A Tight Lower Bound for Processor Coordination*

Soma Chaudhuri† Maurice Herlihy‡ Nancy A. Lynch† Mark R. Tuttle‡

Abstract

We prove a tight lower bound on the running time of oblivious solutions to k-set agreement. In *k-set agreement*, processors start with input values from a given set and choose output values from the same set. In every execution, the set of output values must be contained in the set of input values, and the set of output values must have size at most k. A solution is *oblivious* if it does not make use of processor identities. We analyze this problem in a synchronous model where processors can fail by just stopping. We prove a lower bound of $\lfloor f/k \rfloor + 1$ rounds of communication for oblivious solutions that tolerate f failures. This shows that there is an inherent trade-off between the running time, the degree of coordination required, and the number of faults tolerated, even in idealized models like ours.

1 Introduction

Many of the problems that arise when building a *responsive system* are related to problems in theoretical distributed computing. These problems include coordinating the activities of concurrent processors—both the actions they perform and the resources they use—and resolving the conflicts that arise, recovering from the failure of processors and communication links, and coping with uncertainty about the amount of time taken by events like message delivery and processor steps. Given the long history of theoretical work on these problems in distributed computing, it is reasonable to hope that some of the tools and techniques developed there might be useful when building and analyzing responsive systems. For example, over the years, the theory has developed some sophisticated techniques for proving lower bounds on the amount of time or resources needed to solve a problem in a distributed system, and we believe these techniques may be useful when proving lower bounds in responsive systems. In this paper, we illustrate an elegant combination of these techniques [FL82, DM90, MT88, Cha91, HS93] by proving a tight lower bound on the time needed to solve a processor coordination problem called *k-set agreement* [Cha91].

The *k-set agreement* problem is defined as follows. Each processor in the system starts with an arbitrary input value from a set V, and halts after choosing an output value from V. These output values must satisfy two conditions: each output value must be some processor's input value,

*This paper appeared in *Proceedings of the Third International Workshop on Responsive Computer Systems*, pages 4–15, September 1993.

†MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. Authors' email addresses are soma@theory.lcs.mit.edu and lynch@theory.lcs.mit.edu. Supported in part by NSF grant CCR-89-15206, DARPA contracts N00014-89-J-1988, N00014-92-J-4033, and N00014-92-J-1799, and ONR contract N00014-91-J-1046.

‡Digital Equipment Corporation, Cambridge Research Lab, One Kendall Square, Bldg. 700, Cambridge, MA 02139. Authors' email addresses are herlihy@crl.dec.com and tuttle@crl.dec.com.

and at most k distinct output values are chosen. The first condition rules out trivial solutions in which a hardwired value $v \in V$ is chosen by all processors in all executions, and the second condition requires that the processors coordinate their choices in some way. When $k = 1$, the second condition requires that all processors choose the same output value, so 1-set agreement is equivalent to the well-known *consensus* problem [LSP82, PSL80, FL82, FLP85, Dol82, Fis83]. The consensus problem demands a high degree of processor coordination, and arises in applications as diverse as on-board aircraft control [W+78], database transaction commit [BHG87], and concurrent object design [Her88]. Varying the value of k allows us to vary the degree of coordination required.

Before we can prove any lower bound for k-set agreement in a responsive system, we have to choose a system model. The purpose of a model is to define the set of behaviors that the processors in the system can exhibit. The strategy for proving a lower bound is to show that, in one of these behaviors, the processors run for a long time before solving the problem. Choosing the right system model is difficult, since it is not clear what properties determine whether a system is responsive or not. Fortunately, there is a popular model in distributed computing that is likely to be a special case of whatever more general model emerges as the definition of a responsive system. The *synchronous model* consists of n processors that communicate by sending messages over a completely connected network. The model makes some strong (and possibly unrealistic) assumptions, such as that all processors take steps at the same rate, and that all messages take the same amount of time to be delivered. Communication is considered to be reliable, but up to f processors can fail by stopping in the middle of their protocol.

