All-to-All Gradecast using Coding with Byzantine Failures

John F. Bridgman, III[*] and Vijay K. Garg[**]

Parallel and Distributed Systems Lab
Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX USA
johnfbiii@utexas.edu, garg@ece.utexas.edu

Abstract. This paper presents a method that uses forward error correction codes to minimize the message bit complexity when acquiring consistent global information in the presence of faulty processes. We show a modification to the gradecast algorithm that implements our method. Gradecast, first proposed by Feldman and Micali, is a broadcast algorithm for distributed systems that can handle Byzantine failures. It can be used as a basic building block to solve many important problems in distributed computing in the presence of Byzantine failures, such as agreement, clock synchronization, and approximate agreement. Many of these problems require a step where all processes need to send information to all other processes. We refer to the version of gradecast where all processes broadcast to all other processes as all-to-all gradecast. In a distributed system with n processes, n instances of the original gradecast algorithm to perform all-to-all gradecast has a message bit complexity of $O(mn^3)$, where m is the length of the message. In this paper, we present an all-to-all gradecast algorithm that takes $O(mtn^2)$ message bits, where t is the maximum number of faulty processes. This is a significant reduction in message bit complexity in real systems where $t << n$. Our all-to-all gradecast algorithm uses coding theory to mask Byzantine failures and has wide applicability in distributed systems. For example, by replacing the original gradecast in the byzantine agreement algorithm proposed by Ben-Or, Dolev and Hoch with $O(mtn^3)$ message bit complexity, we get a new byzantine agreement algorithm with $O(mt^2n^2)$ message bit complexity. Also, this algorithm can be used with their approximate agreement algorithm to get $O(kn^2t)$ instead of $O(kn^3)$ message bit complexity.

1 Introduction

Many distributed algorithms require a step in which every participating process needs a value from every other process. For example, in a clock synchronization

[*] This research was supported in part by the Virginia and Ernest Cockrel, Jr. Fellowship in Engineering.

[**] This research was supported in part by the NSF Grants CNS-0718990, CNS-1115808, Cullen Trust for Higher Education Endowed Professorship.

algorithm, every process may collect the values of clocks of all other processes. In a sensor network, a group of sensors may collect values from each other to compute the average value, or some other global function such as the minimum, the maximum or the sum of all the values. In a system that requires a uniform action, the processes may collect proposals from all other processes to determine an action. This paper addresses these problems in the presence of Byzantine failures. Many fault tolerant algorithms need to have information about what other processes know about other processes. We call this second-order information. In order to perform a fault tolerant broadcast, second-order knowledge is required. The usual method to acquire second-order information is for every process to broadcast the information that they have; then, every process rebroadcasts what they receive. But, rebroadcasting the information is inefficient when it is known that the number of faulty processes is bounded. The technique described in this paper uses a forward error correction (FEC) code to minimize the size of the messages that are rebroadcast. As an example, we apply the technique to gradecast. Gradecast can be used as a basic building block for many distributed algorithms that handle Byzantine failures.

The gradecast algorithm, first proposed by Feldman and Micali[1], is a broadcast algorithm that gives the receivers a confidence level in the value received. Let $value_j[k]$ be the value that process P_j outputs for process P_k, $confidence_j[k]$ be the confidence value process P_j outputs for process P_k, and v_k be the initial input value to the algorithm for process P_k. The confidence level returned is from the set $\{0, 1, 2\}$ and the confidence value gives information about the state of the other processes. The gradecast algorithm provides three main properties of the confidence level that allow a process to reason about the knowledge of other processes.

1. For all non-faulty process P_i, and non-faulty process P_j, and any process P_k, if $confidence_j[k] > 0$ and $confidence_i[k] > 0$; then, $value_j[k] = value_i[k]$.
2. For any non-faulty process P_i, and non-faulty process P_j, and any process P_k, $|confidence_i[k] - confidence_j[k]| \leq 1$.
3. If P_k is non-faulty, then for all non-faulty processes P_i, $confidence_i[k] = 2$ and $value_i[k] = v_k$.

The original one-to-all gradecast algorithm broadcasts a value from one process to all the other processes. We define message bit complexity as the total number of bits sent by all non-faulty processes in one invocation of the algorithm. The one-to-all gradecast algorithm has a message bit complexity of $O(mn^2)$, where m is the length of the message and n is the number of processes. The properties of gradecast make it a useful primitive in distributed systems.

