MIT/LCS/TM-275

THE BYZANTINE FIRING SQUAD PROBLEM

James E. Burns

Nancy A. Lynch

April 1985

THE BYZANTINE FIRING SQUAD PROBLEM

JAMES E. BURNS
Indiana Unveristy
and
NANCY A. LYNCH
Massachusetts Institute of Technology

A new problem, the Byzantine Firing Squad problem, is defined and solved in two versions, Permissive and Strict. Both problems provide for synchronization of 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 might transmit r times as many bits, where r is the number of rounds used. Additional solutions are provided which use at most one (Permissive) or two (Strict) additional rounds and send at most n^2 bits plus four times the number of bits sent 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*

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

---

This work was supported in part by the following grants: ARO DAAG29-84-K-0058, DARPA N00014-83-K-0125, and NSF 8302391-A01-DCR.

Author's addresses: J.E. Burns, Computer Science Department, 101 Lindley Hall, Indiana University, Bloomington, Indiana 47401; N.A. Lynch, 545 Technology Square NE-43-525, Cambridge, Massachusetts 02139.

assume the global availability of a "current time." A solution to this synchronisation 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], approximate agreement [3]), 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 [9]. In the original problem, a finite 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 [10]). In our problem, the finite state machines are replaced by (not necessarily finite) automata connected by a complete network.

The Byzantine Generals problem was first proposed by Pease, Shostak and Lamport [11], although it did not receive that name until a later work appeared [8]. For a recent bibliography of work on the problem see Fischer [5]. 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 at the end of the algorithm the following conditions hold:

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

(A2) *Validity:* If processor i is reliable, then i^{th} 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. We will use f for the number of faulty processors and n for the total number of processors for the remainder of the paper.

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 special START signals to some of the processors (possibly at different times). Within a finite number of rounds, all of the reliable processors must simultaneously send special FIRE signals, 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, but might require sending r times as many bits as sent by the Byzantine Agreement algorithm. We show in section 4 how to reduce this to only n^2 bits plus four times as many bits as sent by Byzantine Agreement 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 give an example of a reduction between distributed problems (it would be nice to have more such examples). 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 other processor and an external source, sends a message to every other processor and an external destination, and goes to a new 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 that processors can append unforgeable signatures to their messages. For results on the Byzantine Firing Squad problem with signatures refer to Coan, Dolev, Dwork and Stockmeyer [2].

In a synchronous system, information can be conveyed by the absence of a signal as well as by an explicit signal. Thus, we distinguish a particular message, called the *null message*; 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 processors be quiescent in their initial states. Initial quiescence guarantees that no signals will be sent by any reliable processor until the external source or 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. The only signal which is ever sent to the external destination is a special FIRE signal, indicating that the processor has fired.

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.)

An f-resilient *Permissive Byzantine Firing Squad algorithm* must satisfy the following conditions whenever the number of faulty processors does not exceed f:

(C1) *Agreement:* If any reliable processor sends a FIRE message in some round, then all reliable processors send a FIRE message in that round.

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

An f-resilient *Strict Byzantine Firing Squad algorithm* will satisfy (C1) and the following additional condition whenever the number of faulty processors does not exceed f:

(C2') *Strict Validity:*

a) If at least $f + 1$ reliable processors receive a START signal, then some reliable processor eventually sends a FIRE message.

b) If any reliable processor sends a FIRE message, 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 these might be unbounded. We also wish to avoid charging for "preliminary rounds" which are caused by faulty processors and do not lead to termination. We therefore introduce the concept of "measured portion of a computation."

