section V-A. Hence any further misbehavior by the faulty nodes will be detected.

Theorem 3: In the modified algorithm, further misbehavior by the faulty nodes will either be detected by at least one fault-free peer, or all the fault-free peers will reach agreement correctly.

Proof: Here we also consider the cases when source node 0 is fault-free and faulty separately.

- *Source is faulty:* Denote T and A as the set of fault-free peers that are trusted by the source and accused by the source, respectively. Since source is faulty, there are at least $n-t$ fault-free peers, i.e. $|T|+|A| \geq n-t$. As stated in Theorem 2, an edge is set to f only if at least one node associated with this edge is faulty. This implies that all edges connecting two fault-free nodes are always marked as g, and two fault-free nodes will never accuse each other. Thus, every peer $i \in T$ sends y_i to all peers in $T \bigcup A$ during step 2. Similarly, every peer $i \in A$ sends z_i to all peers in $T \bigcup A$ during step 3. As a result, the fault-free peers $T \bigcup A$ share at least $|T|+|A| \geq n-t$ identical packets (as Property 1).
  Similar to the argument of the first part in the proof of Theorem 1, the fault-free peers will either all agree on the unique solution of these $|T|+|A|$ packets, or at least one fault-free peer will detect a failure.

- *Source is fault-free:* Remember that two fault-free nodes will never accuse each other. Thus every fault-free peer receives 2 correct packets from source node 0 in step 1. Then in step 2, since all fault-free peers trust each other, every fault-free peer i sends the correct packet y_i to all other fault-free peers. Then the rest of this proof is the same as the second part of the proof of Theorem 1.

■

h) Step 4: If a failure is detected by a node that accuses no more than t other nodes (otherwise this node must be faulty and must have been isolated as stated before), step 4 is entered and carried out as before. Similar to Theorem 2, at least one new edge adjacent to a faulty node will be set to f in the diagnosis graph.

If by the end of step 4 the source node 0 is accused by more than t peers, it is identified as faulty, and the fault-free peers can terminate the algorithm and all agree on some default value. On the other hand, if the source node is not identified as faulty, the packets it broadcast in step 4 must have a unique solution. So the fault-free peers agree on this unique solution as the data packets as before. Then the current generation completes.

C. Complexity of the proposed algorithm

We have finished describing the proposed Byzantine agreement algorithm above. Now let us study the communication complexity of this algorithm.

1) When no failure is yet detected, excluding step 4:
   - Every peer receives n packets (2 from source and $n-2$ from the other peers), so $n(n-1)$ packets are transmitted in steps 1 and 2, which is $n(n-1)c$ bits in total.
   - Each peer broadcasts 1 bit notification. Let us denote B as the bit-complexity of achieving agreement on 1 bit. So in step 3, totally $(n-1)B$ bits are transmitted.

   So when no failure is yet detected, the number of communicated bits per generation (excluding step 4) is
   $$n(n-1)c + (n-1)B \ bits. \tag{4}$$
2) After failure detected, excluding step 4:
   - Every peer trusted by source node 0 receives 2 packets from source and 1 packet from every peer it trusted. So it receives no more than n packets in steps 1, 2 and 3, which is no more than the number of packets it receives when no failure is yet detected.
   - For every peer i accused by source node 0, let b and p be the number of peers that both source and

node i trust and that only node i trusts, respectively. Node i receives $\max(b, n-t)$ packets from the peers trusted by the source and i both, and p packets from peers trusted only by node i (but accused by the source), respectively. Observe that $b+p \le n-2$ and $p \le t$, thus node i receives at most $\max(b, n-t)+p \le n$ packets in total, which is no more than the number of packets it receives when no failure is yet detected.

- Only nodes accused by no more than t nodes need to achieve agreement on the 1-bit notifications, which results in no more than $(n-1)B$ bits being communicated.

Now it should not be hard to see that after failures are detected, the number of bits communicated per generation is at most the same as the case when no failure is yet detected, excluding step 4. So in the normal operation (Steps 1 to 3), at most $n(n-1)c+(n-1)B$ bits are communicated in every generation.

3) Step 4: Every time step 4 is executed, every packet transmitted in Steps 1 through 3 is broadcast by two nodes: the node that sends this packet and the node that receives this packet. According to the analysis above, no more than $n(n-1)$ packets are transmitted throughout steps 1 to 3. So in every step 4, no more than $2n(n-1)$ packets are broadcast, which results in $2n(n-1)cB$ bits being communicated.

