A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols

Brian A. Coan

Massachusetts Institute of Technology

Abstract: Many fault-tolerant distributed protocols are known. Some of these require a large (exponential) amount of communication. We present a general simulation of any synchronous fault-tolerant consensus protocol by a communication-efficient protocol. An important corollary of the simulation technique is a new communication-efficient Byzantine agreement protocol that uses about half the number of rounds required by the best previously-known communication-efficient Byzantine agreement protocol. Our new protocol approaches the known lower bound for rounds to within a small factor arbitrarily close to 1. The only known protocols which achieve the lower bound for rounds use an exponential amount of communication.

1. Introduction

For almost the past ten years, the task of achieving consensus in a fault-tolerant distributed computer system has been recognized as a fundamental problem in distributed computing. Protocols have been designed to solve many consensus problems including the agreement problem (see [13] and [15]), the approximate agreement problem (see [7] and [9]), the crusader agreement problem (see [5]), the firing squad problem (see [2] and [4]), and the weak agreement problem (see [12]). These protocols operate in a variety of fault models including Byzantine, authenticated Byzantine, failure-by-omission, and fail-stop. Some protocols for the Byzantine fault model require a large (exponential) amount of communication, for example, the agreement protocol of Lamport *et al.* [13] and the approximate agreement protocol of Fekete [9].

We give general upper bounds on the number of bits of communication needed for any synchronous simultaneous-start non-cryptographic consensus protocol. More precisely, we show how to transform an arbitrary consensus protocol into a canonical-form protocol; the new protocol solves the same problem as the original, but only uses an amount of communication that is polynomial in the number of processors and the number of rounds of message exchange. To achieve this small communication cost, the new protocol incurs an increase in running time (*i.e.*, rounds of message exchange). There is a tradeoff between the number of rounds and the degree of the polynomial bounding the communication. The value of this tradeoff is determined by a numerical parameter to the transformation. For any $\epsilon > 0$ the transformation can produce a canonical-form protocol that increases the number of rounds of the original protocol by a factor of $1 + \epsilon$ and that uses $O(r \cdot n^{\lceil 2/\epsilon \rceil + 3} \cdot \log |V|)$ bits of communication where n is the number of processors, r is the number of rounds, and V is the set of possible inputs to the original protocol. Throughout this paper we let n be the number of processors in the system and we let t be an upper bound on the number of processor faults that a protocol need tolerate.

Our transformation was developed for the Byzantine fault model. In more benign fault models like failure-by-omission and fail-stop there is a simple extension of our transformation that causes no increase in the number of rounds. Because the most interesting applications of our transformation are in the Byzantine fault model, we will restrict our attention to that model in the remainder of this paper.

As a corollary to the results in the Byzantine fault model, we obtain a major new result about the communication requirements of Byzantine agreement. The earliest Byzantine agreement protocols [13] used exponential communication and $t+1$ rounds; $t+1$ is the known lower bound on rounds [10]. Subsequently, improved protocols yielded

why does the number of rounds only depend on the number of faults and not on message orderings

This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125, by the National Science Foundation under Grant DCR-83-02391, by the Office of Army Research under Contract DAAG29-84-K-0058, and by the Office of Naval Research under Contract N00014-85-K-0168.

polynomial communication using about $2t$ rounds (see [6], [8], [14], and [18]). An open question among researchers in this area for the past few years has been whether there are any protocols that simultaneously use fewer than $2t$ rounds and polynomial communication. We obtain an interesting answer to this question. For any $\epsilon > 0$ there is a protocol that uses $(1+\epsilon)(t+1)$ rounds and polynomial communication. We obtain this result by applying our transformation to the communication-inefficient $(t+1)$-round protocol of Lamport *et al.* Another use for our transformation is improving the communication complexity of a new approximate Byzantine agreement protocol of Fekete [9]. His protocol has the optimal convergence rate for any multi-round approximate agreement protocol, but requires exponential communication. Using our new technique, we can transform his protocol into a polynomial-communication protocol with a near optimal convergence rate.

Our new method for transforming an arbitrary consensus protocol, say $\mathcal{P}$, into a communication-efficient canonical-form protocol is a two-step process. The protocol $\mathcal{P}$ is first transformed into a *full-information* protocol; the resulting protocol is then transformed into a *compact full-information* protocol. A full-information protocol is a well known [10] communication-inefficient canonical-form protocol in which each processor, at each round, broadcasts its entire state, receives one message from each processor, and forms its new state as the ordered collection of all messages received. To transform $\mathcal{P}$ into a full-information protocol it suffices to find a decision rule for each processor to apply locally to its state. It is not necessary to devise message generation rules because all full-information protocols have the same rule: at each round each processor broadcasts its entire state. Pease *et al.* [15] show that any consensus protocol $\mathcal{P}$ can be transformed into a full-information protocol, by showing how to find the decision rule that corresponds to $\mathcal{P}$. This first transformation of $\mathcal{P}$ gives a full-information canonical-form protocol that uses the same number of rounds as $\mathcal{P}$ and exponential communication. We want to do much better than this in communication.

In the second step we further transform $\mathcal{P}$ into a compact full-information protocol that uses only polynomial communication. Because we have already shown how to put $\mathcal{P}$ in the form of a full-information protocol, it is sufficient to show how to simulate the message exchange of a full-information protocol using polynomial communication. The compact full-information protocol consists of this simulation together with the same decision rule used in the full-information protocol.

The heart of our technique is our new method for efficiently simulating the message exchange portion of a full-information protocol. We do this by using data compression techniques to condense the information being sent around. Messages are compressed by the sender and expanded by the recipient. In parallel with the rest of the compact full-information protocol, each processor at each round computes an expansion function that it can apply to incoming compressed messages to obtain the full message text. It is necessary that all correct processors be able to consistently expand any message sent by a correct processor. This consistency requirement seems difficult to achieve in the presence of faults, because it requires that the correct processors agree on how to carry out the compression and expansion. Such agreement might, at first, seem to require Byzantine agreement or some other time-costly protocol. We overcome this difficulty by using a new, different form of agreement that we call avalanche agreement. The difference between avalanche agreement and Byzantine agreement is explained in Section 4. Using an avalanche agreement protocol to agree on their expansion function enables the correct processors to achieve a sufficient level of agreement at a cost that we can afford.

