THE BYZANTINE FIRING SQUAD PROBLEM

James E. Burns and Nancy A. Lynch

ABSTRACT

A new problem, the Byzantine firing squad problem, is defined and solved in two versions, permissive and strict. Both problems provide for synchronization of n initially unsynchronized processors in a synchronous network, in the absence of a common clock and in the presence of a limited number of faulty processors. Solutions are given which take the same number of rounds as Byzantine agreement but transmit at most r times as many bits, where r is the number of rounds. Additional solutions are provided which use one (permissive) or two (strict) additional rounds and send only a constant times the number of bits used by a chosen Byzantine agreement algorithm.

Categories and Subject Descriptors: C.2.4 **[Computer–Communication Networks]**: Distributed Systems; D.1.3 **[Programming Techniques]**: Concurrent Programming; D.4.1 **[Operating Systems]**: Process Management—*synchronization*; D.4.5 **[Operating Systems]**: Reliability—*fault tolerance*; D.4.7 **[Operating Systems]**: Organization and Design—*distributed systems*; *real-time systems*

Advances in Computing Research, vol. 4, pages 147–161

ISBN: 0-89232-682-4

General Terms: reliability.

Additional Key Words and Phrases: agreement, Byzantine generals problem, firing squad problem.

1. INTRODUCTION

We consider a problem of synchronizing a collection of processors, some of which might be faulty. We assume that the processors are connected by a complete, synchronous network. Although communication is synchronous, we will not assume the global availability of a "current time." A solution to this synchronization problem, which we call the "Byzantine firing squad" problem, would be useful in the following types of situations.

a. *Real-time processing.* It might be necessary for several processors to carry out some external action simultaneously, perhaps after the occurrence of a particular unpredictable event. For example, several processors on board an aircraft might be responsible for causing several actuators to perform a specific action in concert, in response to a signal from the pilot. The signal might arrive at the different processors at different times. A Byzantine firing squad algorithm could be used to synchronize the processors' actions.
b. *Distributed initiation.* Most synchronous parallel distributed algorithms assume that all processors begin their protocols together. If we would like to use such algorithms in a network in which there is no common notion of time, we need to cause the processors participating in the algorithm to synchronize their start times. A preliminary Byzantine firing squad algorithm could be used to accomplish this.
c. *Distributed termination.* In certain algorithms (e.g., synchronous probabilistic agreement [1] and approximate agreement [7]), individual processors might complete their parts of the algorithm at different times. If it is necessary to guarantee simultaneous termination, a Byzantine firing squad algorithm could be run after the main algorithm.

This synchronization problem can be considered to be a combination of two well-known problems: the firing squad synchronization problem and the Byzantine generals problem. Accordingly, we call the new problem the *Byzantine firing squad problem.*

The firing squad synchronization problem was first proposed in about 1957 by John Myhill and described by Edward Moore in 1962 [19]. In the

original problem, a finite but unknown number of finite-state machines connected in a line are to be programmed so that they all go to a particular state ("fire") simultaneously after a "start" signal is given by one of the machines at the end of the line, the "general." Over the years, this problem has been generalized and widely studied (see the bibliography in Nishitani and Honda [20]). In our problem, the finite-state machines are replaced by automata connected by a complete network. This would make the problem trivial if we did not introduce the possibility of faulty behavior.

The Byzantine generals problem was first proposed by Pease, Shostak, and Lamport [21], although it did not receive that name until a later work appeared [18]. For a recent bibliography of work on the problem, see Fischer [12]. The Byzantine generals problem can be paraphrased as follows. The general must broadcast a value to the remaining processors, even though some processors might be faulty. If the general is a reliable processor, then all reliable processors must correctly determine the value. Even if the general is faulty, all reliable processors must agree on some (arbitrary) value. (A reliable processor always behaves according to a given protocol, while a faulty processor can behave in an arbitrary way.) We will assume that all processors are acting as generals, broadcasting a local value to the others, so that at the end of the algorithm all reliable processors agree on a vector of values. Thus, *Byzantine agreement* for broadcasting a local value of each processor is reached if and only if the algorithm always terminates and the following conditions hold at termination:

(A1) *Agreement:* All reliable processors agree on the same vector of values.

(A2) *Validity:* If processor i is reliable, then the ith component of the agreed upon vector is the value that i broadcast.

A Byzantine agreement algorithm is called *f-resilient* if Byzantine agreement is reached for any number of faulty processors not exceeding f. For the remainder of the paper we will use f for the maximum number of faulty processors which can be tolerated and n for the total number of processors.

The Byzantine firing squad problem combines the firing squad problem with the Byzantine generals problem. Initially, all the (reliable) processors are "quiescent" (not communicating). At an unpredictable time, we can require the system to begin the firing protocol. This is done by sending **START** signals (from an external source) to some of the processors (possibly at different times). Within a finite number of rounds, all of the reliable processors must simultaneously "fire," even though a limited number of processors might exhibit "Byzantine" failure.