Now we can compute an upper bound of $C(l)$. Notice that $(n-t)c$ bits are being agreed on in every generation, so there are $l/(n-t)c$ generations in total[3]. Thus in steps 1 to 3 in all generations, no more than $\frac{n(n-1)}{n-t}l + \frac{(n-1)B}{(n-t)}\frac{l}{c}$ bits are communicated. Meanwhile, according to Theorem 2, ever time step 4 is performed, at least one new edge associated with a faulty node will be set to f in the diagnosis graph. Since it takes $t+1$ f-edges to identify a faulty node, at most $(t+1)t$ step 4's will be performed before all faulty nodes are identified. So the total number of bits communicated in step 4's in all generations is at most $2n(n-1)(t+1)tcB$ bits. An upper bound on the communication complexity $C(l)$ of the proposed algorithm is then computed as

$$\frac{n(n-1)}{n-t}l + \frac{(n-1)B}{n-t}\frac{l}{c} + 2n(n-1)(t+1)tcB. \qquad (5)$$

Complexity for large l: For a large enough value of l (compared to n), with a suitable choice of $c = \sqrt{\frac{l}{2n(n-t)(t+1)t}}$ in Equation 5, we have

$$C(l) \le \frac{n(n-1)}{n-t}l + 2Bl^{1/2}\sqrt{\frac{2n(n-1)^2(t+1)t}{n-t}} \qquad (6)$$

$$\le \frac{n(n-1)}{n-t}l + 2Bl^{1/2}\sqrt{\frac{n(n-1)^2(n+3)}{3}} \qquad (7)$$

$$= \frac{n(n-1)}{n-t}l + Bl^{1/2}\Theta(n^2). \qquad (8)$$

The "$\le$" in Equation 7 is due to the fact that the second term in Equation 6 is an increasing function of t and $t < n/3$.

Notice that deterministic BA algorithms of complexity $\Theta(n^2)$ are known [11], so we assume $B = \Theta(n^2)$. Then the complexity of our algorithm for all $t < n/3$ upper bounded by

$$C(l) \le \frac{n(n-1)}{n-t}l + l^{1/2}\Theta(n^4) < \frac{3}{2}nl + l^{1/2}\Theta(n^4). \qquad (9)$$

Table I lists the communication complexities for agreeing on l bits, both for the most efficient algorithms in the literature and for the algorithm presented in this paper. In particular, compared with the best known algorithm in [8], our algorithm has strictly lower complexity when $l \ge \omega n^6$ for some constant $\omega > 0$. Moreover, the low complexity in [8] is achieved by allowing a positive probability of error (fault-free nodes may decide on different values), while our algorithm is guaranteed to achieve agreement deterministically such that all fault-free nodes always agree on the same (correct) value.

For a given network with size n, the per-bit communication complexity of our algorithm is upper bounded by

$$\alpha(l) \le \frac{n(n-1)}{n-t} + \frac{\Theta(n^4)}{l^{1/2}} \rightarrow \frac{n(n-1)}{n-t},\ as\ l \rightarrow \infty. \qquad (10)$$

Thus, in the worst case, when $l \rightarrow \infty$, the per-bit communication complexity of our algorithm, referred below as α_{our}, becomes $\alpha_{our} = \frac{n(n-1)}{n-t}$. Note that for finite l, $\alpha(l)$ would exceed α_{our}. However, $\alpha(l)$ will approach α_{our} as l becomes large.

VI. Optimality of our Algorithm

Authors of [8] proved that any algorithm that achieves Byzantine agreement for l bits requires at least $(n-1)l$ bits to be communicated, which implies a lower bound of the per-bit complexity of all BA algorithms: $\alpha \ge n-1$. Their algorithm has communication complexity of $2nl + \Theta(n^3(n+\kappa))$ bits, which results in per-bit communication complexity of $2n$. We will refer to this as $\alpha_{[8]}$ (thus, $\alpha_{[8]} = 2n$). Hence, the algorithm in [8] is *order-optimal.*