One limitation of our technique is that it can use a large amount of local computing resources. A complete reconstruction of the local state of processors in a full-information protocol requires exponential space and time. It is straightforward to devise an efficient data representation that requires only a polynomial amount of space; however, the question of how much time is required to reach a decision remains open.

In Section 2 we review the definition of the Byzantine agreement problem. In Section 3 we give our definition of simulation. In Section 4 we define the avalanche agreement problem and give a protocol that solves the problem. In Section 5 we present a compact full-information protocol — a communication-efficient protocol that simulates a full-information protocol.

2. The Byzantine Agreement Problem

A synchronous Byzantine agreement protocol is run by a distributed system of n processors, at most t of which may fail. Communication is over a network that is fully connected and reliable. The computation takes place in a series of rounds. In each round the correct processors first send messages, then receive messages and finally make a local state change based on the messages received and their state. Correct processors send messages according to their programs. Failed processors can send arbitrary messages.

Each processor starts the protocol with an input value, v, from a fixed set of legal inputs, V. The goal is that after some number of rounds each correct processor will irrevocably decide on an element of V as its answer. There are two conditions that the correct processors must satisfy.

- *Agreement condition:* All correct processors reach the same decision.
- *Validity condition:* If all correct processors start the protocol with input v then v is the decision of all of the correct processors.

3. Simulations

In this section we give our definition of one protocol simulating another and we characterize some important properties of protocol behavior that are preserved

by our simulations. We take the first step toward showing that any consensus protocol can be simulated by a communications-efficient protocol, that is, we show that the full-information protocol can simulate any consensus protocol. After this section, the remainder of the paper is devoted to showing how to simulate the full-information protocol using a particular communications-efficient protocol which we call the compact full-information protocol.

3.1 Definitions

Following Lynch, Fischer, and Fowler [14] we model a consensus protocol as a synchronous system of automata. We find it convenient to introduce this formalism in order to discuss simulations. Later, when we give our protocols we will use a higher level language. The mapping back to automata is straightforward. A protocol $\mathcal{P}$ is described by the following.

- V is the set of input values.
- Q is the set of processor states.
- L is the set of messages.
- $\mu_{p,q} : Q \rightarrow L$, for $p, q \in \{1, \ldots, n\}$, is the message generation function for messages sent from processor p to processor q.
- $\delta_p : L^n \rightarrow Q$, for $p \in \{1, \ldots, n\}$, is the state transition function for processor p. (The prior state of processor p is omitted from the domain of δ_p because it would be redundant. Processor p can send any required information in a message to itself.)
- $\gamma_p : Q \rightarrow \{\perp\} \cup V$, for $p \in \{1, \ldots, n\}$, is the decision function for processor p.

An element of Q is identified with each element of V. These are the initial states. The decision of processor p in some execution of protocol $\mathcal{P}$ is the first non-$\perp$ value of $\gamma_p(s_i)$ where s_i is the round i state of processor p. After processor p has decided, future values of γ_p are ignored.

A round consists of sending messages, receiving messages, and making a local state change. Each processor starts in the initial state corresponding to its input value. In any execution of protocol $\mathcal{P}$ a correct processor sends according to its message generation function and a faulty processor sends arbitrary messages from L. An *execution* of protocol $\mathcal{P}$ is a 4-tuple (k, F, I, M) where k is the number of rounds in the execution, F is the set of faulty processors, I is a vector of inputs for the processors, and M is the set of messages (with their origin, destination, and round sent) sent by faulty processors in all rounds up to round k. In this paper we restrict our attention to executions in which the number of faulty processors is less than t. Let processor p be correct in execution E. We let $state(p, i, E)$ denote the round i state of processor p in execution E.

Let $\mathcal{P}$ and $\mathcal{P}'$ be protocols with the same set of possible inputs. Protocol $\mathcal{P}'$ *simulates* protocol $\mathcal{P}$ if there is a non-decreasing function r from the natural numbers onto the natural numbers and a set of functions f_p for $p \in \{1, \ldots, n\}$ from the processor states of $\mathcal{P}'$ to the processor states of $\mathcal{P}$ such that for any execution $E' = (k, F, I, M')$ of $\mathcal{P}'$ there is an execution $E = (r(k), F, I, M)$ of $\mathcal{P}$ such that for any correct processor p and for any i (where $1 \leq i \leq k$) it is the case that $f_p(state(p, i, E')) = state(p, r(i), E)$. We say that the f_p are the *simulation functions* and that r is the *scaling function*.

Execution E of protocol $\mathcal{P}$ is a *deciding execution* if all processors that are correct in E have decided. Protocol $\mathcal{P}$ *terminates* if there is some k such that any k-round execution of $\mathcal{P}$ is a deciding execution. If $E = (k, F, I, M)$ is a deciding execution of $\mathcal{P}$ then $ans(E)$ is defined to be the n-tuple whose p^{th} component is $state(p, k, E)$ if processor p is correct in E and $\perp$ otherwise.

Predicate C is a *correctness predicate* if its domain is $(V \cup \{\perp\})^n \times 2^{\{1,\ldots,n\}} \times V^n$. Protocol $\mathcal{P}$ satisfies correctness predicate C if for any deciding execution $E = (k, F, I, M)$ the value of $C(ans(E), F, I)$ is true. We observe that the correctness conditions for Byzantine agreement and approximate agreement can be formulated as correctness predicates.

Theorem 1: *If protocol $\mathcal{P}'$ simulates protocol $\mathcal{P}$ with simulation functions f_p for $p \in \{1, \ldots, n\}$, $\mathcal{P}$ has decision functions γ_p for $p \in \{1, \ldots, n\}$, and $\mathcal{P}'$ has decision functions $\gamma'_p(s) = \gamma_p(f_p(s))$, for $p \in \{1, \ldots, n\}$ then the following two conditions hold.*

(1) If protocol $\mathcal{P}$ terminates then so does protocol $\mathcal{P}'$.

(2) If protocol $\mathcal{P}$ satisfies some correctness predicate C then so does protocol $\mathcal{P}'$.

Proof: Let r be the scaling function of the simulation.