Section 2 gives a more formal description of two versions, permissive and strict, of the Byzantine firing squad problem. The versions differ in the

number of **START** signals which the external source must send to force firing. Section 3 presents a family of solutions to these Byzantine firing squad problems; each solution is based on a chosen Byzantine agreement algorithm. These solutions take no more rounds than the chosen algorithm and require sending at most r times as many bits, where r is the maximum number of rounds used by the Byzantine agreement algorithm. We show in Section 4 how to reduce the factor r to a constant with the addition of only one preliminary round for the permissive case and two preliminary rounds for the strict case.

We hope that our solutions will seem simple and clear to the reader, but this should not imply that the algorithms are easily obtained. Indeed, a direct solution to the problem is not immediately obvious. Instead, we solve the problem by reducing it to the problem of Byzantine agreement. That is, we describe an "algorithm schema" that converts an arbitrary Byzantine agreement algorithm to an algorithm for the Byzantine firing squad problem. We encourage the reader to consider the problem carefully before examining the solutions in Sections 3 and 4.

2. THE DEFINITION OF THE PROBLEM

We model a synchronous system by a state transition system. We will not burden the reader with a lot of notational detail, but trust that the following description is sufficient to construct the formal state transition system that we have in mind.

A *synchronous system* consists of a set of processors, an initial state for each processor, and transition functions which determine the protocols of the processors. In each transition (also referred to as a *round*), a processor receives a message from every processor and from an external source, sends a message to every processor and goes to a new state. Each processor has a set of *firing states* distinct from its initial state.

The reliable processors always send the messages specified by their protocols, but the faulty processors can send any messages. In particular, we do *not* assume a secure authentication capability is available. (In a system with authentication, a processor can *sign* a message in an unforgeable way so that all other processors can reliably determine the original sender of the message [9].) The Byzantine firing squad problem with authentication is examined by Coan, Dolev, Dwork, and Stockmeyer [5].

In a synchronous system, information can be conveyed by the absence of a signal as well as by an explicit signal. We distinguish a particular message, called the *null message* ($\varnothing$) to represent the absence of an explicit signal; all other messages are simply called *signals.* A processor is said to be *quiescent* at a certain state if, in any transition from that state in which

it receives only null messages, it sends only null messages and remains in the same state. If a processor is not quiescent, then it is *awake.*

We require that all reliable processors be quiescent in their initial states. Initial quiescence guarantees that no signals will be sent by any reliable processor until the external source of a faulty processor sends a signal to some reliable processor.

For the Byzantine firing squad problem, the only signal which is ever sent by the external source is a special **START** signal, which is used to initiate the firing protocol. (Only the external source can send **START**.) A processor *fires* the first time it enters a firing state. (We only consider a single firing, but it should be obvious how to modify our solution for multiple firings.)

The Byzantine firing squad problem admits several variations depending on how we wish to force firing. We might want firing to occur if just a single **START** signal (from the external source) is received by any reliable processor. Note that this implies that a faulty processor can cause firing by pretending to be a reliable processor which has received a **START** signal. On the other hand, if we prohibit firing until some reliable processor has received a **START** signal, then a single **START** signal is not sufficient to guarantee firing, since a lone processor cannot (in general) convince the others that it is reliable. We term these two variations *permissive* and *strict.* (An algorithm which solves one of these does not solve the other.)

A computation in which at most f processors exhibit faulty behavior (by sending messages which contradict their protocols) is called *f-correct.* An f-resilient *permissive Byzantine firing squad algorithm* must satisfy the following conditions for every f-correct computation:

(C1) *Agreement:* If any reliable processor fires in some round, then all reliable processors fire in that round.

(C2) *Permissive Validity:* If any reliable processor receives a **START** signal, then some reliable processor eventually fires.

An f-resilient *strict Byzantine firing squad algorithm* will satisfy (C1) and the following additional condition for every f-correct computation:

(C2′) *Strict Validity:* (a) If at least $f + 1$ reliable processors receive a **START** signal, then some reliable processor eventually fires. (b) If any reliable processor fires, then some reliable processor previously received a **START** signal.

We wish to measure the efficiency of communication of our algorithms. It is not useful to measure the direct costs incurred by faulty processors, since such costs might be unbounded. For the firing squad problems, we

also wish to avoid charging for "preliminary rounds" that are caused by faulty processors.

Let $\mathcal{A}$ be an algorithm and C be a computation of $\mathcal{A}$. (A *computation* records all of the states of the processors and all of the messages sent during each round of an execution of an algorithm.) If $\mathcal{A}$ is a Byzantine agreement algorithm, then the *starting point* is the beginning of the algorithm; if $\mathcal{A}$ is a permissive Byzantine firing squad algorithm, then the *starting point* occurs at the first reception of a **START** message by a reliable processor; if $\mathcal{A}$ is an f-resilient strict Byzantine firing squad algorithm, then the *starting point* occurs when the $(f+1)$st reliable processor receives a **START** signal. (Note that for the firing squad algorithms, there can be computations in which the starting point is undefined.) We define $start(C)$ to be the number of rounds from the beginning of the algorithm until the starting point is reached, if this is defined. If $\mathcal{A}$ is a Byzantine agreement algorithm, then C *terminates* when all reliable processors halt, while if $\mathcal{A}$ is a Byzantine firing squad algorithm, then C *terminates* when all reliable processors fire. If termination is reached in C, then $finish(C)$ is the number of rounds from the beginning of the algorithm until termination, otherwise $finish(C)$ is undefined. We define $length(C)$ to be $finish(C) - start(C)$ when $start(C)$ and $finish(C)$ are both defined, and to be 0 otherwise. If $\mathcal{A}$ is f-resilient, we define $MaxRounds(\mathcal{A})$ to be the maximum value of $length(C)$ for all f-correct computations C of $\mathcal{A}$.