Consider the case where all processes wish to broadcast a value to all other processes using gradecast. We call this all-to-all gradecast and it is used in many applications such as Byzantine agreement, approximate agreement, and multi-consensus[2]. The standard implementation of all-to-all gradecast, where n instances of the one-to-all gradecast algorithm are used, has $O(mn^3)$ message bit complexity. We show a method, using coding, that gives an all-to-all gradecast algorithm with only $O(mtn^2)$ message bit complexity, where t is the specified maximum number of faulty processes. This is a significant reduction in message bit complexity when t is much smaller than n, which is usually the case. Gradecast requires $t < n/3$ for correctness.

Our all-to-all gradecast algorithm uses error correction codes[3] to mask Byzantine failures and has wide applicability in distributed systems. For example, by replacing the original gradecast in the byzantine agreement algorithm proposed by Ben-Or, Dolev and Hoch[2] with $O(mtn^3)$ message bit complexity, a new byzantine agreement algorithm with $O(mt^2n^2)$ message bit complexity results. If the number of actual failures is $f \leq t$, then, the algorithm by Ben-Or, Dolev and Hoch will take $\min(f+2, t+1)$ rounds. This property is often referred to as early stopping. The bit complexity of approximate agreement algorithm [4, 5, 2] is reduced from $O(kn^3)$ to $O(kn^2t)$, where k is the number of rounds used in the approximate algorithm. Algorithms that have better message bit complexity exist; but, they sacrifice round complexity or reduce the maximum number of faulty processes tolerated. The example byzantine agreement algorithms in this paper are given because of the simplicity of their implementation on top of an all-to-all gradecast algorithm. There exist algorithms with better message bit complexity. For example, the algorithm by Coan and Welch[6] has message bit complexity of $O(t^2 + nt)$ to agree on a single bit. This algorithm does not posses an early stopping property.

Error correction codes can be viewed as a projection from a smaller space to a larger space with good separation. Because the points in the larger space are separated, small perturbations in the point in the larger space are still close to the original mapped point and the point in the original smaller space can be recovered. Generally, the spaces are high dimensional vector spaces over finite fields and the measure of distance between two elements of the space is the number of coordinates that have a different value. *Systematic* codes can be constructed that encode a vector as the original vector concatenated with an error correction vector. Our method relies on the observation that every process is sending a value to every other process and only the faulty processes will send conflicting data. So, the vector built at each process will differ in at most t locations. This can be viewed as transmitting the vector and each process receiving a corrupted version. Then, only the error correction part of the encoded vector can be sent between processes to correct these "errors". The original vector is not actually transmitted. In a traditional application of error correction codes, an input block is encoded and then the whole output codeword is transmitted. We are not transmitting the whole codeword. Only a portion of the codeword is transmitted. A proper selection of the code allows an error correction vector that can correct t errors to be no longer than $2t + 1$.

This method of using coding is also applicable to other types of broadcast algorithms. Srikanth and Toueg[7] give a broadcast algorithm to simulate authenticated broadcasts that has the important properties of authenticated messages. These are as follows: If a correct process P_i broadcasts a message; then, all other correct processes receive that message and if a correct process P_i does not broadcast a message; then, no correct process receives a message from P_i. The message bit complexity of a consistent broadcast is $O(mn^2)$ and therefore, the message bit complexity of all-to-all consistent broadcast is $O(mn^3)$. By using our method, the bit complexity of all-to-all consistent broadcast can be reduced to $O(mtn^2)$.

All-to-all gradecast can also be used to implement an interactive consistency algorithm. Interactive consistency[8, 9] is the problem in which each process has a vector with an entry that needs to be filled from every other process and all vectors should be the same at the end of the algorithm. Interactive consistency is at least as difficult as Byzantine agreement.

There are earlier works that use error correcting codes for Byzantine broadcast algorithms. Liang and Vaidya[10] give an algorithm that achieves communication complexity of $O(mn)$ bits for broadcast with Byzantine failures if $m = \Omega(n^6)$. This is quite useful in situations where the message being broadcast is a very long stream of bits. An example of such messages is all the samples from a sensor in a long running system. However, for small message size, m, the communication complexity is $O(nm + n^4 m^{1/2} + n^6)$. Our work is useful when every process is doing a broadcast and the message size may not be large. Friedman, Mostéfaoui, Rajsbaum and Raynal [11] show a mapping from a distributed agreement problem to a coding problem. Our approach is to use coding to reduce the size of the messages being sent. The work by Krol[12] gives a set of algorithms that use coding to perform Byzantine consensus. Essentially, Krol[12] replaces broadcast with encoding, and decision making with decoding. These algorithms are based on the original algorithm by Pease, Shostak and Lamport[13] and have exponential message complexity.