We prove our lower bound in this synchronous model, and we show that any oblivious protocol for k-set agreement in this model requires $\lfloor f/k \rfloor + 1$ rounds of communication in the worst case, assuming $n \geq f + k + 1$ (that is, there are at least $k + 1$ nonfaulty processors). Loosely speaking, an *oblivious* protocol is one that is oblivious to processor identities, in the sense that two processors receiving the same set of messages will choose the same output value, regardless of their processor ids. This lower bound is tight, since Chaudhuri has already demonstrated a protocol solving k-set agreement in $\lfloor f/k \rfloor + 1$ rounds [Cha91].[1] In addition, since consensus is 1-set agreement, this lower bound implies the well-known lower bound of $f + 1$ rounds for consensus when $n \geq f + 2$. Our lower bound is intriguing because it shows that there is a smooth and inescapable tradeoff between the number f of faults tolerated, the degree k of coordination demanded, and the execution time required.

Our synchronous model is a special case of almost every realistic model of a responsive system we can imagine. Proving lower bounds in this model is a good idea, because any lower bound holding in this model also holds in more general models. For example, consider the slightly more realistic *partially synchronous model*. In this model, the rate at which processors take steps varies between two constants c_1 and c_2, and message delivery times vary between 0 and d. Every behavior possible in the synchronous model corresponds to an orderly, well-behaved execution in the partially synchronous model in which all processors take steps every c_1 time units and all messages are delivered in d time units, so the existence of a long execution in the first model implies the existence of a long execution in the second. In particular, our lower bound of $\lfloor f/k \rfloor + 1$ rounds in the synchronous model translates into a lower bound of $(\lfloor f/k \rfloor + 1)d$ time units in the partially synchronous model.

[1] In the same paper, she also proves the matching lower bound of $\lfloor f/k \rfloor + 1$ rounds for k-set agreement, but for a much more restricted class of protocols. In particular, a protocol's decision function can depend only on vectors giving partial information about which processors started with which initial values, but can not depend on processor identities or message histories.

The problem with this kind of translation is that the translated lower bound may not be as tight as possible. For example, the well-known $f+1$ round lower bound for consensus in the synchronous model translates into a lower bound of $(f+1)d$ time units in the partially synchronous model. On the other hand, Attiya et al. [ADLS93] have proven a lower bound of $(f-1)d + Cd$, where $C = c_2/c_1$, and this is better than the translated lower bound when $C > 2$. We think that proving lower bounds for k-set agreement in this partially synchronous model is important. Either the techniques in [ADLS93] can be used to translate our lower bound for k-set agreement into this model, or new techniques will be required. In either case, we will better understand how to reason about responsive systems. Good lower bounds in this model remain for future work.

2 Overview

In this section, we give an informal overview of our lower bound proof for k-set agreement. Suppose P is a protocol solving k-set agreement in r rounds, and tolerating the failure of f out of n processors. Our goal is to consider the *global states* that occur at time r in executions of P, and to show that in one of these states there are $k+1$ processors that have chosen $k+1$ distinct values, violating k-set agreement. Our strategy is to consider the *local states* of processors that occur at time r in executions of P, and to investigate the combinations of these local states that occur in global states. This investigation depends on constructing a geometric object, and in this section we use a simplified version of this object to illustrate the general ideas in the proof. These ideas include ideas due to Chaudhuri [Cha91, Cha93], Fischer and Lynch [FL82], Herlihy and Shavit [HS93], and Dwork, Moses, and Tuttle [DM90, MT88].

We begin by constructing a k-dimensional *simplex* in k-dimensional Euclidean space [Cha93, HS93]. A simplex is just the natural generalization of a triangle to k dimensions: for example, a 1-dimensional simplex is an edge, a 2-dimensional simplex is a triangle, and a 3-dimensional simplex is a tetrahedron. We jokingly refer to this simplex as the *Bermuda Triangle* B, since all fast protocols vanish somewhere in its interior. The simplex contains a number of *grid points*, which are the points in Euclidean space with integer coordinates. We *triangulate* this simplex with respect to these grid points via a collection of smaller k-dimensional simplexes. We then label each grid point with a local state in such a way that for each simplex T in the triangulation there is a global state g consistent with the local states labeling the simplex: for each local state s labeling a corner of T, there is a nonfaulty processor p with local state s in g.