There does not seem to be a generally accepted method for measuring communication costs of distributed systems in absolute terms, perhaps because order results are sufficient for most analysis. We choose a measure which leads to a simple analysis and a cleanly stated result. Any "reasonable," coding-independent measure would give similar results. Our measure is made on a round-by-round basis, both because many Byzantine agreement algorithms use longer and longer messages for succeeding rounds and because we need to combine messages from different rounds in our simulation. On the other hand, we ignore differences between individual processors or communication links. (Some Byzantine agreement algorithms optimize by running the algorithm on only a subset of processors when f is small. Our measure ignores this optimization.)

Let $\mathcal{A}$ be an f-resilient algorithm with $MaxRounds(\mathcal{A}) = r$, and let C be a computation of $\mathcal{A}$. The round in which termination is reached is called round r, and the preceding rounds are called round $r-1$, round $r-2$, and so on. (This is convenient for the firing squad algorithms because it allows us to use equal labels for rounds which occur the same number of rounds before firing in different computations. For firing squad algorithms, the firing point is a more reasonable origin for a time scale than the beginning of the computation.) We also label points in the conversation by round numbers. To be precise, we use "*at round r*" to refer to the point in the

computation just before the round r message is sent. Over all f-correct computations of $\mathcal{A}$, let $M_i(\mathcal{A})$ be the set of messages sent by reliable processors in round i. Then the (worst case) *number of bits sent in round i of* $\mathcal{A}$ is $m_i(\mathcal{A}) = n^2 \log_2|M_i(\mathcal{A})|$, for i in $\{1, \ldots, MaxRounds(\mathcal{A})\}$. [The n^2 factor represents a charge for each of the n^2 messages sent in each round, even including messages sent by faulty processors. However, we do not charge for messages not in $M_i(\mathcal{A})$ which might be sent by a faulty processor in round i.] Finally, we let $Bits(\mathcal{A}) = \sum_{i=1}^{r} m_i(\mathcal{A})$. Note that *Bits* is a measure of *bits of information* and thus is independent of any particular coding used for the messages [15].

3. TIME-EFFICIENT SOLUTIONS OF THE BYZANTINE FIRING SQUAD PROBLEMS

Our solutions are based on an arbitrary Byzantine agreement algorithm (which satisfies the restriction specified below). Our algorithms inherit most of the characteristics of the chosen agreement algorithm, so that behavior can be tailored as desired (e.g., minimizing *MaxRounds* or *Bits*). Also, the resiliency of the derived Byzantine firing squad algorithm is identical to that of the Byzantine agreement algorithm. Since it is known that $n > 3f$ is sufficient for Byzantine agreement [18], the Byzantine firing squad problem can also be solved whenever $n > 3f$. It has also been shown by Coan, Dolev, Dwork, and Stockmeyer [5] (by reducing Lamport's weak Byzantine agreement problem [17] to the Byzantine firing squad problem) and by Fischer, Lynch, and Merritt [14] (directly), that the Byzantine firing squad problem cannot be solved unless $n > 3f$.

All of the deterministic Byzantine agreement algorithms that we know of satisfy the following condition:

(A3) $MaxRounds(\mathcal{A})$ is finite.

Dolev, Reischuk and Strong [8] define a Byzantine agreement algorithm to be "immediate" if all processors reach agreement in the same round and to be "eventual" otherwise. Either type of algorithm can satisfy (A3) as long as there is some fixed limit on the number of rounds (over all input vectors and patterns of faulty behavior) for all processors to reach agreement.

If (A3) holds for $\mathcal{A}$, we say $\mathcal{A}$ is a *bounded Byzantine agreement algorithm.* In the remainder of the paper, we choose a fixed bounded Byzantine agreement algorithm $\mathcal{A}$ and let r be equal to $MaxRounds(\mathcal{A})$. It is easy to modify any such algorithm so that all its computations take exactly r rounds.

For convenience of exposition, we assume that $\mathcal{A}$ has been modified in this way, if necessary.

We use $\mathcal{A}$ to construct new algorithms $\mathcal{B}_P(\mathcal{A})$ and $\mathcal{B}_S(\mathcal{A})$ which solve the permissive and strict Byzantine firing squad problems, respectively. When $\mathcal{A}$ is understood from context, we simply refer to $\mathcal{B}_P$ and $\mathcal{B}_S$. Also, since $\mathcal{B}_P$ and $\mathcal{B}_S$ are very similar, it is convenient to use $\mathcal{B}$ to refer to them jointly. In algorithm $\mathcal{B}_P$, the reliable processors will all fire within at most r rounds after the first reliable processor receives a **START** signal. In algorithm $\mathcal{B}_S$ all reliable processors fire in at most r rounds after $f+1$ reliable processors have received a **START** signal.