The remainder of this paper is organized in the following manner. First, an overview of the original algorithm is given in section 2. The algorithm is described in section 3. Next, proofs of its correctness are in section 4. Then, in section 5, applications of an all-to-all gradecast algorithm are discussed. Concluding remarks are in section 6.

Table 1. Notation

n	number of processes
t	maximum number of faulty processes
i, j, k	process IDs
u, v, w, x, y, z	scalar values
U, V, W, X, Y, Z	non-scalar values
$confidence_i[j]$	confidence value P_i has in P_j's value
$value_i[j]$	value process P_i received from process P_j
G	set of all non-faulty processes

2 One-to-All Gradecast

2.1 Execution Model

The execution model used in this paper is the standard reliable synchronous message passing model. Processes can only communicate by passing messages. Processes are assumed to be fully connected. Message passing is assumed to be such that a process knows the identity of who sent the message. Only deterministic algorithms are considered. The algorithm assumes that only t out of the n processes in the system may fail; but, they may be Byzantine, that is to say, faulty in arbitrary ways. For algorithm correctness, we require that $n - 2t > t$ which simplifies to $t < n/3$. This bound is optimal because a Byzantine agreement algorithm can be build on top of gradecast that has the same requirements on t as the underlying gradecast algorithm. It has been proven that no Byzantine agreement algorithm exists for $t \geq n/3$. This model assumes that authenticated messages are not available; otherwise, the broadcast problem becomes trivial. Authenticated messages allow a process to verify the message's contents and source.

2.2 Overview of original one-to-all gradecast

This section gives a quick overview of the original algorithm presented by Feldman and Micali[1]. This algorithm broadcasts a value from one process to all other processes. This can be modified to an all-to-all gradecast algorithm by vectorizing. Pseudo-code for the algorithm is in Fig. 1. It assumes that the values n, t, and h are common knowledge to all processes, where n is the number of processes, t is the maximum number of faulty processes, and h is the broadcasting process. The algorithm proceeds in four steps. In the first step, the broadcaster h sends out its value to all processes. After this step, the algorithm is symmetric. In the second step, all process rebroadcast the value received from h to all other processes. Then, in the third step, each process looks at the values received from Step 2. If there is a common value that has been received at least $n - t$ times; then, it broadcasts that value. Otherwise, the process broadcasts no value. Finally, in Step 4, the received values from Step 3 are examined. Let x be the value that appears the most in Z_i. If there is a tie between two values, some common agreed upon tie breaking strategy must be performed. For example, if values are real numbers, we can always take the minimum. If x appears at least $2t + 1$ times; then, P_i outputs x with confidence 2. If x appears less than $2t + 1$ and more than t times; then, P_i outputs x with confidence 1. Otherwise, P_i outputs $\perp$ with confidence 0.

This algorithm has message bit complexity $O(mn^2)$ and when replicated to perform all-to-all gradecast, will have $O(mn^3)$ message bit complexity. The next section gives a vectorized modification to this algorithm that reduces the all-to-all gradecast message bit complexity to $O(mtn^2)$.

```
$P_i$::
    Inputs to $P_h$:
        $v_h$ : input value for broadcaster
    Common Knowledge:
        $n$ : number of processes
        $t$ : maximum number of faulty processes
        $h$ : broadcaster
    Variables:
        $u_i$ : value process $P_i$ receives in Step 2
        $X_i[1..n]$ : vector of values received in Step 3
        $Z_i[1..n]$ : vector of values received in Step 4

    // Step 1
    if $i = h$ then
        for $j$ : 1 to $n$ do $P_i.send(P_j, v_h)$; end
    end
    // Step 2
    $u_i = P_i.receive(P_h)$;
    for $j$ : 1 to $n$ do $P_i.send(P_j, u_i)$; end
    // Step 3
    for $j$ : 1 to $n$ do $X_i[j] = P_i.receive(P_j)$; end
    if $\exists x$ such that $|\{k : X_i[k] = x\}| \geq n - t$ then
        for $j$ : 1 to $n$ do $P_i.send(P_j, x)$; end
    end
    // Step 4
    for $j$ : 1 to $n$ do
        if $P_j$ sent a message then
            $Z_i[j] = P_i.receive(P_j)$;
        else $Z_i[i] = \perp$; end
    end
    if $\max_x |\{k : Z_i[k] = x\}| \geq 2t + 1$ then
        $value_i = \arg\max_x |\{k : Z_i[k] = x\}|$;
        $confidence_i = 2$;
    elseif $\max_x |\{k : Z_i[k] = x\}| > t$ then
        $value_i = \arg\max_x |\{k : Z_i[k] = x\}|$;
        $confidence_i = 1$;
    else
        $value_i = \perp$; $confidence_i = 0$;
    end
    Output $value_i$ and $confidence_i$.
```