A simplified Bermuda Triangle B is illustrated in Figure 1, assuming P is a protocol for 5 processors solving 2-set agreement in 1 round. Given 3 distinct input values a, b, c, we write $bb?aa$ to denote the local state of a processor p at the end of a round in which the first two processors have input value b and send messages to p, the middle processor fails to send a message to p, and the last two processors have input value a and send messages to p. We label the points of B with local states as shown in Figure 1. Following any horizontal line from left to right, the input values are changed from a to b. The value of each processor is changed (one after another) by first silencing the processor, and then reviving the processor with the input value b. Similarly, moving along any vertical line from bottom to top, processors' input values change from b to c.

This labeling of local states has the following property. In the local state on a corner of B, each processor starts with the same input value, so any processor with this local state at the end of P must choose this value. In a local state on an edge of B, each processor starts with one of the two input values labeling the ends of the edge, so any processor with this local state at the end of P

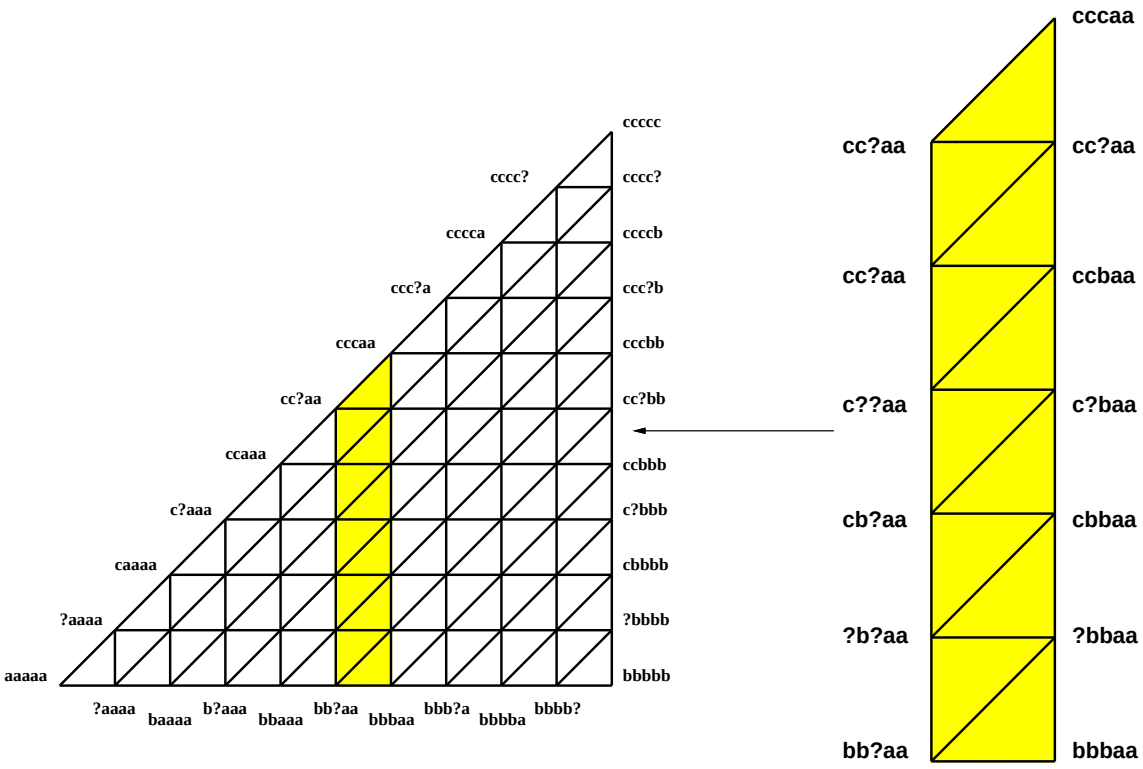

Figure 1: The Bermuda Triangle for 5 processors and a 1-round protocol for 2-set agreement.

must choose one of these two values. Similarly, in a local state in the interior of B, any processor with this local state at the end of P must choose one of the three values labeling the corners of B.