We begin by describing algorithms $\mathcal{B}'_P(\mathcal{A})$ and $B'_S(\mathcal{A})$ which satisfy all the required conditions for a slightly more general model in which the processors are not required to be quiescent initially. (This model might admit simpler solutions to the Byzantine firing squad problems, but we will use it only to construct a specific algorithm which we can modify for use in our original model.) The basic idea of algorithm $\mathcal{B}'(\mathcal{A})$ is to simulate a copy of algorithm $\mathcal{A}$ beginning in each round. Each simulation runs for exactly r rounds, so that at any time, at most r are in progress. At each round t, each processor begins participating in a simulation of algorithm $\mathcal{A}$ in which it sends a value which is coded to mean 0: "Not *Ready*," or 1: "*Ready*." A processor is initially not *Ready*, becomes *Ready* upon the receipt of a **START** signal, and remains *Ready* thereafter. At round $t+r$ this simulation terminates, and a vector of values is computed. For $\mathcal{B}'_P$, all reliable processors fire if the vector is not all zero. For $\mathcal{B}'_S$, they fire if there are at least $f+1$ nonzero elements.

THEOREM 1. *Let $\mathcal{A}$ be an f-resilient bounded Byzantine agreement algorithm. Then algorithms $\mathcal{B}'_P(\mathcal{A})$ and $\mathcal{B}'_S(\mathcal{A})$ are f-resilient and satisfy conditions (C1) and (C2), and (C1) and (C2'), respectively. Also, $MaxRounds(\mathcal{B}'(\mathcal{A})) \leq MaxRounds(\mathcal{A})$, and $Bits(\mathcal{B}'(\mathcal{A})) \leq MaxRounds(\mathcal{A}) \times Bits(\mathcal{A})$, where $\mathcal{B}'$ stands for $\mathcal{B}'_P$ and for $\mathcal{B}'_S$.*

PROOF. The f-resiliency of $\mathcal{B}'_P$ and $\mathcal{B}'_S$ follows directly from the f-resiliency of $\mathcal{A}$. By assumption, $\mathcal{A}$ satisfies (A1), (A2), and (A3). By (A1), all reliable processors use the same vector to make their firing decisions in each round, so (C1) is satisfied (for both $\mathcal{B}'_P$ and $\mathcal{B}'_S$). By (A2), this vector will be nonzero for the simulation beginning with the round in which the first reliable processor receives a **START** signal, so (C2) is satisfied for $\mathcal{B}'_P$; since all simulations take r rounds, $MaxRounds(\mathcal{B}'_P) \leq MaxRounds(\mathcal{A})$.

Algorithm $\mathcal{B}'_S$ satisfies (C2'b), since if no reliable processor ever receives a **START** signal, then no vector can be computed with more than f 1's (by (A2)), so no reliable processor will fire. Condition (C2'a) is also satisfied since if $f+1$ reliable processors have received **START** signals by round t, then a vector will be computed by round $t+r$ which has at least $f+1$ 1's,

causing some reliable processor to fire and implying $MaxRounds(\mathcal{B}'_S(\mathcal{A})) \leq MaxRounds(\mathcal{A})$.

Recall that $M_i(\mathcal{A})$ is the set of messages sent in round i of algorithm $\mathcal{A}$. In any round, $\mathcal{B}'$ is simulating at most r staggered copies of $\mathcal{A}$, so the set of messages sent in any round of $\mathcal{B}'$ is a subset of $M_1(\mathcal{A}) \times \cdots \times M_r(\mathcal{A})$, and the number of bits sent is at most $n^2 \log_2(|M_1(\mathcal{A}) \times \cdots \times M_r(\mathcal{A})|) = n^2 \sum_{i=1}^{r} \log_2 |M_i(\mathcal{A})| = Bits(\mathcal{A})$. Therefore, $Bits(\mathcal{B}'(\mathcal{A})) \leq MaxRounds(\mathcal{B}') \times Bits(\mathcal{A}) = MaxRounds(\mathcal{A}) \times Bits(\mathcal{A})$, for both $\mathcal{B}'_P$ and $\mathcal{B}'_S$. □

We now show how to modify the $\mathcal{B}'$ algorithms to obtain $\mathcal{B}$ algorithms which meet the condition of initial quiescence required by our model. The difficulty is that when a reliable processor receives its first signal, some simulations might already be in progress. However, a great deal can be inferred about these computations.

Consider the specific computation of algorithm $\mathcal{A}$ in which all processors are reliable and each sends value 0. We call this computation the *zero computation* and refer to the messages that are sent as *zero messages.* These computations and their messages are completely defined and precomputable.