Let A be an algorithm. If A is a Byzantine Agreement algorithm, then the entire computation from initial state to termination is measured. If A is a Permissive Byzantine Firing Squad algorithm, then the measured portion of the computation is from the first reception of a START message by a reliable processor until a reliable processor fires. If A is a Strict Byzantine Firing Squad algorithm, then the measured portion of the computation is from the round in which the $f + 1^{st}$ reliable processor receives a START signal until a reliable processor fires. Now we can define our time measure, $Rounds(A)$ simply as the worst case number of rounds in the measured portion of the computation. Many communication measures are possible. We shall use $Bits(A)$ as the worst case total number of bits sent by all the reliable processors in the measured portion of the computation. We assume that variable length messages are used so that the shortest, non-null message that can be sent costs one bit.

3. TIME EFFICIENT SOLUTIONS TO 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 *Rounds* 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 [8], the Byzantine Firing Squad problem can also be solved whenever $n > 3f$. It has also been shown [2], by reducing Lamport's Weak Byzantine Agreement problem [7] to the Byzantine Firing Squad problem, that the latter problem cannot be solved unless $n > 3f$.

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

(A3) *Rounds*($\mathcal{A}$) is bounded.

In this case, we say $\mathcal{A}$ is a Bounded Byzantine Agreement algorithm. (Note that (A3) need not imply that $\mathcal{A}$ is "immediate" as defined by Dolev, *et al.* [4].) In the remainder of the paper, we will let *Rounds*($\mathcal{A}$) = r.

Let $\mathcal{A}$ be a Bounded Byzantine Agreement algorithm. 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 problem, 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 $\mathcal{B}'_S(\mathcal{A})$ which satisfy all the required conditions for a slightly more general model in which the processors are not required to be quiescent initially. The basic idea of algorithm $\mathcal{B}'(\mathcal{A})$ is to simulate a copy of algorithm $\mathcal{A}$ starting in each round. Each simulation runs for

exactly r rounds, so that at any time only r are in progress. The messages from the r active simulations of algorithm A are coded into a single message for algorithm B' in a straightforward way. At each time t, each processor begins participating in a simulation of algorithm A in which it sends a value which is coded to mean 0: "Not *Ready*," or 1: "*Ready*." A processor becomes *Ready* upon the receipt of a START signal and remains *Ready* thereafter. At time $t + r$ this simulation terminates, and a vector of values is computed. For B'_P, all reliable processors fire if the vector is not all zero. For B'_S, they fire if there are at least $f + 1$ non-zero elements.

Theorem 1. *Let A be an f-resilient Bounded Byzantine Agreement algorithm. Then algorithms $B'_P(A)$ and $B'_S(A)$ are f-resilient and satisfy conditions (C1) and (C2), and (C1) and (C2'), respectively. Also, $Rounds(B'_P(A)) = Rounds(A)$ and $Bits(B'_P(A)) \leq Rounds(A) \times Bits(A)$ hold for B'_P, while $Rounds(B'_S(A)) = Rounds(A)$ and $Bits(B'_S(A)) \leq Rounds(A) \times Bits(A)$ hold for B'_S.*

Proof: The f-resilency of B'_P and B'_S follow directly from the f-resiliency of A. By assumption, 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 B'_P and B'_S). By (A2), this vector will be non-zero for the simulation beginning with the round in which the first reliable processor receives a START signal, so (C2) is satisfied for B'_P; furthermore, by (A3), firing occurs within r rounds after the first reception of a START signal by a reliable processor, so $Rounds(B'_P(A)) = Rounds(A)$.

Algorithm 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 ones (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 $f+1$ ones, causing some reliable processor to fire. Also, firing must occur within r rounds after $f+1$ reliable processors have received a START signal, $Rounds(\mathcal{B}'_S(\mathcal{A})) = Rounds(\mathcal{A})$.