Condition (1): Because protocol $\mathcal{P}$ terminates there is a k such that all k-round executions of $\mathcal{P}$ decide. Because r is onto, there is a k' such that $r(k') = k$. We show that an arbitrary k'-round execution $E' = (r(k), F, I, M')$ of $\mathcal{P}'$ must decide. By the definition of simulation there is a deciding execution $E = (k, F, I, M)$ of $\mathcal{P}$ such that for all correct processors p and for all i (where $1 \leq i \leq k'$) it is the case that $f_p(state(p, i, E')) = state(p, r(i), E)$. Because r is onto and by the choice of γ', execution E' is also a deciding execution. Therefore protocol $\mathcal{P}'$ terminates.

Condition (2): Suppose not. Then there is a deciding execution $E' = (k, F, I, M')$ of $\mathcal{P}'$ such that $C(ans(E'), F, I)$ is false. By the definition of simulation there is an execution $E = (r(k), F, I, M)$ of $\mathcal{P}$ such that for any correct processor p and for any i (where $1 \leq i \leq k$) it is the case that $f_p(state(p, i, E')) = state(p, r(i), E)$. By choice of γ', $C(ans(E), F, I)$ is also false, contradiction. □

3.2 A Simple Simulation

In the full-information protocol (shown as Protocol 1) each processor at each round broadcasts its entire state, receives one message from each processor, and forms its new state as the ordered collection of all messages received. We now review the well-known result that a full-information protocol can simulate an arbitrary consensus protocol.

Initialization for processor p:

STATE $\leftarrow$ the initial value of processor p

Code for processor p in round r:

1. broadcast STATE
2. receive MSG_q from processor q for $1 \leq q \leq n$
3. STATE $\leftarrow \langle \text{MSG}_1, \ldots, \text{MSG}_n \rangle$

Protocol 1: The Full-Information Protocol

Theorem 2: *Let protocol $\mathcal{P}$ be an arbitrary consensus protocol. The full-information protocol simulates protocol $\mathcal{P}$.*

Proof: Let Q be the state set of the full-information protocol. Suppose protocol $\mathcal{P}$ has input set V, message generating functions $\mu_{p,q}$ for $p, q \in \{1, \ldots, n\}$, and state transition functions δ_p for $p \in \{1, \ldots, n\}$. Let r be the identity function on integers and define $f_p(s)$ for $s \in Q$ and $p \in \{1, \ldots, n\}$ to be

$$f_p(s) = \begin{cases} s & \text{if } s \in V; \\ \delta_p(\mu_{1,p}(f_1(s_1)), \ldots, \mu_{n,p}(f_n(s_n))) & \text{otherwise.} \end{cases}$$

We can verify that the simulation has simulation functions f_p for $p \in \{1, \ldots, n\}$ and scaling function r. □

4. Avalanche Agreement

We formulate and solve the avalanche agreement problem as a building block for use in our compact full-information protocol. At various points in the compact full-information protocol it is convenient to achieve some measure of agreement among the correct processors. We might try using a standard Byzantine agreement protocol for this purpose. Unfortunately, we find that we cannot afford the cost (in rounds) of standard Byzantine agreement. By using an avalanche agreement protocol instead, we are able to achieve a sufficient level of agreement among the correct processors at a cost that we can afford.

A protocol that solves the avalanche agreement problem operates under the same failure and communication assumptions as a Byzantine agreement protocol. Each processor begins the protocol either with an input value from some fixed set V or with no input. We refer to the elements of the set V as *values*, and we indicate the absence of an input by saying that a processor has input $\perp$. Each correct processor may, at some point during the execution of the protocol, irrevocably decide on a value (element of V) as its answer. There are three conditions that the correct processors must satisfy.

- *Avalanche condition:* If any correct processor decides v in round r then all correct processors decide v by round $r + 1$.
- *Consensus condition:* If all correct processors start the protocol with input v then v is the decision of all of the correct processors by round 2.
- *Plausibility condition:* If any correct processor decides v then v must have been the input to some correct processor.

There are five ways in which the avalanche agreement problem differs from the standard Byzantine agreement problem. First, there is no requirement that all executions (of an avalanche agreement protocol) terminate. Second, certain executions (those in which all correct processors have the same input) are required to terminate very fast (in two rounds). Third, in any execution that terminates, all of the correct processors are required to make their decisions within some window of two rounds. Fourth, some correct processors may begin the protocol with no input value. Fifth, no correct processor is permitted to produce as an answer any value that was not the input to at least one correct processor. The first of these differences tends to make the avalanche agreement problem easier to solve than the Byzantine agreement problem. The remainder of the differences tend to make the avalanche agreement problem harder to solve than the Byzantine agreement problem. The combined effect of all of the differences is to make the two problems incomparable.

A variant of the Byzantine agreement problem formulated by Dolev [5] is the crusader agreement problem. At first glance, the avalanche agreement problem may appear similar to the crusader agreement problem, but this similarity is superficial. The two problems are incomparable. Crusader agreement is a harder problem in that all executions of a protocol must be deciding executions. Avalanche agreement is harder in that the answer, if it exists, must be unique. By contrast, up to two distinct answers can be produced by correct processors in an execution of a crusader agreement protocol. Some correct processors agree on some answer; the rest decide that the sender is faulty.

It is straightforward to use standard techniques like those of Fischer, Lynch, and Merritt [11] to show that there is no avalanche agreement protocol that tolerates t processor faults unless the total number of processors, n, is at least $3t + 1$. This bound is tight. Protocol 2 solves the avalanche agreement problem for $n = 3t + 1$. It is a new deterministic protocol designed to solve this new problem; however, it incorporates many ideas from previously known randomized protocols for the standard Byzantine agreement problem. Among these are the protocols of Ben-Or [1], of Chor and Coan [3], and of Rabin [17].

Consider the variant of the avalanche agreement problem in which the consensus condition has been strengthened to require agreement in one round rather than two. It is straightforward to use the proof techniques of Fischer and Lynch [10] to show that if $n \leq 4t$ there is no solution to this variant. If $n \geq 4t + 1$ then it is easy to solve the problem using a simple variant of Protocol 2. We omit the details here.

In the following discussion and proof of Protocol 2 we append two subscripts to each variable from the protocol. The first subscript, say r, is a positive integer and the second subscript, say p, is in $\{1, \ldots, n\}$. By this notation we mean the value of the subscripted variable at processor p at the *end* of round r. For example, $\text{VAL}_{r,p}$ is the value of variable VAL at processor p at the end of round r.