Any one-to-one encoding of meanings to messages can be used without affecting the behavior of an algorithm. We choose to code a special meaning into the null message. A null message is interpreted to consist of zero messages for each of the r simulations in progress. Now consider the particular computation of algorithm $\mathcal{B}'$ using this coding in which all processes are reliable and no **START** signal is received from the external source. After r rounds, all processors begin sending null messages and continue to do so throughout the remainder of the computation. Thus, just after r rounds, all processors are in quiescent states, according to our definition. We therefore define the $\mathcal{B}$ algorithms to be identical to the $\mathcal{B}'$ algorithms except that the initial states of the processors are chosen to be the states reached using algorithm $\mathcal{B}'$ after r rounds of the particular computation described above.

THEOREM 2. *Let $\mathcal{A}$ be an f-resilient bounded Byzantine agreement algorithm. Then algorithms $\mathcal{B}_P(\mathcal{A})$ are f-resilient solutions to the permissive and strict Byzantine firing squad problems, respectively. Furthermore, we have $MaxRounds(\mathcal{B}(\mathcal{A})) \leq MaxRounds(\mathcal{A})$, and $Bits(\mathcal{B}(\mathcal{A})) \leq MaxRounds(\mathcal{A}) \times Bits(\mathcal{A})$, where $\mathcal{B}$ stands for $\mathcal{B}_P$ and for $\mathcal{B}_S$.*

PROOF. By construction, all processors are quiescent in their initial states, so the initial condition required by the model is satisfied for both $\mathcal{B}_P$ and $\mathcal{B}_S$. The remaining conditions follow directly from Theorem 1. □

4. COMMUNICATION-EFFICIENT SOLUTIONS TO THE BYZANTINE FIRING SQUAD PROBLEMS

The solutions presented in the preceding section send up to r times as many bits as the chosen Byzantine agreement algorithm. Since it is known that $r > f$ [13], this is a significant increase in communication cost. We will show how to reduce the increase in cost to a constant factor. Our method requires at most one additional round for the permissive problem and two additional rounds for the strict problem.

We wish to define new algorithms, $\mathscr{C}_P(\mathscr{A})$ and $\mathscr{C}_S(\mathscr{A})$, which are similar to $\mathscr{B}_P(\mathscr{A})$ and $\mathscr{B}_S(\mathscr{A})$, respectively, but send smaller messages than the $\mathscr{B}$ algorithms. We begin by defining auxiliary algorithms $\mathscr{C}'_P(\mathscr{A})$ and $\mathscr{C}'_S(\mathscr{A})$ which are identical to $\mathscr{B}_P(\mathscr{A})$ and $\mathscr{B}_S(\mathscr{A})$ except in the way that *Ready* is defined and the condition under which firing occurs. The $\mathscr{C}'$ algorithms also use some preliminary messages to establish the *Ready* condition. We will show that all reliable processors become ready within a two-round interval. Reliable processors can thus deduce that the firing round will occur within a small, predictable interval. (Simulations which begin before any reliable processor becomes *Ready* cannot fire.) We take advantage of this by modifying the $\mathscr{C}'$ algorithms to give $\mathscr{C}$ algorithms which simulate only a fixed number of executions of algorithm $\mathscr{A}$.

In $\mathscr{C}'_P$, a processor becomes *Ready* upon receiving any signal, rather than only upon receiving a **START** signal as in $\mathscr{B}_P$. The firing condition is changed to "fire if there are at least $f + 1$ nonzero elements in the computed vector." The first time a reliable processor receives a signal and becomes *Ready*, it sends a special **GO** signal to every other processor. The **GO** signals are used to make sure that all reliable processors awaken within a two-round period.

In $\mathscr{C}'_S$, a processor sends the **GO** signal to every processor after receiving either a **START** signal or **GO** signals from $f + 1$ other processors (which implies that some reliable processor has received a **START** signal). Thus a **GO** signal is a claim that a **START** signal has been received by at least one reliable processor. A reliable processor sends **GO** signals only the first time such a condition occurs and sends only null messages otherwise until it becomes *Ready*. A reliable processor becomes *Ready* only after receiving **GO** signals from at least $2f + 1$ processors (perhaps including itself). The firing condition for $\mathscr{C}'_S$ is the same as for $\mathscr{C}'_P$: "Fire if there are at least $f + 1$ nonzero elements in the computed vector."

THEOREM 3. *Let $\mathscr{A}$ be an f-resilient bounded Byzantine agreement algorithm. Then $\mathscr{C}'_P(\mathscr{A})$ and $\mathscr{C}'_S(\mathscr{A})$ are f-resilient and satisfy conditions $(C1)$ and $(C2)$, and $(C1)$ and $(C2')$, respectively. Furthermore,*

$MaxRounds(\mathscr{C}'_P(\mathscr{A})) \leq MaxRounds(\mathscr{A}) + 1$ *and* $MaxRounds(\mathscr{C}'_S(\mathscr{A})) \leq MaxRounds(\mathscr{A}) + 2$.

PROOF. Since $\mathscr{C}'_P$ and $\mathscr{C}'_S$ simulate $\mathscr{A}$ and all processors use the same firing condition, both are f-resilient and (C1) is satisfied for both.