Fig. 1. Original one-to-all gradecast algorithm

3 Algorithm For All-To-All Gradecast

This section gives our all-to-all gradecast algorithm that has $O(mtn^2)$ message bit complexity. This algorithm is based on vectorizing the gradecast algorithm presented by Feldman and Micali[1]. As before, each process P_i has an input value v_i and the algorithm produces two vectors $value_i$ and $confidence_i$ which are the received values and the confidence level respectively. The algorithm assumes that the set of all messages can be encoded as members of a finite field, with one field member reserved to represent "no message" which we will denote as $\perp$. This assumption only requires that there exists a mapping between the messages and the field elements such that every message has a unique field element assigned to it with at least one field element unassigned.

There is a standard technique, called interleaving, to apply a small code to larger blocks without increasing the code length. The tool Parity Archive Volume Set[14] uses this technique. Our usage of this technique relies on the fact that only t blocks may be corrupt. It is very similar to breaking up the message to be transmitted into blocks and running each block through the code, except it is broken into interleaved blocks. What this means for the problem here, is that, if a code that uses octets as the basic unit and one message is ten octets; then, the first block will be the first octet from each message in the vector of messages, the second block will be the second octet from each message, and so on. Note, for the purposes here, the blocks are only interleaved in this manner for the encoding and decoding process. For example, if the vector to encode is $[[a, b], [c, d], [e, f]]$; then, $[a, c, e]$ would be run through the encoder to produce $[a, c, e, g, h]$ and $[b, d, f]$ to produce $[b, d, f, i, j]$ and the final output of the encoder is then $[[a, b], [c, d], [e, f], [g, i], [h, j]]$, which is then used in our algorithm. With this method, messages longer than the field size can be used.

Pseudo-code for the algorithm is provided in Fig. 2. This algorithm proceeds in four steps. The following description is from the point of view of process P_i, because the algorithm is symmetric. First, in Step 1, P_i broadcasts its value to every other process. Step 2 starts to differ from the original gradecast algorithm. The original algorithm rebroadcasts the values received from Step 1. Because of the messaging system reliability, $\forall P_i, P_j, P_k \in G : V_i[k] = V_j[k]$, where G is the set of all non-faulty processes. This implies that $\forall P_i, P_j \in G : |\{k : V_i[k] \neq V_j[k]\}| \leq t$. This means that at least $n - t$ values between non-faulty processes are identical; so, sending the whole vector, V_i, is inefficient. Therefore, our algorithm uses coding techniques to send at most $2t + 1$ values, which can be used in conjunction with the knowledge that the receiving process possesses to recover everything the sender knows. To finish Step 2, P_i sends the error correction vector of V_i.

In Step 3, P_i receives the encoded message from all other processes and uses its current knowledge to construct matrix X_i of all the values that every process claims that every other process possesses as their input value. The value $X_i[j][k]$ is the value that j claims k sent to it. The reliability of the messaging system and how the coding process works implies that $\forall P_i, P_j \in G, \forall k : X_i[j][k] = V_j[k]$. Now an array Y_i is constructed from X_i in the following manner. For each P_j, if there is a value that appears at least $n - t$ times in the column $X_i[\cdot][j]$; then, set $Y_i[j]$ to that value, otherwise, set $Y_i[j]$ to $\perp$. Then, an encoding of Y_i is sent to all processes.

Finally, in Step 4, Z_i is constructed in the same manner as X_i in Step 3. P_i uses its knowledge of Y_i and the encoded value sent to it from each other process j to recover Y_j and then places that value in the row $Z_i[j][\cdot]$. That gives the property $\forall P_i, P_j \in G, \forall k : Z_i[j][k] = Y_j[k]$. Then, P_i looks at columns of $Z_i[\cdot][j]$ for each P_j to decide its output. If $\max_x |\{k : Z_i[k][j] = x\}| \geq 2t + 1$; then, P_i sets $value_i[j] = x$ and $confidence_i[j] = 2$. If $2t + 1 > \max_x |\{k : Z_i[k][j] = x\}| > t$; then, P_i sets $value_i[j] = x$ and $confidence_i[j] = 1$. Otherwise, P_i sets $value_i[j] = \perp$ and $confidence_i[j] = 0$. Notice that the reduction in message bit complexity comes from taking advantage of the knowledge that is known to be common across processes, because of the constraint that at most t processes can be faulty. The processes also do not know which of the t values are not common. This is why they must exchange information in Step 2 and 3. But, coding is used to ensure that the amount of information exchanged is small.