Now let us "color" each grid point with the output value that P has a processor choose when its local state is the state labeling the grid point. This coloring of B has the property that the color of each of the corners is determined uniquely, the color of each point on an edge between two corners is forced to be the color of one of the corners, and the color of each interior point can be the color of any corner. Colorings with this property are called *Sperner colorings*, and have been studied extensively in the field of algebraic topology. At this point, we exploit a remarkable combinatorial result first proved in 1928: *Sperner's Lemma* [Spa66, p.151] states that any Sperner coloring of any triangulated k-dimensional simplex must include at least one simplex whose corners are colored with all $k+1$ colors. In our case, however, this simplex corresponds to a global state in which $k+1$ processors choose $k+1$ distinct values, which contradicts the definition of k-set agreement. Thus, in the case illustrated above, there is no protocol for 2-set agreement halting in 1 round.

The technical challenge in this paper is labeling the grid points of B with local states when the protocol P runs for more than a single round. Our approach consists of three steps. First, we label points on the edges of B with global states. For example, consider the edge between the corner where all processors start with input value a and the corner where all processors start with b. We construct a long sequence of global states that begins with a global state in which all processors start with a, ends with a global state in which all processors start with b, and in between systematically changes input values from a to b. These changes are made so gradually, however, that for any two adjacent global states in the sequence, at most one processor can distinguish them. Second, we then label each remaining point by combining global states on the edges. Finally, we project

each global state onto the local state of an arbitrarily chosen nonfaulty processor, completing the labeling of B.

In the remainder of the paper, we define k-set consensus and our model more precisely, describe the construction above in more detail, and discuss generalizing our lower bound to other models.

3 The Problem

In this section, we define the k-set agreement problem, define our model of computation, and define a compact representation of global and local states.

3.1 k-Set Agreement

The *k-set agreement problem* [Cha91] is defined as follows. We assume that each processor p_i has two private registers in its local state, a read-only input register and a write-only output register. Initially, p_i's input register contains an arbitrary input value v_i from a set V containing at least $k+1$ values, and its output register is empty. A protocol solves the problem if it causes each processor to halt after writing an output value to its output register in such a way that (1) every processor's output value is some processor's input value, and (2) the set of output values chosen has size at most k.

3.2 Model

We use a synchronous, message-passing model with processor stopping failures. The system consists of n processors, $p_1, \ldots, p_n$. Processors share a global clock that starts at 0 and advances in increments of 1. Computation proceeds in a sequence of *rounds*, with round r lasting from time $r-1$ to time r. Computation in a round consists of three phases: first each processor p sends messages to some of the processors in the system, possibly including itself, then it receives the messages sent to it during the round, and finally it performs some local computation and changes state. We assume that the communication network is totally connected: every processor is able to send distinct messages to every other processor in every round. We also assume that communication is reliable (although processors can fail): if p sends a message to q in round r, then the message is delivered to q in round r.

Processors follow a deterministic *protocol* that determines what messages a processor should send and what output a processor should generate. A protocol has two components: a *message component* that maps a processor's local state to the list of messages it should send in the next round, and an *output component* that maps a processor's local state to the output value (if any) that it should choose. Processors can be faulty, however, and any processor p can simply *stop* at any time r. In this case, processor p follows its protocol and sends all messages the protocol requires in rounds 1 through $r-1$, sends some subset of the messages it is required to send in round r, and sends no messages in rounds after r. We say that p is *silent* from round r if p sends no messages in round r or later.

A *full-information protocol* is one in which every processor broadcasts its entire local state to every processor, including itself, in every round. For simplicity, and without loss of generality, we restrict attention to full-information protocols. Thus, in an r round full-information protocol,

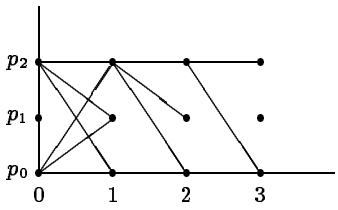

Figure 2: A three-round communication graph.

processors exchange their local states for r rounds and then simultaneously apply their output functions to their local states to choose an output value.

We need one more technical restriction. An r-round full-information protocol is said to be *oblivious* if the output component applied to processor states occurring after r rounds is a function of just the list of messages a processor p receives in the rth round, independent of p's processor id. We assume that our protocols are oblivious, but more recent results have removed this restriction [CHLT93].