Let t be the round in which the first reliable processor receives a **START** message in $\mathscr{C}'_P$. Then at least $f+1$ reliable processors will be *Ready* by round $t+1$, and all reliable processors will fire no later than round $t+r+1$. Thus, $\mathscr{C}'_P$ satisfies (C2) and $MaxRounds(\mathscr{C}'_P(\mathscr{A})) \leq MaxRounds(\mathscr{A}) + 1$.

Let t be the round in which the $(f+1)$st reliable processor receives a **START** message in $\mathscr{C}'_S$. Then by round $t+1$ every reliable processor will have received **GO** signals from at least $f+1$ reliable processors, and by round $t+2$ every reliable processor will be *Ready*. (Here we use the fact that $\mathscr{A}$ is a Byzantine agreement algorithm which implies $n > 3f$ and thus at least $n - f \geq 2f + 1$ processors will have sent **GO** signals.) Thus, firing will occur by round $t+r+2$, and $\mathscr{C}'_S$ satisfies (C2'a) and $MaxRounds(\mathscr{B}'_S(\mathscr{A})) \leq MaxRounds(\mathscr{A}) + 2$. Finally, if no reliable processor receives a **START** signal, then no reliable processor will send a **GO** signal and no reliable processor will become *Ready*, hence firing will not occur and (C2'b) is satisfied. □

We now show how to construct $\mathscr{C}$ and $\mathscr{C}'$ by reducing the number of simulations of $\mathscr{A}$. We take advantage of the fact that all reliable processors become *Ready* within a period of at most two rounds, which is shown by the following lemma.

LEMMA 4. *In either $\mathscr{C}'_P$ or $\mathscr{C}'_S$, if a reliable processor first becomes Ready in round t, then every reliable processor becomes Ready in either round t or $t+1$.*

PROOF. In $\mathscr{C}'_P$, all reliable processors which do not become *Ready* in round t will receive a **GO** signal and become *Ready* in round $t+1$. In $\mathscr{C}'_S$, since some reliable processor received $2f+1$ **GO** signals by round t, every reliable processor must have received $f+1$ **GO** signals by round t. Thus, every reliable processor will send a **GO** signal in round t if not before, and every reliable processor will be *Ready* no later than round $t+1$. □

Let us denote the simulation which will terminate in round $t+r$ (and hence conceptually began in round t) by S_t. If simulation S_t would cause firing if carried to completion (i.e., the computed vector would have more than f nonzero values), then we say that S_t *will fire*. In our revision of $\mathscr{C}'$, a processor will not send the messages of all r simulations that are used in $\mathscr{C}'$. If processor p does send the messages of simulation S_t, then we say that p *participates in* simulation S_t.

Suppose processor p becomes *Ready* in round t. Then, by Lemma 4, p can deduce that S_{t+1} will fire since all reliable processors will be *Ready* no later than round $t+1$. Also, by Lemma 4, S_{t-2} will not fire since no reliable processor can have been *Ready* in that round, implying that at most f 1's will be in the vector computed. Computations S_{t-2}, S_{t-1}, S_t, and S_{t+1} are the only ones which p needs to consider.

Algorithm $\mathscr{C}$ is identical to algorithm $\mathscr{C}'$ except that if processor p becomes *Ready* in round t then p will participate only in simulations S_{t-2}, S_{t-1}, S_t, and S_{t+1}. Also, p will ignore the result of S_{t-2} and only act (fire or not) on the results of S_{t-1}, S_t, and S_{t+1}.

THEOREM 5. *Let $\mathscr{A}$ be an f-resilient bounded Byzantine agreement algorithm. Then algorithm $\mathscr{C}_P(\mathscr{A})$ and $\mathscr{C}_S(\mathscr{A})$ are f-resilient solutions to the permissive and strict Byzantine firing squad problems. For $\mathscr{C}_P$, MaxRounds$(\mathscr{C}_P(\mathscr{A})) \leq$ MaxRounds$(\mathscr{A})+1$ and for $\mathscr{C}_S$, MaxRounds$(\mathscr{C}_S(\mathscr{A})) \leq$ MaxRounds$(\mathscr{A})+2$. If $n>1$, then both Bits$(\mathscr{C}_P(\mathscr{A}))$ and Bits$(\mathscr{C}_2(\mathscr{A}))$ are $O($Bits$(\mathscr{A}))$.*

PROOF. Let C be a computation of either $\mathscr{C}_P$ or $\mathscr{C}_S$ and suppose that round t is the first round in which a reliable processor becomes *Ready*. By Lemma 4, all reliable processors become *Ready* in either round t or $t+1$. Call the former *early* and the latter *late*.

Early processors will participate in simulations S_{t-2}, S_{t-1}, S_t, and S_{t+1}. However, since they will not act on the result of S_{t-2}, the messages which are input to these simulations are irrelevant. Late processors will participate in simulations S_{t-1}, S_t, S_{t-1}, and S_{t+2}. Since all reliable processors participate in simulations S_{t-1}, S_t, and S_{t+1}, the resulting vectors that they compute must satisfy conditions (A1) and (A2). Since either S_t or S_{t+1} will fire, this implies that (C1) is satisfied by both $\mathscr{C}_P$ and $\mathscr{C}_S$ and that both $\mathscr{C}$ algorithms are f-resilient.