3.1 Example

The following example shows how the algorithm works. For this example, $n = 4$ and $t = 1$. The possible messages are the non-zero values over the finite field $GF(2^8)$ and the zero value is reserved to represent no message. Let P_4 be the faulty process and let the initial value for the non-faulty processes be $\{241, 86, 35\}$. For the encoder, we will use a Reed Solomon[15] code with a code length of 2^8 that can correct one error. The error correction terms are calculated by taking the remainder of the values to encode as a polynomial with the generator polynomial $102 + 164x + x^2$ over the finite field $GF(2^8)$. For example, $[241, 86, 35, 35]$ is encoded as the polynomial $35x^{251} + 35x^{252} + 86x^{253} + 241x^{254}$. The remainder is taken, which gives us the polynomial $78 + 39x$, which corresponds to the values $[39, 78]$. Note that all the arithmetic operations are done over the finite field $GF(2^8)$. Decoding is much more involved and we recommend the reader consult the literature on the subject[3, 15]. The Schifra[16] library was used to compute these values.

For Step 1, all processes send their values to all other processes. For this example, the received values for each process are:

$$\begin{aligned} V_1 &= [241, 86, 35, 35] \\ V_2 &= [241, 86, 35, 35] \\ V_3 &= [241, 86, 35, 40] \end{aligned} \tag{1}$$

Encoding these we get:

$$\begin{aligned} [V_1, Vecc_1] &= [241, 86, 35, 35, 39, 78] \\ [V_2, Vecc_2] &= [241, 86, 35, 35, 39, 78] \\ [V_3, Vecc_3] &= [241, 86, 35, 40, 82, 30] \end{aligned} \tag{2}$$

Then, each process sends the $Vecc_i$ values which are of length two.

Next, for Step 3, all processes receive the values sent in Step 2. Since P_4 is faulty, it will send $[22, 77]$ to P_1, $[0, 136]$ to P_2 and $[121, 159]$ to P_3. For this

```
$P_i$::
 Inputs:
     $v_i$ : Input value for $P_i$
 Common knowledge:
     $n$ : The number of processes
     $t$ : Maximum number of faulty processes
 Variables:
     $V_i[1..n]$ : Vector received in Step 2, initially $\perp$
     $Vecc_i[1..2t+1]$ : error correction vector for $V_i$
     $X_i[1..n][1..n]$ : Matrix of decoded values in Step 3
     $Y_i[1..n]$ : Vector of values computed in Step 3
     $Yecc_i[1..2t+1]$ : Error correction vector for $Y_i$
     $Z_i[1..n][1..n]$ : Matrix of decoded values in Step 4
     $value_i[1..n]$ : Vector of output values
     $confidence_i[1..n]$ : Vector of confidence levels

 // Step 1
 for $j$ : 1 to $n$ do $P_i.send(P_j, v_i)$; end
 // Step 2
 for $j$ : 1 to $n$ do $V_i[j] = P_i.receive(P_j)$; end
 $Vecc_i = encode(V_i)$;
 for $j$ : 1 to $n$ do $P_i.send(P_j, Vecc_i)$; end
 // Step 3
 for $j$ : 1 to $n$ do $X_i[j] = decode(V_i, P_i.receive(P_j))$; end
 $\forall j$ let $Y_i[j] = x$ if $\exists x$ s.t. $|\{k : X_i[k][j] = x\}| \geq n - t$ otherwise $Y_i[j] = \perp$
 $Yecc_i = encode(Y_i)$;
 for $j$ : 1 to $n$ do $P_i.send(P_j, Yecc_i)$; end
 // Step 4
 for $j$ : 1 to $n$ do $Z_i[j] = decode(Y_i, P_i.receive(P_j))$; end
 for $j$ : 1 to $n$ do
     if $\max_x |\{k : Z_i[k][j] = x\}| \geq 2t + 1$ then
         $value_i[j] = \arg\max_x |\{k : Z_i[k][j] = x\}|$;
         $confidence_i[j] = 2$;
     elseif $\max_x |\{k : Z_i[k][j] = x\}| > t$ then
         $value_i[j] = \arg\max_x |\{k : Z_i[k][j] = x\}|$;
         $confidence_i[j] = 1$;
     else $value_i[j] = \perp$; $confidence_i[j] = 0$;
     end
 end
 Output $value_i$ and $confidence_i$.
```