3.3 Communication Graphs

We end this section with a compact way to represent an execution of a full-information protocol P called a *communication graph* [MT88]. The communication graph $\mathcal{G}$ for an r-round execution of P is a two-colored graph. The vertices form an $n \times r$ grid, with processor names 1 through n labeling the vertical axis and times 0 through r labeling the horizontal axis. The node representing processor p at time i is labeled with the pair $\langle p, i \rangle$. Given any pair of processors p and q and any round i, there is an edge between $\langle p, i-1 \rangle$ and $\langle q, i \rangle$ whose color determines whether p successfully sends a message to q in round i: the edge is green if p succeeds, and red otherwise. In addition, each node $\langle p, 0 \rangle$ is labeled with p's input value. Figure 2 illustrates a three round communication graph; in this figure, only green edges are indicated.

In the stopping failure model, a processor is silent in all rounds following the round in which it stops. This means that all communication graphs representing executions in this model have the *consistency property* that if there is a red edge from $\langle p, i-1 \rangle$ to $\langle q, i \rangle$, then all edges leaving nodes of the form $\langle p, j \rangle$, $j \geq i+1$, are also red. We assume that all communication graphs in this paper have this property, and we note that every r-round graph with this property corresponds to an r-round execution of P.

Since a communication graph $\mathcal{G}$ describes an execution of P, it also determines the global state at the end of P, so we sometimes refer to $\mathcal{G}$ as a *global communication graph.* In addition, for each processor p, there is a subgraph of $\mathcal{G}$ that corresponds to the local state of p at the end of P, and we refer to this subgraph as a *local communication graph.* If $\mathcal{G}$ is an r-round graph, the local communication graph for p is the is the subgraph $\mathcal{G}(p)$ containing all the information visible to p. Namely, $\mathcal{G}(p)$ consists of the node $\langle p, r \rangle$ and all earlier nodes reachable from $\langle p, r \rangle$ by a sequence (directed backwards in time) of green edges followed by at most one red edge. In the remainder of this paper, we use graphs to represent states, and the word "graph" should be substituted for the word "state" wherever we used "state" in the informal overview of Section 2.

If $\mathcal{G}$ is an r-round communication graph, then the output produced by process p in the corresponding execution can be represented as a function of the local communication graph of p at

time r. In an oblivious protocol, this output is actually a function of a *reduced* form of the local communication graph, with the processor label $\langle p, r\rangle$ removed from the final node $\langle p, r\rangle$.

4 The Bermuda Triangle

We now define the *Bermuda Triangle* B, which is the heart of our proof. For the rest of this paper, suppose there exists a protocol P solving k-set agreement in r rounds and tolerating the failure of f out of n processors, and suppose $n \geq f + k + 1$ and $rk \leq f$ (which implies $r \leq \lfloor f/k \rfloor$). We will use the Bermuda Triangle to prove that there exists an execution of P in which $k + 1$ processors choose $k + 1$ distinct values, violating the definition of k-set agreement.

We define the Bermuda Triangle B in three steps. First we describe the structure of the triangle (really, a k-dimensional simplex), and its triangulation into smaller simplexes. Next we show how to label the points of B with (global) communication graphs. Finally, we project each communication graph onto the reduced local graph of some nonfaulty processor, thus producing a labeling of points of B with reduced local communication graphs. Each simplex in the triangulation of B will be labeled with compatible local graphs.

The structure of the *Bermuda Triangle* B is defined by a k-dimensional simplex in k-dimensional Euclidean space, the k-dimensional analogue of a triangle. The *corners* of the triangle B are the $k + 1$ grid points $(0, \ldots, 0)$, $(N, 0, \ldots, 0)$, $(N, N, 0, \ldots, 0)$, $\ldots$, $(N, \ldots, N)$, where N is some huge integer to be determined in Section 4. The *points* of B are the grid points contained in B, namely the grid points of the form $x = (x_1, \ldots, x_k)$, where the x_i are integers between 0 and N satisfying $x_i \geq x_{i+1}$.

The Bermuda Triangle B is *triangulated* with respect to its points by a collection of smaller k-dimensional simplexes whose corners are points of B. We sometimes refer to them as *primitive simplexes* to distinguish them from the simplex B itself. Speaking informally, these primitive simplexes are defined as follows: pick any point of B and walk one step in the positive direction along each dimension. The set of $k + 1$ points visited by this walk are the corners of the simplex, and the triangulation consists of all simplexes determined by such walks. This is known as Kuhn's triangulation [Cha93].