Since all reliable processors are *Ready* by round $t+1$, S_{t+1} is guaranteed to fire. By the definition of *Ready* for $\mathscr{C}_P$, condition (C2) is satisfied by $\mathscr{C}_P$, and firing will occur within $r+1$ rounds after a reliable processor receives a **START** signal (or any other signal), so MaxRounds$(\mathscr{C}_P) \leq$ MaxRounds$(\mathscr{A})+1$.

Let t' be the round in which the starting point is reached for $\mathscr{C}_S$. Then $t' \geq t-2$ since all processors will be ready within two rounds after $f+1$ reliable processors have received **START** signals. But if $t' = t-2$, then S_t will fire. In any case, S_{t+1} will fire, so MaxRounds$(\mathscr{C}_S) \leq$ MaxRounds$(\mathscr{A})+2$ and condition (C2'a) holds. On the other hand, if no reliable processor receives a **START** signal, then no reliable processor will send a **GO** signal and hence no reliable processor will become *Ready*, so (C2'b) holds.

For both $\mathscr{C}_P$ and $\mathscr{C}_S$ each processor participates in at most four simulations of algorithm $\mathscr{A}$. For the moment we ignore **GO** messages. A typical message will be of the form $M_k(\mathscr{A}) \times \cdots \times M_{k+3}(\mathscr{A})$, where we let $M_i(\mathscr{A}) = \{\varnothing\}$ if $i < 1$ or $i > r$. However, the early and late processors will have values of k differing by one for the same round. In addition, since firing can occur in either round $t + r$ or $t + r + 1$, our definition of the set of messages which a reliable processor can send in a round has the form

$$M(k) = M_k(\mathscr{A}) \times \cdots \times M_{k+3}(\mathscr{A}) \cup M_{k-1}(\mathscr{A}) \times \cdots \times M_{k+2}(\mathscr{A}) \cup M_{k-2}(\mathscr{A}) \times \cdots \times M_{k+1}(\mathscr{A}).$$

Now let $G = \{\mathbf{GO}, \varnothing\}$. Reliable processors only send **GO** messages in rounds t and $t + 1$ of $\mathscr{C}_P$. Depending on whether S_t or S_{t+1} fires, round t will be called either round 1 or round 2, so G needs to be added only to $M_1(\mathscr{C}_P)$, $M_2(\mathscr{C}_P)$, and $M_3(\mathscr{C}_P)$. Thus we have $M_i(\mathscr{C}_P) \subseteq M(i-1)$ for $4 \le i \le r+1$ and $M_i(\mathscr{C}_P) \subseteq G \times M(i-1)$ for $1 \le i \le 3$. A similar analysis for $\mathscr{C}_S$ shows that $M_i(\mathscr{C}_S) \subseteq M(i-2)$ for $5 \le i \le r+2$ and $M_i(\mathscr{C}_S) \subseteq G \times M(i-2)$ for $1 \le i \le 4$. Now since $|M(k)| \le |M_{k-2}(\mathscr{A}) \times \cdots \times M_{k+3}|$, we have $Bits(\mathscr{C}_P) \le 3n^2 + 6 \cdot Bits(\mathscr{A})$ and $Bits(\mathscr{C}_S) \le 4n^2 + 6 \cdot Bits(\mathscr{A})$, where the values "3" and "4" come from the **GO** messages. As a consequence of our definition of *Bits* and the fact that some communication must take place to reach agreement even when $f = 0$, we have $Bits(\mathscr{A}) \ge n^2$ when $n > 1$. Therefore, $Bits(\mathscr{C}_P)$ is $O(Bits(\mathscr{A}))$ and $Bits(\mathscr{C}_S)$ is $O(Bits(\mathscr{A}))$. □

5. CONCLUSIONS

We have described and solved a new problem in distributed computing, which we call the Byzantine firing squad problem. It is an abstraction of synchronization problems that arise in several ways in distributed computing, such as real-time processing and synchronizing starting and termination of algorithms. It is an abstract problem of interest in its own right and deserves further study.

Our solutions are presented in a way that we believe is particularly easy to understand, via simple reductions from other distributed algorithms, in this case Byzantine agreement algorithms. If the reader has attempted to solve the problem directly, without using such a reduction, he has probably found the task to be not entirely straightforward. The separation of concerns provided by reducing the problems to Byzantine agreement has made the problem much simpler. In general, we believe that the idea of reducing certain distributed problems to others can provide a powerful means of organizing the field of distributed algorithms. A partial list of other papers

about distributed consensus problems that use this approach include Bracha [2], Coan's main thesis result [3, 4], Dwork, Lynch, and Stockmeyer [10], Dwork and Skeen [11], Lamport, Shostak, and Pease [18], Srikanth and Toueg [22], and Turpin and Coan [23].