The composite message transmitted by a reliable processor in one round includes exactly one message from each round of a simulation of $\mathcal{A}$, so the number of bits sent by all reliable processors in any round (using a suitable encoding) is bounded by $Bits(\mathcal{A})$. Since at most r rounds occur in the measured portion of the computation, $Bits(\mathcal{B}'(\mathcal{A})) \leq Rounds(\mathcal{A}) \times Bits(\mathcal{A})$, 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. At this point, all processors are quiescent, 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 A be an f-resilient Bounded Byzantine Agreement algorithm. Then algorithms $B_P(A)$ and $B_S(A)$ are f-resilient solutions to the Permissive and Strict Byzantine Firing Squad problems, respectively. Furthermore, we have $Rounds(B_P(A)) = Rounds(B_S(A)) = Rounds(A)$, and both $Bits(B_P(A))$ and $Bits(B_S(A))$ are less than or equal to $Rounds(A) \times Bits(A)$.*

Proof: By construction, all processors are quiescent in their initial states, so the initial condition required by the model is satisfied both for B_P and 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 $r > f$ [6], this is a significant increase in communication cost. Various coding tricks (such as using short codes for expected messages and taking advantage of knowledge of which processors are faulty when possible) could be used to reduce this cost. However, we will show how to reduce the increase in cost to a constant factor (and an additional n^2 bits) without any sophisticated coding. 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, $C_P(A)$ and $C_S(A)$, which are similar to $B_P(A)$ and $B_S(A)$, respectively, but send many fewer bits than A. We begin by defining auxiliary algorithms $C'_P(A)$ and $C'_S(A)$ which are identical to $B_P(A)$ and $B_S(A)$ except in the way that *Ready* is defined and the condition under which firing

occurs. The C' algorithms also use some preliminary messages to establish the *Ready* condition. We will then show how to modify the C' algorithms to get the C algorithms.

In C'_P, a processor becomes *Ready* upon receiving any signal, rather than only upon receiving a START signal as in B_P. The firing condition is changed to "fire if there are at least $f+1$ non-zero 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. At most n^2 GO signals will be sent.

In 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). 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 C'_S is the same as for C'_P: "fire if there are at least $f+1$ non-zero elements in the computed vector."

Theorem 3. *Let A be an f-resilient Bounded Byzantine Agreement algorithm. Then $C'_P(A)$ and $C'_S(A)$ are f-resilient and satisfy conditions (C1) and (C2), and (C1) and (C2'), respectively. Furthermore, $Rounds(C'_P(A)) \leq Rounds(A) + 1$ and $Rounds(C'_S(A)) \leq Rounds(A) + 2$.*

Proof: Since C'_P and C'_S simulate 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 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, C'_P satisfies

(C2) and $Rounds(C'_P(A)) \leq Rounds(A) + 1$.

Let t be the round in which the $f+1^{st}$ processor receives a START message in C'_S. Then by round $t+1$ every reliable processor will have received GO signals from at least $f+1$ processors, and by round $t+2$ every reliable processor will be *Ready* (since at least $2f+1$ processors will have sent GO signals). Thus, firing will occur by round $t+r+2$, and C'_S satisfies (C2'a) and $Rounds(B'_S(A)) \leq Rounds(A) + 2$. 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 derive C from C' by reducing the number of simulations of A. We take advantage of the fact that all reliable processors become *Ready* within a time period of at most two rounds, which is shown by the following lemma.

Lemma 4. *In either C_P or C_S, if a reliable processor becomes Ready in round t then all reliable processors become Ready in either rounds t and $t-1$ or in rounds t and $t+1$.*

Proof: Let t be the first round in which a reliable processor becomes *Ready*. In C_P, all reliable processors which are not *Ready* in round t will receive a GO signal and become *Ready* in round $t+1$. In 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 will have more than f non-zero values),

then we say that S_t *will fire*. In our revision of C', a processor will not send the messages of all r simulations that are used in 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 ones 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 C is identical to algorithm 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}. There is no difficulty in coding the four (at most) messages of algorithm A so that each receiving processor can match them up with the appropriate simulations.