Fig. 2. All-to-all gradecast algorithm

example, the processes then receive:

$$\begin{array}{l} P_1.receive(P_1) = [39, 78] \\ P_1.receive(P_2) = [39, 78] \\ P_1.receive(P_3) = [82, 30] \\ P_1.receive(P_4) = [22, 77] \end{array} \tag{3}$$

$$\begin{aligned} P_2.receive(P_1) &= [39, 78] \\ P_2.receive(P_2) &= [39, 78] \\ P_2.receive(P_3) &= [82, 30] \\ P_2.receive(P_4) &= [0, 136] \end{aligned} \tag{4}$$

$$\begin{aligned} P_3.receive(P_1) &= [39, 78] \\ P_3.receive(P_2) &= [39, 78] \\ P_3.receive(P_3) &= [82, 30] \\ P_3.receive(P_4) &= [121, 159] \end{aligned} \tag{5}$$

Each process concatenates the received value to the end of its V_i vector and runs this through the decoder to get:

$$X_1 = \begin{bmatrix} 241, 86, 35, 35 \\ 241, 86, 35, 35 \\ 241, 86, 35, 40 \\ 241, 49, 35, 35 \end{bmatrix} \tag{6}$$

$$X_2 = \begin{bmatrix} 241, 86, 35, 35 \\ 241, 86, 35, 35 \\ 241, 86, 35, 40 \\ 241, 86, 129, 35 \end{bmatrix} \tag{7}$$

$$X_3 = \begin{bmatrix} 241, 86, 35, 35 \\ 241, 86, 35, 35 \\ 241, 86, 35, 40 \\ 157, 86, 35, 40 \end{bmatrix} \tag{8}$$

Following the instructions for building Y_i in Step 3 we get:

$$\begin{aligned} Y_1 &= [241, 86, 35, 35] \\ Y_2 &= [241, 86, 35, 35] \\ Y_3 &= [241, 86, 35, 0] \end{aligned} \tag{9}$$

Then building $Yecc_i$ gets:

$$\begin{aligned} [Y_1, Yecc_1] &= [241, 86, 35, 35, 39, 78] \\ [Y_2, Yecc_2] &= [241, 86, 35, 35, 39, 78] \\ [Y_3, Yecc_3] &= [241, 86, 35, 0, 8, 182] \end{aligned} \tag{10}$$

Each process i then sends its $Yecc_i$. Let P_4 send $[87, 77]$ to process 1 and 2 and $[123, 149]$ to process 3.

Finally, in Step 4, each process constructs the Z_i matrix in the same way it constructed the X_i matrix. Then, we have:

$$Z_1 = Z_2 = \begin{bmatrix} 241, 86, 35, 35 \\ 241, 86, 35, 35 \\ 241, 86, 35, 0 \\ 241, 86, 0, 35 \end{bmatrix} \tag{11}$$

$$Z_3 = \begin{bmatrix} 241, 86, 35, 35 \\ 241, 86, 35, 35 \\ 241, 86, 35, 0 \\ 241, 86, 35, 82 \end{bmatrix} \tag{12}$$

Finally, the algorithm will output for each process:

$$\begin{array}{ll} value_1 = & [241, 86, 35, 35] \\ confidence_1 = & [2, 2, 2, 2] \\ value_2 = & [241, 86, 35, 35] \\ confidence_2 = & [2, 2, 2, 2] \\ value_3 = & [241, 86, 35, 35] \\ confidence_3 = & [2, 2, 2, 1] \end{array} \tag{13}$$

4 Proof of Correctness

In this section, we show the correctness of our algorithm. The first lemma shows a crucial property of Y in Step 3 of Fig. 2.

Lemma 1. *Assume P_i and P_j are non-faulty processes. In Step 3 of Fig. 2, if P_i sets $Y_i[k]$ to $x \neq \perp$ and P_j sets $Y_j[k]$ to $y \neq \perp$; then, $x = y$. Formally, $\forall P_i, P_j \in G, \forall k : Y_i[k] \neq \perp \wedge Y_j[k] \neq \perp \implies Y_i[k] = Y_j[k]$.*

Proof. If P_i sets $x \neq \perp$ to $Y_i[k]$, then the kth column of X_i contained at least $n - t$ copies of x. Only t rows can correspond to faulty processes, so at least $n - 2t$ of the rows that contain x in column k come from non-faulty processes. This means that those $n - 2t$ non-faulty processes also sent vectors to P_j which set x to the kth column for those processes. Suppose $y \neq x$ and $y \neq \perp$. This means that there must be $n - 2t$ values which are $\perp$ in the kth column of X_j at process P_j. But, $n - 2t > t$ so P_j will set $Y_j[k]$ to $\perp$, which contradicts that $y \neq \perp$ and $y \neq x$.