We can now define the assignment of global communication graphs to points in B. We begin by defining three simple operations on communication graphs. Then we define a sequence $\sigma[v]$ of these operations that can be used to change any failure-free communication graph to the failure-free graph with all inputs equal to v, by changing just one edge or input value at a time. Finally, we use the intermediate graphs in this sequence to construct a labeling of the points of B by communication graphs.

The operations on communication graphs are as follows:

1. *delete*(i, p, q): This operation changes the color of the edge between $\langle p, i - 1\rangle$ and $\langle q, i\rangle$ to red, and has no effect if the edge is already red. This makes the delivery of the round i message from p to q unsuccessful. It can only be applied to a graph if p and q are silent in rounds $i+1$ through r.

2. *add*(i, p, q): This operation changes the color of the edge between $\langle p, i - 1\rangle$ and $\langle q, i\rangle$ to green, and has no effect if the edge is already green. This makes the delivery of the round i message from p to q successful. It can only be applied to a graph if p and q are silent in rounds $i + 1$ through r, and if p does not fail in rounds 1 through $i - 1$.

3. *change*(p, v): This operation changes the input value for processor p to v, and has no effect if the value is already v. It can only be applied to a graph if p is silent in rounds 1 through r.

In each case, since p and q are silent from the moment of the change, no other processor can detect the change.

We now define a sequence $\sigma[v]$ of graph operations that can be applied to a failure-free graph $\mathcal{G}$, resulting in another failure-free graph $\mathcal{G}[v]$ in which all processors have input v. Given a graph $\mathcal{G}$, let $\mathcal{G}_i[v]$ be a graph identical to $\mathcal{G}$, except that processor p_i has input v. Moses and Tuttle [MT88] prove a technical lemma implying that there is a "similarity chain" of graphs between $\mathcal{G}$ and $\mathcal{G}_i[v]$. The proof shows that each graph in the chain can be obtained from the preceding graph by applying a sequence of graph operations of the three kinds defined above, and that at most r processors fail in any graph in the chain. Their proof is a refinement of a similar proof by Dwork and Moses [DM90], and implies the following:

Lemma 1: If $\mathcal{G}$ is a failure-free graph, then there is a sequence $\sigma_i[v]$ of graph operations that transforms $\mathcal{G}$ into $\mathcal{G}_i[v]$ and fails at most r processors at any step.

By concatenating some of these operation sequences, we can transform $\mathcal{G}$ into $\mathcal{G}[v]$ by changing processors' input values one at a time:

Lemma 2: Let $\sigma[v] = \sigma_1[v] \cdots \sigma_n[v]$. If $\mathcal{G}$ is a failure free graph, then $\sigma[v]$ transforms $\mathcal{G}$ into $\mathcal{G}[v]$ and fails at most r processors at any step.

Now we can define the parameter N used in defining the shape of B: N is the length of the sequence $\sigma[v]$.

Next we describe how to label points in B with communication graphs. For simplicity, and without loss of generality, let $0, \ldots, k$ be the set of $k+1$ distinct input values. Informally, we will use the operations in $\sigma[1], \ldots, \sigma[k]$ along the respective dimensions $1, \ldots, k$ in B, and "merge" the results from different dimensions.

More formally, we define the *merge* of a collection $\mathcal{H}_1, \ldots, \mathcal{H}_k$ of r-round communication graphs as follows: first, an edge e is colored red if it is red in any of the graphs $\mathcal{H}_1, \ldots, \mathcal{H}_k$, and green otherwise; and second, an initial node $\langle p, 0 \rangle$ is labeled with the maximum i such that $\langle p, 0 \rangle$ is labeled with i in $\mathcal{H}_i$, (or 0 if no such i exists). The first condition says that a message is missing in the merged graph if it is missing in any of the communication graphs. To understand the second condition, study Figure 1 and notice that if we move along any line in the jth dimension, then processor input values are being changed from $j-1$ to j. If we choose a grid point x in B and move from the origin to x by moving along each dimension in turn, then the second condition is just a compact way of identifying the last dimension in which a processor's input value is changed, and hence identifying the processor's final input value.