Initialization for processor p:

VAL $\leftarrow$ the initial value of processor p

Code for processor p in round r:

1. broadcast VAL
2. receive MSG_q from processor q for $1 \leq q \leq n$
3. let ANS be the most frequent non-$\perp$ message among the MSG_i (break ties arbitrarily)
4. let NUM be the number of occurrences of ANS
5. if $r = 1$ then
6. if NUM $\geq 2t + 1$ then VAL $\leftarrow$ ANS else VAL $\leftarrow \perp$
7. if $r > 1$ then
8. if NUM $\geq t + 1$ then VAL $\leftarrow$ ANS
9. if NUM $\geq 2t + 1$ and have not decided yet then decide VAL

Protocol 2: The Avalanche Agreement Protocol

In any execution of Protocol 2, value v is *persistent* if there is some correct processor p such that $\text{VAL}_{1,p} = v$. Processor p *votes* for value v in round r if it sends any round r messages containing only v. In every round each correct processor broadcasts a message containing at most one value. A single message that contains more than one value is obviously erroneous and is discarded immediately by its recipient. So, a correct processor only votes for one value in each round, but a faulty processor may vote for many values by sending conflicting votes to different recipients.

We give an informal description of the avalanche agreement protocol before proving it correct. All processors run the same code. For convenience we describe the protocol from the point of view of an arbitrary correct processor p. At the end of round r, the variable $\text{VAL}_{r,p}$ holds the value, if any, that processor p currently prefers as its answer. In round $r + 1$ processor p votes for $\text{VAL}_{r,p}$ and then updates its preference based on the votes it receives. The first round plays a special role in the protocol. In round 1, the number of values favored by correct processors is reduced to at most one — the persistent value. The protocol ensures that after round 1 no correct processor votes for any value other than the persistent value. In the second and subsequent rounds processor p uses the number of votes to predict when there will be an "avalanche" of correct processors favoring some value v (which must be the unique persistent value). As soon as processor p gets enough $(2t + 1)$ votes to predict an avalanche it decides v. Processor p continues to participate in the protocol (send and receive messages) after it has decided.

Lemma 3: *There is at most one persistent value.*

Proof: Assume not. Then, there are values v and v' and correct processors p and q such that $\text{VAL}_{1,p} = v \neq v' = \text{VAL}_{1,q}$. In round 1 processor p must have received at least $2t + 1$ votes for value v and processor q must have received at least $2t + 1$ votes for value v'. The total number of processors is $3t + 1$; therefore, at least $t + 1$ processors including at least one correct processor voted for both v and v'. This is impossible behavior for a correct processor, contradiction. □

Lemma 4: *For all correct processors p and for all rounds $r \geq 1$, either $\text{VAL}_{r,p}$ is the persistent value or $\text{VAL}_{r,p} =$*

Proof: The claim for $r = 1$ follows immediately from Lemma 3, so assume that $r \geq 2$ is the first round in which the claim fails. There is some correct processor p and some non-persistent value v such that $\text{VAL}_{r,p} = v$. In round r processor p must have received at least $t + 1$ votes for v; at least one is from some correct processor q. So, $\text{VAL}_{r-1,q} = v$. This contradicts the assumption that r is the first round in which the claim fails. □

Theorem 5: *Protocol 2 solves the avalanche agreement problem.*

Proof: We show that the avalanche, consensus, and plausibility conditions are satisfied.

Avalanche condition: Say that processor p decides v in round r. By Lemma 4, any correct processor that decides must pick the unique persistent value. Thus, the decision of an arbitrary correct processor q is v. We conclude the proof by showing that all correct processors decide v by round $r + 1$. In round r processor p gets at least $2t + 1$ votes for v; at least $t + 1$ are from correct processors. So, all processors get at least $t + 1$ votes for v in round r. By Lemma 4, any correct processor gets at most t votes for any value $v' \neq v$. Therefore, processor q sets $\text{VAL}_{r,q}$ to v in round r, broadcasts v in round $r + 1$, gets at least $2t + 1$ votes for v in round $r + 1$, and decides v by round $r + 1$.

Consensus condition: Let value v be the input to all of the correct processors. There are at least $2t + 1$ correct processors that all broadcast v in round 1. All correct processors receive at least $2t + 1$ votes for v in round 1 and therefore broadcast v in round 2. All correct processors receive at least $2t + 1$ votes for v in round 2 and therefore decide v in round 2.

Plausibility condition: Let value v be the decision of a correct processor p. By Lemma 4, v is the persistent value. So, at least $2t + 1$ processors (at least $t + 1$ of which are correct) voted for v in round 1. Value v is input to all of these correct processors. □

The communication cost of Protocol 2 is high because processors send messages for an unbounded number of rounds. This cost can be limited in two ways. In many applications (including Section 5) we are only interested in the results of an avalanche agreement protocol for a small fixed number of rounds. We can limit the communication cost by halting the protocol in the first round in which we are uninterested in its results. Alternatively, a simple coding convention for messages allows us to implement Protocol 2 so that at most $O(n^2 \cdot \log |V|)$ message bits are used in any execution. In Protocol 2 each correct processor broadcasts a non-null message each round. The convention gives a meaning to null messages. A processor that wishes to send the same message that it sent in the previous round instead sends the null message (at a cost of 0 bits). It is easy to show that using this convention

each correct processor sends at most 3 non-null messages in any execution.

5. Compact Full-Information Protocols

In Section 1 we outlined the two-step process by which we transform an arbitrary consensus protocol P into a communication-efficient canonical form. The protocol P is first transformed into a full-information protocol that is then transformed into a compact full-information protocol. In this section we complete the description by showing how the compact full-information protocol can simulate the message exchange portion of a full-information protocol using only a polynomial number of message bits.

5.1 Definitions

For any set S a *0-dimensional array of* S is any $s \in S$. An *i-dimensional array of* S is any vector $\langle m_1, \ldots, m_n \rangle$ where, for all j, m_j is an $(i-1)$-dimensional array of S. Our definition of array is standard except that the size along each dimension is always n. An *index array* is an array of $\{1, \ldots, n\}$. A *value array* is an array of V where V is the set of possible inputs to P. In a full-information protocol, all messages sent by correct processors are value arrays and at each round the state of each correct processor is a value array.