On the other hand, according to Equation 10, our algorithm is more efficient since its worst-case per-bit complexity when $l \rightarrow \infty$ is $\alpha_{our} = \frac{n(n-1)}{n-t} \le 3(n-1)/2 < 0.75\alpha_{[8]}$. Thus our algorithm is at least 25% more efficient than the best known algorithm. Moreover, since $\alpha \ge n-1$ for any BA algorithms, our algorithm is within a factor of 1.5 of the optimal. In fact, we believe that our algorithm is really optimal in the sense of minimizing α, i.e. $\frac{n(n-1)}{n-t}$ is a lower bound on the per-bit communication complexity of the BA problem. This claim is stated formally as the following conjecture:

Conjecture 1: In order to achieve Byzantine agreement on l bits, at least $\frac{n(n-1)}{(n-t)}l$ bits need to be communicated in the network.

To argue this conjecture, we first prove the following theorem

[3]To simplify the presentation, we assume that l is an integer multiple of $(n-t)c$ here. For other values of l, the analysis and results are still valid by applying the ceiling function $\lceil\cdot\rceil$ to the number of generations.

Theorem 4: The communication complexity of the any BA algorithm for n nodes and up to t faulty nodes is lower bounded by $\frac{n(n-1)}{(n-t)(n-t-1)}E_l(n-t)$, where $E_l(k)$ is a lower bound of the communication complexity of the following k-party equality function under the point-to-point communication model: Given k nodes each of which is assigned an arbitrary initial value of l bits, if the k initial values are not identical, as least one node will detect a mismatch; otherwise none of the nodes detects mismatch.

Proof: We prove that in a network with n nodes and up to t failures, when no failure is yet detected initially, at least $E_l(n-t)$ bits need to be communicated on links among every subset of $n-t$ nodes, according to the following reduction

Remember that our goal is to achieve agreement with up to t faults (any subset of $\leq t$ nodes). Consider one subset $F = \{f_0, \dots, f_{t-1}\}$ of t nodes containing the source node 0 (f_0) that may be faulty. The other $n-t$ nodes are known to be fault-free and denote the set of these nodes as $G = \{g_1, \dots, g_{n-t}\}$.

Given any algorithm that achieve agreement on l bits, we construct the state machine illustrated in Fig.3. In this state machine, for node g_i, $F_i = \{f_{i,0}, \dots, f_{i,t-1}\}$ is the set of virtual nodes corresponding to F and run the same *correct* code as nodes in F should run, and $g_{i,j}$ is the virtual node corresponding to g_j and runs the same code as node g_j. The good node g_i sends identical messages to node g_j and $g_{i,j}$. Each virtual source node $f_{i,0}$ is given an initial value v_i of l bits. In the real network, node f_j behaves to node g_i as node $f_{i,j}$ in the state machine. It should be easy to see that if $v_1 = v_2 = \cdots = v_{n-t}$ the nodes in F are actually not misbehaving.

Let us assume that all the nodes in G know that nodes in F behave in the way described above. Observe that the behavior F_i and $g_{i,j}$ is fully determined by v_i and the messages node g_i sends. So if g_i knows the value of v_i, it can emulate the behavior of F_i and $g_{i,j}$. Now we can use any Byzantine agreement algorithm for n node and up to t failures to solve the multiparty equality function problem of $n-t$ nodes:

Let each node $i = 1, \dots, n-t$ run the same code as node g_i, and emulate F_i with the initial value v_i. If $v_1 = v_2 = \cdots = v_{n-t}$, it is as if F is not misbehaving in the original n-node network. Then the agreed value v must equal to v_i for all i, and none of the $n-t$ nodes detects a mismatch. On the other hand, if the $n-t$ initial values are not identical, the agreed value v must be different from at least one initial value v_i. Then node i will detect the mismatch. Now, by the definition of $E_l(n-t)$, it is impossible to check whether all initial values are identical with fewer than $E_l(n-t)$ bits being communicated. This means that any BA algorithm must have at least $E_l(n-t)$ bits communicated on links between nodes in G.

Since the location of the faulty peers are unknown a priori, so at least $E_l(n-t)$ bits must be assigned for every subset of $n-t$ peers. For the subsets containing the source and $n-t-1$ peers, a similar reduction can be applied and leads to the same result.