Theorem 1 (Property (1)). *All non-faulty processes with positive confidence about process k have identical $value[k]$. Formally,*

$$\forall P_i, P_j \in G, \forall k : confidence_i[k] > 0 \wedge confidence_j[k] > 0$$

implies

$$value_i[k] = value_j[k].$$

Proof. First note that the Z_i matrix will contain the Y_j vectors from Step 3 of Fig. 2 for all P_j. By Lemma 1, if there is a majority of a value that is not $\perp$ in the kth column of Z_i; then, all values in that column that are not the majority and not $\perp$ are from a faulty process. This implies that if any non-faulty process P_j sets $confidence_i[k] \geq 1$; then, all other non-faulty processes P_j that set $confidence_j[k] \geq 1$ also set $value_j[k] = value_i[k]$.

Theorem 2 (Property (2)). *For any two non-faulty processes, the difference in their confidence levels for any process P_k can differ by at most* 1*. Formally,* $\forall P_i, P_j \in G, \forall k : |confidence_i[k] - confidence_j[k]| \leq 1$.

Proof. Assume some non-faulty process P_i sets $confidence_i[k] = \delta$ and $value_i[k] = x$. Process P_i setting $confidence_i[k] = \delta$ implies that a set R of processes sent x to P_i in Step 3 of Fig. 2. Let $R_e \subseteq R$ be the faulty processes that sent x to P_i. By problem setup, $|R_e| \leq t$. This means that the number of processes that also sent x to any other process can differ by at most t. Let P_j be the process that receive the most messages in support of x. Then, all other processes receive at least $|R| - t$ messages in support of x. Step 4 of the algorithm in Fig. 2 compares the support of x to $2t + 1$ and t to select the confidence level. By the above reasoning, the support of x differs by at most t between any non-faulty process. Therefore, the difference in confidence level between any non-faulty processes is at most 1.

Theorem 3 (Property (3)). *If P_k is non-faulty, then, all non-faulty processes P_i have the value sent by process P_k and their confidence level on this value is* 2*. Formally,*
$\forall P_i, P_k \in G : (confidence_i[k] = 2) \wedge (value_i[k] = v_k)$.

Proof. If P_i is a non-faulty process, then, all processes will receive v_i from P_i in Step 2 of Fig. 2. Next, all non-faulty processes will also claim that P_i sent v_i for Step 3. Let G be the set of all non-faulty processes, by the assumptions of our problem $|G| \geq n - t$ and all non-faulty processes will distribute error correction vectors with v_i in the ith entry in Step 3. So, every non-faulty process P_j will set $confidence_j[i] = 2$ and $value_j[i] = v_i$ in Step 4.

Theorem 4. *The algorithms in Fig. 2 has bit message complexity of $O(mtn^2)$.*

Proof. In Step 1, every process sends its value to every other process taking mn^2 message bits. In Step 2, each process computes $Vecc_i$ which contains at most $2t + 1$ values of length m bits. Every process then sends its $Vecc_i$ to every other process resulting in at most $m(2t+1)n^2$ message bits. In Step 3, the same number of message bits are sent as Step 2. This results in a total of at most $mn^2 + 2m(2t+1)n^2$ message bits being sent by this algorithm. This is $O(mtn^2)$.

5 Application

The all-to-all gradecast algorithm can be used to create an exceptionally simple byzantine agreement algorithm. Ben-Or, Dolev and Hoch[2] give a simple algorithm for Byzantine agreement and approximate agreement based on the gradecast algorithm. A modification to the gradecast algorithm is needed for their Byzantine agreement algorithm. The modification is to make the algorithm take a set of known faulty processes that the algorithm will ignore and set all values for processes in the faulty set to $\perp$. This has the effect of making that process in the faulty set disappear as if they had crashed. The Byzantine consensus algorithm is symmetric, can agree upon an arbitrary value (as long as there is some method of resolving a tie), and has an early stopping property. They define early stopping to mean if there are $f \leq t$ actual failures; then, the algorithm terminates in $\min(f+2, t+1)$ rounds. The message bit complexity with our all-to-all gradecast algorithm is $O(mt^2n^2)$.