There is additional work to be done. It would be interesting, for example, to examine solutions to this and related problems in "partially synchronous" models of distributed computing, e.g., models in which processors do not have a common notion of rounds but rather have real-time clocks (possibly with small drift rates) and in which messages can take variable amounts of time. Is it possible to convert Byzantine agreement algorithms for partially synchronous models to Byzantine firing squad algorithms for the same models? Some particular partially synchronous models have been considered in Coan, Dolev, Dwork, and Stockmeyer [5], but there are others still to be studied.

ACKNOWLEDGMENT

This work was supported in part by the Army Research Office under contract DAAG29-84-K-0058, the Defense Advanced Research Projects Agency (DARPA) under grant N00014-83-K-0125, the Office of Naval Research under contract N00014-85-K-0168, and the National Science Foundation under grant 8302391-A01-DCR.

We wish to thank Mike Fischer for suggestions on the presentation of these ideas and Brian Coan, John Franco, and the referees for their comments and criticism.

REFERENCES

1. Ben-Or M: Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). *Proc. 2nd Ann. ACM Symposium on Principles of Distributed Computing, New York, 1983* pp. 27–44.
2. Bracha G: An asynchronous $[(n-1)/3]$-resilient consensus protocol. *Proc. 3rd Ann. ACM Symposium on Principles of Distributed Computing, New York, 1984* pp. 154–162.
3. Coan B: Achieving consensus in fault-tolerant distributed computer systems: Protocols, lower bounds, and simulations. Ph.D. thesis, MIT, 1987 (forthcoming).
4. Coan B: A communication-efficient canonical form for fault-tolerant distributed protocols. *Proc. 5th Ann. ACM Symposium on Principles of Distributed Computing, New York, 1986* pp. 63–72.
5. Coan B, Dolev D, Dwork C, Stockmeyer L: The distributed firing squad problem (preliminary version). *Proc. 17th Ann. ACM Symposium on the Theory of Computing, New York, 1985* pp. 335–345.
6. Cook SA: The complexity of theorem proving procedures. *Proc. 3rd Ann. ACM Symposium on Theory of Computing, New York, 1971* pp. 151–158.
7. Dolev D, Lynch N, Pinter S, Stark E, Weihl W: Reaching approximate agreement in the presence of faults. *Proc. 3rd Ann. IEEE Symposium on Reliability in Distributed Software and Database Systems, New York, 1983* pp. 145–154. (Revised version to appear in *JACM*, 33(3):449–516, July 1986.

8. Dolev D, Reischuk R, Strong HR: "Eventual" is earlier than "immediate." *Proc. 23rd Ann. IEEE Symposium on Foundations of Computer Science, New York, 1982* pp. 196–202.
9. Dolev D, Strong HR: Authenticated algorithms for Byzantine agreement. *SIAM J. Comp.* 12:656–666, 1983.
10. Dwork C, Lynch N, Stockmeyer L: Consensus in the presence of partial synchrony. *Proc. 3rd Ann. ACM Symposium on Principles of Distributed Computing, New York, 1984* pp. 103–118.
11. Dwork C, Skeen D: Patterns of communication in consensus protocols. *Proc. 3rd Ann. ACM Symposium on Principles of Distributed Computing, New York, 1984* pp. 143–153.
12. Fischer MJ: The consensus problem in unreliable distributed systems (a brief survey). Pp. 127–140 in Karpinsky M (ed.), *Foundations of Computation Theory.* Lecture Notes in Computer Science No. 158. Springer-Verlag, New York, 1983.
13. Fischer MJ, Lynch NA: A lower bound for the time to assure interactive consistency. *Information Processing Letters* 14(4):183–186, June 1982.
14. Fischer MJ, Lynch NA, Merritt M: Easy impossibility proofs for distributed consensus problems. *Proc. 4th Ann. ACM Symposium on Principles of Distributed Computing, New York, 1985* pp. 59–70.
15. Hamming RW: *Coding and Information Theory*, 2nd ed. Prentice-Hall, Englewood Cliffs, NJ, 1986.
16. Karp RM: Reducibility among combinatorial problems. Pp. 85–103 in Miller RE, Thatcher JW (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 1972.
17. Lamport L: The weak Byzantine generals problem. *J. ACM* 30(3):668–676, July 1983.
18. Lamport L, Shostak R, Pease M: The Byzantine generals problem. *ACM Trans. Programming Languages and Systems* 4(3):382–401, July 1982.
19. Moore EF: The firing squad synchronization problem. Pp. 213–214 in Moore EF (ed.), *Sequential Machines: Selected Papers.* Addison-Wesley, Reading, MA, 1964.
20. Nishitani Y, Honda N: The firing squad synchronization problem for graphs. *Theoret. Comp. Sci.* 14:39–61, 1981.
21. Pease M, Shostak R, Lamport L: Reaching agreement in the presence of faults. *J. ACM* 27(2):228–234, April 1980.
22. Srikanth TK, Toueg S: Simulating authenticated broadcasts to derive simple fault tolerant algorithms. Tech. Report 84-623, Department of Computer Science, Cornell University, July 1984.
23. Turpin R, Coan B: Extending binary Byzantine agreement to multivalued Byzantine agreement. *Information Processing Letters* 18(2):73–76, February 1984.