Theorem 5. *Let A be an f-resilient Bounded Byzantine Agreement algorithm. Then algorithm $C_P(A)$ and $C_S(A)$ are f-resilient solutions to the Permissive and Strict Byzantine Firing Squad problems. For C_P, $Rounds(C_P(A)) \leq Rounds(A) + 1$ and for C_S, $Rounds(C_S(A)) \leq Rounds(A) + 2$. Both $Bits(C_P(A))$ and $Bits(C_S(A))$ are at most $n^2 + 4 \times Bits(A)$.*

Proof: Suppose that round t is the first round in which a reliable processor becomes *Ready*. (If no reliable processor becomes *Ready*, then the theorem is vacuously true.) For C_S, round t is also the first round of the measured portion of the computation. For C_P, the first round of the measured portion of the computation is round $t-1$. By Lemma 4, all reliable processors awaken 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). This implies that (C1) is satisfied by both C_P and C_S and that both 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 C_P, condition (C2) is satisfied by C_P, and firing will occur within $r+1$ rounds after a reliable processor receives a START signal (or any other signal), so $Rounds(C_P) \leq Rounds(A) + 1$.

In C_S, if $f+1$ reliable processors receive a START signal in round t', then some reliable will become *Ready* by round $t'+1$. By the foregoing discussion, some reliable processor will fire by round $t'+r+2$, so condition (C2'a) holds and $Rounds(C_S) \leq Rounds(A)+2$. 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.

Each processor participates in at most four simulations of algorithm A. There is no difficulty in coding the messages of these simulations to use at most four times the number of bits used by algorithm A. The GO messages can usually "piggyback" at no cost in C_S and sometimes do so in C_P since any non-null message will do to communicate a GO signal. Otherwise a single bit will suffice to send a GO signal, so, $Bits(C_P(A)) \leq n^2 + 4 \times Bits(A)$, and $Bits(C_S(A)) \leq n^2 + 4 \times Bits(A)$. □

Acknowledgment: We wish to thank Mike Fischer for suggestions on the presentation of these ideas, and Brian Coan and John Franco for their criticism of early drafts of this paper.

REFERENCES

1. BEN-OR, M. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing* (Montreal, Quebec, Canada, August 17-19, 1983). ACM, New York, 1983, pp. 27–44.
2. COAN, B., DOLEV, D., DWORK, C. AND STOCKMEYER, L. The distributed firing squad problem. To appear in ACM Symposium on the Theory of Computing, 1985.
3. DOLEV, D., LYNCH, N., PINTER, S., STARK, E., AND WEIHL, W. Reaching approximate agreement in the presence of faults. In *Proceedings of 3rd Annual IEEE Symposium on Reliability in Distributed Software and Database Systems*. IEEE, New York, 1983.
4. DOLEV, D., REISCHUK, R., AND STRONG, H.R. 'Eventual' is earlier than 'immediate.' In *Proceedings 23rd Annual Symposium on Foundations of Computer Science* (Chicago, IL, November 3–5, 1982). IEEE, New York, 1982, pp. 196–202.
5. FISCHER, M.J. The consensus problem in unreliable distributed systems (A brief survey). YALEU/DCS/RR-273, Yale University, New Haven, CT, June 1983.
6. FISCHER, M.J. AND LYNCH, N.A. A lower bound for the time to assure interactive consistency. *Information Processing Letters 14*, 4, (June 1982), pp. 183–186.
7. LAMPORT, L. The weak Byzantine generals problem. *Journal of the ACM 30*, 3 (July 1983), 668–676.
8. LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Bysantine generals problem. *ACM Transactions on Programming Languages and Systems 4*, 3

(July 1982), 382–401. (Also see: The Byzantine generals problem, Tech. Report 54, Computer Science Lab., SRI International, 1980.)

9. MOORE, E.F. The firing squad synchronization problem. In *Sequential Machines, Selected Papers*. MOORE, E.F., Ed., Addison-Wesley, Reading, MA, 1964, pp. 213–214.
10. NISHITANI, Y., AND HONDA, N. The firing squad synchronization problem for graphs. *Theoretical Computer Science 14*, (1981), 39–61.
11. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *Journal of the ACM 27*, 2 (Apr. 1980), 228–234.