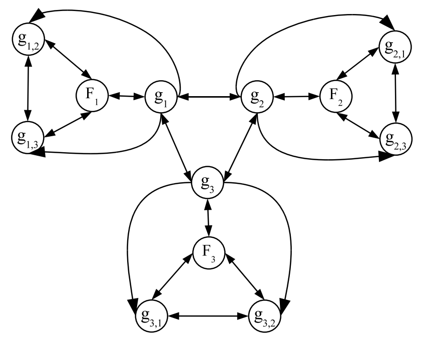


Fig. 3. State machine for $|G| = 3$.

Now we have shown that $E_l(n-t)$ bits are necessary for every subset of $n-t$ nodes in the network in order to solve the BA problem. If we sum over all $\binom{n}{n-t}$ subsets of $n-t$ nodes, the summation is $\binom{n}{n-t}(n-t-1)l$ bits, while each links is counted $\binom{n-2}{n-t-2}$ times. Then we compute the following lower bound on the communication complexity for any BA algorithm:

$$C(l) \geq \frac{\binom{n}{n-t}E_l(n-t)}{\binom{n-2}{n-t-2}} \quad (11)$$

$$= \frac{n(n-1)}{(n-t)(n-t-1)}E_l(n-t). \quad (12)$$

■

So what is $E_l(k)$? We conjecture that

Conjecture 2: Under the point-to-point communication model, $E_l(k) \geq (k-1)l$.

Argument of Conjecture 2: It is easily shown to be true when $l = 1$. Since we assume 1 bit as the smallest unit of data, if two nodes communicate with each other, at least 1 bit is transmitted on one of the two links connecting the two nodes. We call such a pair of nodes *connected*. It should not be hard to see that just to check if all k initial bits are identical, the network must be connected. And it is well-known from graph theory, that to connect k nodes, at least $k-1$ links are needed. Thus, at least $k-1$ bits are necessary just to check initial values of 1 bit among k nodes. Intuitively, every bit of the l-bit initial value is independent. So the checking result of a particular bit is independent of the results of the other bits. This implies that every bit needs to be checked individually. For each bit, we need to form a connected graph using at least $k-1$ bits. Thus, to check l bits, $(k-1)l$ bits are needed.

Although the above intuition sounds reasonable, the formal proof is much more intriguing (consider that the bits a nodes sends can be any arbitrary function of its own initial value and the bits it receives). It is actually a multiparty communication complexity problem [7], [30]. By the fooling-set argument from [7], it is easy to show that every nodes must communicates (send and receive) at least l bits. This gives us a lower bound of $kl/2$ bits. However, this lower bound is obviously not tight for $k \geq 3$ since we have shown that for $l = 1$, it requires at least $k-1 > k/2$ bits.

We are unaware of a past tight lower bound for the communication complexity of the multiparty equality function under the point-to-point communication model.

If Conjecture 2 is proved to be true, then according to Theorem 4, Conjecture 1 is true too. Then for a network of size n with up to t faults, the communication complexity of any Byzantine agreement algorithm is at least $\frac{n(n-1)}{n-t}l$, which results in $\alpha(l) \geq \frac{n(n-1)}{n-t}$. Thus, our algorithm is optimal in the sense of approaching the lower bound of α with large l.

VII. Conclusion

In this work, we have proposed a highly efficient algorithm that solves the Byzantine agreement problem deterministically on values of length $l > 1$ bits. This algorithm uses error detecting network codes to ensure that fault-free nodes will never disagree, and routing scheme that is adaptive to the result of error detection. Our algorithm has communication complexity of $\frac{n(n-1)}{n-t}l + l^{1/2}\Theta(n^4)$, which leads to a linear cost $\frac{n(n-1)}{n-t}$ per-bit agreed upon for large enough value of l, and overcomes the quadratic lower bound $\Omega(n^2)$ in [12]. Such linear per-bit complexity has only been achieved in the literature by allowing a positive probability of error. Our algorithm achieves the linear per-bit complexity while achieving agreement correctly deterministically in all possible cases. In addition, our algorithm has the lowest per-bit communication complexity among all known BA algorithms, including those that reaches agreement probabilistically. In fact, we believe that our algorithm achieves the lowest per-bit communication complexity among all algorithms that solves the BA problem.

Acknowledgment

This research is supported in part by Army Research Office grant W-911-NF-0710287. 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. We thank Martin Hirt and Valerie King for their feedback.