A *partial* function may be undefined (denoted $\perp$) on some elements of its domain. We adopt the convention that any partial function used in this paper is undefined whenever any of its arguments is undefined and that any array used in this paper is undefined whenever any of its elements is undefined. Partial function f is an *extension* of partial function g if for all x either $f(x) = g(x)$ or $g(x) = \perp$. A function f defined on arrays is *substitutive* if for all $a_1, \ldots, a_n$ the following holds: $f(\langle a_1, \ldots, a_n \rangle) = \langle f(a_1), \ldots, f(a_n) \rangle$.

When we transform an arbitrary protocol P into a compact full-information protocol the tradeoff between time and communication is determined by parameter k. For any integer $k > 0$, there is a compact full-information protocol $\mathcal{Q}$ that is structured as a series of blocks of $k+2$ rounds. In each of the first k rounds of a block, $\mathcal{Q}$ makes one round of progress in its simulation of P. The last two rounds are overhead — no progress is made.

We define some functions that relate various ways of numbering rounds. Let $r > 0$ be a round in a compact full-information protocol.

- $\text{BLOCK}(r) = \lceil r/(k+2) \rceil$ is the block of which round r is a part.
- $\text{PRIOR}(r) = (\text{BLOCK}(r) - 1) \cdot (k+2)$ is the last round prior to the current block.
- $\text{PHASE}(r) = r - \text{PRIOR}(r)$ is the number of rounds since the start of the current block.
- $\text{SIMUL}(r) = k \cdot (\text{BLOCK}(r) - 1) + \text{MIN}(\text{PHASE}(r), k)$ is the number of rounds of progress that have currently been made in the simulation of the full-information protocol.

In Table 1, we illustrate the relationship among these quantities for 14 actual and 8 simulated rounds of a compact full-information protocol with parameter 2.

r	1	2	3	4	5	6	7	8	9	10	11	12	13	14
BLOCK(r)	1	1	1	1	2	2	2	2	3	3	3	3	4	4
PRIOR(r)	0	0	0	0	4	4	4	4	8	8	8	8	12	12
PHASE(r)	1	2	3	4	1	2	3	4	1	2	3	4	1	2
SIMUL(r)	1	2	2	2	3	4	4	4	5	6	6	6	7	8

Table 1: An Execution of 14 Rounds with $k = 2$

5.2 Subprotocols

Ordinary sequential programming problems are frequently decomposed into simpler subproblems using subroutines. In a similar way we simplify our compact full-information protocol by using avalanche agreement as a subprotocol. Subprotocols are similar to subroutines in that they help us decompose problems; however, they have different semantics. For example, our subprotocols run in parallel with the main protocol and their results take at least one round to become available. In this subsection we define the syntax and semantics that we will use for calls to subprotocols.

Recall that each round of any protocol P consists of three components that are performed in order: sending messages, receiving messages, and local state change. In the language we use to write our protocols there is no specific mechanism that ensures that this structure is followed; however, a quick inspection is generally sufficient to verify that it is. We will only write protocols that conform to this round structure.

We adopt the convention that if a call to subprotocol SUB appears in round r of protocol P then the first round of SUB coincides with round r of P. This implies that the call to SUB should appear in the text of P before any round r sends and that all inputs to SUB must be available at the start of round r (*i.e.*, computed at round $r-1$ at the latest). If processor p decides in SUB in the round that coincides with round r' of P then we make this answer available to processor p in round r' of P before it computes its local state change. This simply means that in the local-state-change portion of each round of protocol P we perform all of the local state changes of the subprotocols before we perform the local state change of P. Relative order among the subprotocols does not matter because they do not interact. We require that all processors in protocol P initiate precisely the same subprotocols at precisely the same rounds. If in round r of protocol P there are x active subprotocols running then all round r messages are $(x+1)$-tuples — one component for each subprotocol and one component for P.

Within a protocol P, a call to subprotocol SUB is written

call SUB(input: IN, result: OUT, rounds: r)

where IN and OUT are variables in $\mathcal{P}$ and r is an integer. (We require that all processors in $\mathcal{P}$ use the same value for r.) Let r' be the round in which protocol $\mathcal{P}$ executes the call statement. The variable IN must be defined by the end of round $r'-1$. Subprotocol SUB is started with input IN in round r' and run for r rounds. The variable OUT initially has the value $\perp$. If processor p in subprotocol SUB decides v in round r of $\mathcal{P}$ then the instance of variable OUT at processor p is set to v at the start of the local-state-change portion of round r of $\mathcal{P}$. There is no requirement that processors in SUB eventually decide.

5.3 The Protocol

The code for the compact full-information protocol is given as Protocol 3. In the following discussion and proof we append two subscripts to each variable from Protocol 3. The first subscript, say r, is a positive integer and the second subscript, say p, is in $\{1,\ldots,n\}$. By this notation we mean the value of the subscripted variable at processor p at the *end* of round r. For example, $\text{OUT}_{q,b,r,p}$ is the value of variable $\text{OUT}_{q,b}$ at processor p at the end of round r.

At each round each correct processor computes several *expansion functions* based on its current state — in particular, based on the results of the various avalanche agreement subprotocols that it has run. The round r expansion functions of processor p are denoted $\phi_{b,r,p}$, for $b \in \{1,\ldots,\text{BLOCK}(r)\}$. If $b > 1$ then $\phi_{b,r,p}$ is a substitutive partial function from index arrays to value arrays. If $b = 1$ then $\phi_{b,r,p}$ is the identity function on value arrays (which is also substitutive). Because expansion functions are substitutive, it is sufficient to define them on scalars. We let $\phi_{b,r,p}(x) = \perp$ if either $b = 1$ and $x \notin V$ or $b > 1$ and $x \notin \{1,\ldots,n\}$. In all other cases $\phi_{b,r,p}$ is defined as follows.

$$\phi_{b,r,p}(x) = \begin{cases} x & \text{if } b = 1; \\ \phi_{b-1,r,p}(\text{OUT}_{x,b,r,p}) & \text{otherwise.} \end{cases}$$

(All uses of $\phi_{b,r,p}$ in Protocol 3 refer to the above definition.)