The algorithm starts off with a faulty set which is initially empty. Then, for each round r up to $t+1$ rounds the algorithm performs as follows: The algorithm performs an all-to-all gradecast of the current value ignoring all processes in the faulty set. The algorithm then adds up how often each value was received which had a confidence greater than or equal to one. Next, it sets the current value to the value that has the largest count. If there is more than one with the same count; then, use some tie breaking scheme, such as always choosing the smaller value. The algorithm adds all processes that have confidence one or less to the faulty set. Next, the algorithm counts the number of processes that sent the current value with confidence 2. If this count is greater than $n-t$, the algorithm performs one more iteration of the loop and then exits the loop prematurely. To finish, the algorithm returns the current value.

The approximate agreement algorithm presented by Ben-Or, Dolev and Hoch is very similar to the byzantine agreement algorithm described above. The all-to-all gradecast algorithm we describe can be plugged into their algorithm without changing any of the properties of the original algorithm.

6 Conclusion

Many algorithms have a step where every process broadcasts a value. Gradecast is a broadcast algorithm that gives a confidence level to each receiving process. This confidence level gives information about the state of other processes. We have presented an all-to-all gradecast with message bit complexity $O(mtn^2)$. The original gradecast algorithm presented by Feldman and Micali[1] is a one to all broadcast protocol. Using the original gradecast algorithm to produce all-to-all gives $O(mn^3)$ message bit complexity. Our algorithm can be used in place of the original gradecast algorithm when an all-to-all broadcasts is used. The algorithm presented uses coding to reduce the amount of redundant information being transmitted. We presented proofs that our modified algorithm maintains the important properties of the original gradecast. Having an all-to-all gradecast algorithm that is efficient in message bit complexity admits a simple symmetric arbitrary valued Byzantine agreement with early stopping property that only takes $O(mt^2n^2)$ message bit complexity. Other algorithms may also benefit from using coding in the fashion presented here.

References

1. P. Feldman and S. Micali, "Optimal algorithms for byzantine agreement," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ser. STOC '88. New York, NY, USA: ACM, 1988, pp. 148–161. [Online]. Available: http://doi.acm.org/10.1145/62212.62225

2. M. Ben-Or, D. Dolev, and E. N. Hoch, “Simple gradecast based algorithms,” Sep. 2010. [Online]. Available: http://arxiv.org/abs/1007.1049
3. R. M. Roth, *Introduction to coding theory.* Cambridge University Press, 2006.
4. D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl, “Reaching approximate agreement in the presence of faults,” *Journal of the ACM*, vol. 33, pp. 499–516, 1986.
5. A. D. Fekete, “Asymptotically optimal algorithms for approximate agreement,” in *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, ser. PODC ’86. New York, NY, USA: ACM, 1986, pp. 73–87. [Online]. Available: http://doi.acm.org/10.1145/10590.10597
6. B. A. Coan and J. L. Welch, “Modular construction of a byzantine agreement protocol with optimal message bit complexity,” *Information and Computation*, vol. 97, no. 1, pp. 61 – 85, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089054019290004Y
7. T. K. Srikanth and S. Toueg, “Simulating authenticated broadcasts to derive simple fault-tolerant algorithms,” *Distributed Computing*, vol. 2, pp. 80–94, 1987, 10.1007/BF01667080. [Online]. Available: http://dx.doi.org/10.1007/BF01667080
8. J.-M. Hélary, M. Hurfin, A. Mostéfaoui, M. Raynal, and F. Tronel, “Computing global functions in asynchronous distributed systems prone to process crashes,” in *International Conference on Distributed Computing Systems*, 2000, pp. 584–591.
9. M. Pease, R. Shostak, and L. Lamport, “Reaching agreement in the presence of faults,” *J. ACM*, vol. 27, pp. 228–234, April 1980. [Online]. Available: http://doi.acm.org/10.1145/322186.322188
10. G. Liang and N. H. Vaidya, “Error-free multi-valued consensus with byzantine failures,” *CoRR*, vol. abs/1101.3520, 2011.
11. R. Friedman, A. Mostéfaoui, S. Rajsbaum, and M. Raynal, “Asynchronous agreement and its relation with error-correcting codes,” *IEEE Trans. Computers*, vol. 56, no. 7, pp. 865–875, 2007.
12. T. Krol, “Interactive consistency algorithms based on voting and error-correcting codes,” in *TwentyFifth International Symposium on Fault-Tolerant Computing, Digest of Papers, FTCS-25 Silver Jubilee, IEEE Computer Society Press, Los Alamitos*, 1995, pp. 89–98.
13. L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, pp. 382–401, July 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176
14. “Parchive: Parity archive tool,” http://parchive.sourceforge.net/.
15. I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
16. A. Partow, “Schifra reed-solomon error correcting code library,” http://www.schifra.com/.