References

[1] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1995.
[2] K. Birman, *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer, 2005.
[3] ——, "A history of the virtual synchrony replication model," 1994.
[4] Y. Amir and J. Stanton, "The spread wide area group communication system," 1998.
[5] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *JOURNAL OF THE ACM*, 1980.
[6] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. on Programming Languages and Systems*, 1982.
[7] A. C.-C. Yao, "Some complexity questions related to distributive computing(preliminary report)," in *STOC '79*, 1979.
[8] M. Fitzi and M. Hirt, "Optimally efficient multi-valued byzantine agreement," in *PODC '06*, 2006.
[9] B. Pfitzmann and M. Waidner, "Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$," *Technical Report, IBM Research*, 1996.
[10] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing*, vol. 12(4), pp. 656–666, 1983.
[11] P. Berman, J. A. Garay, and K. J. Perry, "Bit optimal distributed consensus," *Computer science: research and applications*, 1992.
[12] D. Dolev and R. Reischuk, "Bounds on information exchange for byzantine agreement," *J. ACM*, vol. 32, no. 1, pp. 191–204, 1985.
[13] B. A. Coan and J. L. Welch, "Modular construction of a byzantine agreement protocol with optimal message bit complexity," *Inf. Comput.*, vol. 97, no. 1, pp. 61–85, 1992.
[14] E. S. Amdur, S. M. Weber, and V. Hadzilacos, "On the message complexity of binary byzantine agreement under crash failures," *Distributed Computing*, 1992.
[15] V. Hadzilacos and J. Y. Halpern, "The failure discovery problem," in *Mathematical Systems Theory*, 1996.
[16] ——, "Message optimal protocols for byzantine agreement," in *Mathematical Systems Theory*, 1996.
[17] M. Hirt, U. Maurer, and B. Przydatek, "Efficient secure multi-party computation," in *ASIACRYPT 2000*, 2000.
[18] Z. Beerliova-Trubiniova and M. Hirt, "Efficient multi-party computation with dispute control," in *TCC*, 2006.
[19] E. C. Cooper, "Replicated distributed programs," in *SOSP'85*, 1985.
[20] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, 1989.
[21] S.-Y. Li, R. Yeung, and N. Cai, "Linear network coding," *Information Theory, IEEE Transactions on*, vol. 49, no. 2, pp. 371–381, Feb. 2003.
[22] R. Koetter and M. Medard, "An algebraic approach to network coding," in *ISIT'01*, 2001.
[23] C. Fragouli, D. Lun, M. Medard, and P. Pakzad, "On feedback for network coding," in *CISS'07*, 2007.
[24] N. Cai and R. W. Yeung, "Network error correction, part ii: Lower bounds," *Communications in Information and Systems*, 2006.
[25] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks using randomized network coding," 2004.
[26] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard, "Resilient network coding in the presence of byzantine adversaries," in *INFOCOM'07*, 2007.
[27] S. Kim, T. Ho, M. Effros, and S. Avestimehr, "New results on network error correction: capacities and upper bounds," in *ITA'10*, 2010.
[28] O. Kosut and L. Tong, "Nonlinear network coding is necessary to combat general byzantine attacks," in *47th Annual Allerton Conference on Communication, Control, and Computing*, October 2009.
[29] G. Liang, R. Agarwal, and N. Vaidya, "Secure capacity of wireless broadcast networks," *Technical Report, CSL, UIUC*, September 2009.
[30] D. Dolev and T. Feder, "Multiparty communication complexity," *Foundations of Computer Science, Annual IEEE Symposium on*, 1989.

Appendix
Proof of Theorem 2

Proof: Suppose to the contrary that after step 4, no edge is marked as f. This implies that the packets the source node broadcasts in step 4 have an unique solution. Moreover, these packets are also consistent with the packets each peer broadcasts in step 4. As a result, the packets each peer broadcasts must have the same unique solution.

Recall that the broadcast in step 4 is performed only after some peer claims to have detected a failure. As a result, if the peer that claims to have detected a failure is fault-free, the packets it broadcasts in step 4 cannot have an unique solution; on the other hand, if this peer is faulty, what it broadcasts in step 4 contradicts with the 1-bit notification it issues in step 3. In either way, it leads to a contradiction. So at least one edge will be marked as f after step 4.

In addition, it is easy to see that an edge between two fault-free nodes will never be marked as f, since the packets broadcast by two fault-free nodes in step 4 will never mismatch. This completes the proof of Theorem 2. ■