We explain the sense in which $\text{CORE}_{r,p}$ is a compressed form of the state of processor p in the simulated full-information protocol. We define $\text{FULL\_STATE}_{r,p}$ in terms of information available in the round r state of processor p as follows: $\text{FULL\_STATE}_{r,p} = \phi_{\text{BLOCK}(r),r,p}(\text{CORE}_{r,p})$. What we will show in Section 5.5 is that for any round r, if $\text{PHASE}(r) \leq k$ then the round $\text{SIMUL}(r)$ state of correct processor p in the simulated full-information protocol is $\text{FULL\_STATE}_{r,p}$. In this sense $\text{CORE}_{r,p}$ is the compressed state of processor p at round r in Protocol 3. Because of the substitutivity property of the expansion function and other properties which we will state and prove in Section 5.4 it is possible for our protocol to work directly with compressed processor states.

5.4 Technical Lemmas

Lemma 6: *For all correct processors p and for all processors q, if $r' = \text{PRIOR}(r) - 1$, $b = \text{BLOCK}(r)$, $b > 1$, and $\text{OUT}_{q,b,r,p} \neq \perp$ then there is some correct processor u such that $\phi_{b-1,r',u}(\text{OUT}_{q,b,r,p}) \neq \perp$.*

Initialization for processor p:

CORE $\leftarrow$ the initial value of processor p

Code for processor p in round r where $b = \text{BLOCK}(r)$:

```
1.    if PHASE(r) ≤ k then
2.        broadcast CORE
3.        receive MSG_i from processor i for 1 ≤ i ≤ n
4.        for i ← 1 to n do
5.            if φ_{b,r,p}(MSG_i) = ⊥
                  then VAL_i ← CORE
                  else VAL_i ← MSG_i
6.        CORE ← ⟨VAL_1, ..., VAL_n⟩
7.    if PHASE(r) = k + 1 then
8.        broadcast CORE
9.        receive MSG_q from processor q for 1 ≤ q ≤ n
10.       for i ← 1 to n do
11.           if φ_{b,r,p}(MSG_i) = ⊥
                  then IN_{i,b+1} ← ⊥
                  else IN_{i,b+1} ← MSG_i
12.   if PHASE(r) = k + 2 then
13.       for i ← 1 to n do
14.           call AVALANCHE(input: IN_{i,b+1},
                  result: OUT_{i,b+1}, rounds: k + 3)
15.       CORE ← p
```

Protocol 3: The Compact Full-Information Protocol

Proof: $\text{OUT}_{q,b,r,p}$ is the answer produced by an invocation of avalanche agreement. By the plausibility condition of avalanche agreement, there is some correct processor u that started this invocation of avalanche agreement with input $\text{OUT}_{q,b,r,p}$. Processor u must have observed that $\phi_{b-1,r',u}(\text{OUT}_{q,b,r,p}) \neq \perp$ at step 11 of the code; otherwise, it would have used $\perp$ (no legal input) as its input to avalanche agreement. □

Lemma 7: *For all $b > 0$ and for all correct processors p and q, if $\text{BLOCK}(r) = b$ and $\text{PHASE}(r) \neq k+2$ then $\phi_{b,r+1,p}$ is an extension of $\phi_{b,r,q}$.*

Proof: The proof is by induction on b.

Basis: ($b = 1$) This is trivial because $\phi_{b,r+1,p}$ and $\phi_{b,r,q}$ are both the identity function.

Induction: ($b > 1$) Consider an arbitrary message m. If $\phi_{b,r,q}(m) = \perp$ then the claim is trivially true. So, assume that $\phi_{b,r,q}(m) \neq \perp$. We wish to show that $\phi_{b,r+1,p}(m) = \phi_{b,r,q}(m)$. Let I be the set of indices in m. Let $r' = \text{PRIOR}(r)$. By the substitutivity property of the expansion functions, $\phi_{b,r+1,p}$ and $\phi_{b,r,q}$, it is sufficient to show that for all i in I that $\phi_{b-1,r',p}(\text{OUT}_{i,b,r+1,p}) = \phi_{b-1,r',q}(\text{OUT}_{i,b,r,q})$.

Because $\phi_{b,r,q}(m) \neq \perp$ we know that for all i in I, $\text{OUT}_{i,b,r,q} \neq \perp$ and therefore by the avalanche condition of avalanche agreement $\text{OUT}_{i,b,r+1,p} = \text{OUT}_{i,b,r,q}$. By Lemma 6, there is a correct processor s such that $\phi_{b-1,r'-1,s}(\text{OUT}_{i,b,r,q}) \neq \perp$. By the induction hypothesis, $\phi_{b-1,r',p}$ and $\phi_{b-1,r',q}$ are extensions of $\phi_{b-1,r'-1,s}$ and so $\phi_{b-1,r',p}(\text{OUT}_{i,b,r+1,p}) = \phi_{b-1,r',p}(\text{OUT}_{i,b,r,q}) = \phi_{b-1,r'-1,s}(\text{OUT}_{i,b,r,q}) = \phi_{b-1,r',q}(\text{OUT}_{i,b,r,q})$. □

Lemma 8: *If* $\text{PHASE}(r) = k$ *and* $\text{BLOCK}(r) = b$ *then for all correct processors* p *and* q, $\text{OUT}_{p,b+1,r+3,q} = \text{CORE}_{r,p}$.

Proof: In round $r+2$, avalanche agreement on the round $r+1$ message broadcast by processor p will be initiated. The round $r+3$ result of this agreement at processor q is $\text{OUT}_{p,b+1,r+3,q}$. Each correct processor will use either $\text{CORE}_{r,p}$ (which is the round $r+1$ message broadcast by correct processor p) or $\perp$ as its input to this avalanche agreement protocol. Step 5 of the code ensures that, $\phi_{b,r,p}(\text{CORE}_{r,p}) \neq \perp$. So, by Lemma 7, for all correct processors s, $\phi_{b,r+1,s}(\text{CORE}_{r,p}) \neq \perp$. Therefore all correct processors will use $\text{CORE}_{r,p}$ as input to the avalanche agreement protocol started at round $r+2$. By the consensus condition of avalanche agreement $\text{OUT}_{p,b+1,r+3,q} = \text{CORE}_{r,p}$ which is what we sought to show. □

5.5 Proof of Simulation

For all $p \in \{1,\ldots,n\}$ and for any processor state s define the function $f_p(s)$ to be $\phi_{\text{DL}\cdots\text{K}(r),r,p}(\text{CORE}_{r,p})$ where r and CORE are implicit in s.

Theorem 9: *The compact full-information protocol simulates the full-information protocol with simulation functions* f_i *for* $i \in \{1,\ldots,n\}$ *and scaling function* SIMUL.

Proof: We must show that for any r and for any r-round execution E of the compact full-information protocol, there is a $\text{SIMUL}(r)$-round execution E' of the full-information protocol such that for any correct processor p and for any i (where $1 \leq i \leq r$) it is the case that $f_p(state(p,i,E)) = state(p, \text{SIMUL}(i), E')$.

The proof is by induction on r. Let $b = \text{BLOCK}(r)$, and let $r' = \text{SIMUL}(r)$.

Basis: ($r = 0$.) The execution E' is simply constructed. The set of correct processors in E' is the same as in E. The correct processors have the same input in the two executions.

Induction: If $\text{PHASE}(r) > k$ then the theorem follows from Lemma 7 and the induction hypothesis. Assume instead that $\text{PHASE}(r) \leq k$.

If $\text{PHASE}(r) = 1$ and $r > 1$ then let the execution F consist of the first $r-3$ rounds of E; otherwise, let the execution F consist of the first $r-1$ rounds of E. By the induction hypothesis there is an $(r'-1)$-round execution F' of the full-information protocol such that F is a simulation of F'. We show that the $(r'-1)$-round execution F' can be extended by one round to get the r'-round execution E' whose existence is claimed.

The extension to F' is fully described by specifying the round r' messages sent from faulty processors to correct processors. It is unnecessary to specify the messages sent by correct processors because there is no choice and it is unnecessary to specify the messages sent from faulty processors to faulty processors because these messages do not matter.

Let s be an arbitrary faulty processor and let p be an arbitrary correct processor. We specify that the round r' message from s to p in E' is $\phi_{b,r,p}(\text{VAL}_{s,r,p})$. By the induction hypothesis $\phi_{b,r,p}(\text{VAL}_{s,r,p}) \neq \perp$.

Let $\langle v_1,\ldots,v_n\rangle = \phi_{b,r,p}(\text{CORE}_{r,p})$; and let $\langle v'_1,\ldots,v'_n\rangle$ be the final state of processor p in E'. We now verify that $\langle v_1,\ldots,v_n\rangle = \langle v'_1,\ldots,v'_n\rangle$. Consider an arbitrary processor q. It is sufficient to show that $v_q = v'_q$. There are three cases.

Case 1: (Processor q is correct and $\text{PHASE}(r) = 1$ and $r > 1$.) Since q is correct, $v'_q = \phi_{b-1,r-3,q}(\text{CORE}_{r-3,q}) \neq \perp$. By Lemma 8, $\text{OUT}_{q,b,r,p} = \text{CORE}_{r-3,q}$. Therefore processor p places q in the q-th position of $\text{CORE}_{r,p}$ and the q-th component of $\phi_{b,r,p}(\text{CORE}_{r,p})$ is $\phi_{b-1,r-1,p}(\text{OUT}_{q,b,r,p})$. $\phi_{b-1,r-1,p}(\text{OUT}_{q,b,r,p}) = \phi_{b-1,r-3,q}(\text{CORE}_{r-3,q})$ by Lemma 7. Therefore, $v_q = \phi_{b-1,r-3,q}(\text{CORE}_{r-3,q})$. This shows that $v_q = v'_q$.

Case 2: (Processor q is correct and either $\text{PHASE}(r) > 1$ or $r = 1$.) Because q is correct $\text{MSG}_{q,r,p} = \text{CORE}_{r-1,q}$ and $v'_q = \phi_{b,r-1,q}(\text{MSG}_{q,r,p}) \neq \perp$. We can see that by Lemma 7, $\phi_{b,r,p}(\text{MSG}_{q,r,p}) = \phi_{b,r-1,q}(\text{MSG}_{q,r,p})$. Therefore (in steps 5–6) processor p incorporates $\text{MSG}_{q,r,p}$ as the q-th component of $\text{CORE}_{r,p}$ and so $v_q = \phi_{b,r,p}(\text{MSG}_{q,r,p})$. This shows that $v_q = v'_q$.

Case 3: (Processor q is faulty.) By the specification of E', $v'_q = \phi_{b,r,p}(\text{VAL}_{s,r,p})$. Processor p (in steps 5–6) incorporates $\text{VAL}_{s,r,p}$ as the q-th component of $\text{CORE}_{r,p}$. So, $v_q = \phi_{b,r,p}(\text{VAL}_{s,r,p})$. This shows that $v_q = v'_q$. □

5.6 Performance Analysis

Corollary 10: *For any* $\epsilon > 0$, *the Byzantine agreement problem can be solved in* $(1+\epsilon)(t+1)$ *rounds using* $O(t \cdot n^{\lceil 2/\epsilon \rceil + 3} \cdot \log |V|)$ *message bits.*

Proof: There are known $(t+1)$-round exponential-message Byzantine agreement protocols, for example the protocol of Lamport *et al.* [13]. This means that there is a decision rule to apply to the final state if we use the compact full-information protocol to simulate $t+1$ rounds of message exchange in a full-information protocol. The simulation together with the decision rule constitutes a Byzantine agreement protocol.

In each of the first k rounds of a block of the compact full-information protocol, one round of progress is made in the simulation of the full-information protocol. In the last two rounds, no progress is made. Therefore, for all x, in $\frac{k+2}{k}x$ actual rounds, the compact full-information protocol has simulated at least x rounds of the full-information protocol. In order for our Byzantine agreement protocol to terminate within $(1+\epsilon)(t+1)$ actual rounds, we require that $(k+2)/k < 1+\epsilon$. Solving for the minimum integer k we get $k = \lceil 2/\epsilon \rceil$. Therefore, to achieve Byzantine agreement in $(1+\epsilon)(t+1)$ rounds, we run the compact full-information protocol with parameter $k = \lceil 2/\epsilon \rceil$ and then (after $t+1$ simulated rounds) apply the decision rule derived from Lamport's protocol.

For the compact full-information protocol, the communication cost consists of the cost of avalanche agreement and the cost of the remainder of the protocol. In the non-

avalanche portion of the protocol, in each of $O(t)$ rounds each processor broadcasts a message of size $O(n^k \cdot \log |V|)$ for a total cost of $O(t \cdot n^{k+2} \cdot \log |V|)$ bits. This cost is dominated by the cost of avalanche agreement. In the avalanche agreement portion of the protocol, in each of $O(t)$ rounds, each processor broadcasts at most n messages of size $O(n^k \cdot \log |V|)$ for a total of $O(t \cdot n^{k+3} \cdot \log |V|)$. Expressed in terms of ϵ, this communication complexity is $O(t \cdot n^{\lceil 2/\epsilon \rceil + 3} \cdot \log |V|)$ message bits. □

If $n \geq 4t + 1$ then a modification of our technique can transform any $(t + 1)$-round consensus protocol to a $(1+\epsilon)(t+1)$-round protocol that uses $O(t \cdot n^{\lceil 1/\epsilon \rceil + 3} \cdot \log |V|)$ message bits. Given that $n \geq 4t + 1$ it is possible to solve a variant of the avalanche agreement problem with a consensus condition modified to require a decision in one round rather than two. Using this variant avalanche agreement protocol, we can reduce the number of rounds in each block of a compact full-information protocol by one. Analyzing the new compact full-information protocol gives the total communication cost of $O(t \cdot n^{\lceil 1/\epsilon \rceil + 3} \cdot \log |V|)$ message bits.

We compare the cost (*i.e.*, rounds and message bits) of our Byzantine agreement protocol (for $n = 4t + 1$) with the cost of the protocol of Srikanth and Toueg [18] (which uses the smallest number of rounds of any previously known protocol and which only requires that $n \geq 3t + 1$). The protocol of Srikanth and Toueg uses $2t + 1$ rounds and $O(t \cdot n^2 \cdot \log n \cdot \log |V|)$ message bits. If $\epsilon = 1$ our protocol uses $2t + 2$ rounds and $O(t \cdot n^4 \cdot \log |V|)$ message bits. If $\epsilon = \frac{1}{2}$ our protocol uses $1\frac{1}{2}t + 1\frac{1}{2}$ rounds and $O(t \cdot n^5 \cdot \log |V|)$ message bits. If $\epsilon = \frac{1}{3}$ our protocol uses $1\frac{1}{3}t + 1\frac{1}{3}$ rounds and $O(t \cdot n^6 \cdot \log |V|)$ message bits. We find that our protocol uses somewhat more message bits, but it allows us to greatly reduce the number of rounds. Also, our technique is more general and may therefore have greater applicability (*e.g.*, reducing the communications cost of the approximate agreement protocol of Fekete [9]). A significant limitation of our technique is the large amount of local computation that it requires. By contrast the protocol of Srikanth and Toueg uses a small amount of space and time locally at each processor. In this comparison we ignore a possible optimization due to Dolev *et al.* [6] and another due to Perry [16] and to Turpin and Coan [19] because these optimizations have a similar (and small) impact on both protocols.

Acknowledgment

I would like to thank Mike Fischer and Nancy Lynch for suggesting many ways to improve the presentation of this paper and for suggesting the general method of attack that led to the development of this result.

References

[1] M. Ben-Or, "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols," *Proceedings of the 2nd Symposium on Principles of Distributed Computing*, pp. 27–30, 1983.

[2] J. Burns and N. Lynch, "The Byzantine Firing Squad Problem," *Advances in Computing Research: Parallel and Distributed Computing*, vol. 4, JAI Press Inc., Greenwich, Connecticut, to appear. (Also available as MIT Technical Report MIT/LCS/TM-275, 1985.)

[3] B. Chor and B. Coan, "A Simple and Efficient Randomized Byzantine Agreement Algorithm," *Transactions on Software Engineering*, vol. SE–11, pp. 531–539, 1985.

[4] B. Coan, D. Dolev, C. Dwork, and L. Stockmeyer, "The Distributed Firing Squad Problem," *Proceedings of the 17th Symposium on Theory of Computing*, pp. 335–345, 1985.

[5] D. Dolev, "The Byzantine Generals Strike Again," *Journal of Algorithms*, vol. 3, pp. 14–30, 1982.

[6] D. Dolev, M. Fischer, R. Fowler, N. Lynch, and H. Strong, "An Efficient Algorithm for Byzantine Agreement without Authentication," *Information and Control*, vol. 52, pp. 257–274, 1982.

[7] D. Dolev, N. Lynch, S. Pinter, E. Stark, and W. Weihl, "Reaching Approximate Agreement in the Presence of Faults," *Journal of the ACM*, to appear. (Also available in *Proceedings of the 3rd Symposium on Reliability in Distributed Software and Database Systems*, pp. 145–154, 1983.)

[8] D. Dolev and R. Strong, "Polynomial Algorithms for Multiple Processor Agreement," *Proceedings of the 14th Symposium on Theory of Computing*, pp. 401–407, 1982.

[9] A. Fekete, "Asymptotically Optimal Algorithms for Approximate Agreement," *Proceedings of the 5th Symposium on Principles of Distributed Computing*, 1986.

[10] M. Fischer and N. Lynch, "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, vol. 14, pp. 183–186, 1982.

[11] M. Fischer, N. Lynch, and M. Merritt, "Easy Impossibility Proofs for Distributed Consensus Problems," *Proceedings of the 4th Symposium on Principles of Distributed Computing*, pp. 59–70, 1985.

[12] L. Lamport, "The Weak Byzantine Generals Problem," *Journal of the ACM*, vol. 30, pp. 668-676, 1983.

[13] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, pp. 382–401, 1982.

[14] N. Lynch, M. Fischer, and R. Fowler, "A Simple and Efficient Byzantine Generals Algorithm," *Proceedings of the 2nd Symposium on Reliability in Distributed Software and Database Systems*, pp. 46–52, 1982.

[15] M. Pease, R. Shostak, and L. Lamport, "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, vol. 27, pp. 228–234, 1980.

[16] K. Perry, "Early Stopping Protocols for Fault-Tolerant Distributed Agreement," Ph.D. Thesis, Cornell University, 1985.

[17] M. Rabin, "Randomized Byzantine Generals," *Proceedings of the 24th Symposium on Foundations of Computer Science*, pp. 403–409, 1983.

[18] T. Srikanth and S. Toueg, "Byzantine Agreement Made Simple: Simulating Authentication without Signatures," Cornell Technical Report 84–623, 1984.

[19] R. Turpin and B. Coan, "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement," *Information Processing Letters*, vol. 18, pp. 73–76, 1984.