Now let $x = (x_1, \ldots, x_k)$ be an arbitrary point of B. For each value i, let $\mathcal{F}_i$ be the failure-free communication graph in which all processors have input i. For each coordinate j, let σ_j be the prefix of $\sigma[j]$ consisting of the first x_j operations, and let $\mathcal{H}_j$ be the result of applying σ_j to $\mathcal{F}_{j-1}$. In $\mathcal{H}_j$, some subset $p_1, \ldots, p_i$ of the processors have had their inputs changed from $j-1$ to j. The graph $\mathcal{G}$ labeling x is defined to be the merge of $\mathcal{H}_1, \ldots, \mathcal{H}_k$. It turns out that $\mathcal{G}$ satisfies the consistency property required by the definition of a communication graph, and so it is actually a communication graph. We can also show that, for any set of communication graphs $\mathcal{G}_0, \ldots, \mathcal{G}_k$

labeling a primitive simplex in B, the set of processors that fail in any graph $\mathcal{G}_i$ is of size no greater than kr, which is no greater than f.

Now we define the assignment of reduced local communication graphs to points in B. Suppose that x is any point in B, and that x is labeled with global communication graph $\mathcal{G}$. Let p be any nonfaulty processor in $\mathcal{G}$, and let $\mathcal{L}$ be the reduced local communication graph of p in $\mathcal{G}$. Then $\mathcal{L}$ will be the reduced local communication graph associated with x. We can show that the local graphs labeling a simplex are guaranteed to be consistent with some global communication graph with no more than f failures:

Lemma 3: Let $\mathcal{L}_0, \ldots, \mathcal{L}_k$ be the reduced local communication graphs labeling a simplex. Then there are distinct processors $q_0, \ldots, q_k$ and a communication graph $\mathcal{G}$ with at most f faulty processors, in which all the q_i are nonfaulty and each q_i has reduced local communication graph $\mathcal{L}_i$.

5 The Lower Bound

We now state Sperner's Lemma [Spa66, p.151], and use it to prove our lower bound on the number of rounds required to solve k-set agreement.

Remember that a k-dimensional simplex S (like the Bermuda Triangle) is determined by $k+1$ grid points called corners, and an ℓ-dimensional face F of this simplex is an ℓ-dimensional simplex determined by $\ell+1$ corners of S. Both the simplex S and the face F contain some set of grid points called the points of S and F. The simplex S is triangulated with respect to its points via a collection of primitive simplexes as defined earlier. We note that these primitive simplexes partition the space defined by S, and that if a point is contained in a primitive simplex, then it is a corner of that simplex.

A *Sperner coloring* of a k-simplex S is a coloring of the points of S using $k+1$ colors such that each corner of S is colored with a distinct color, and the color of every point contained in a face F of S is the color of a corner of F. Sperner's Lemma says that Sperner colorings have a remarkable property:

Lemma 4 (Sperner's Lemma): Given a Sperner Coloring of a k-simplex S and a triangulation of S with respect to its points into primitive k-simplexes, there is a primitive k-simplex whose $k+1$ corners are colored with $k+1$ distinct colors.

Now consider the protocol P and the corresponding Bermuda Triangle B defined in the previous section, and define a coloring $\mathcal{C}_P$ of B as follows. If $\mathcal{L}$ is the reduced local communication graph labeling a point x, then color x with the value v that the assumed protocol P causes any processor to choose when $\mathcal{L}$ is its reduced local communication graph. Since P is an oblivious protocol, this coloring $\mathcal{C}_P$ is well-defined. Now we can show that $\mathcal{C}_P$ is a Sperner coloring of B, and we can apply Sperner's Lemma and find a global communication graph in which $k+1$ processors choose $k+1$ distinct values, contradicting the fact that P solves k-set agreement:

Theorem 5: If $n \geq f+k+1$, then no oblivious protocol for k-set agreement can halt in fewer than $\lfloor f/k \rfloor + 1$ rounds.

Proof: As above, suppose P is an oblivious protocol for k-set agreement tolerating f faults and halting in $r \leq \lfloor f/k \rfloor$ rounds. Let B be the Bermuda Triangle constructed as above, and $\mathcal{C}_P$ the coloring of B derived from P. Since $\mathcal{C}_P$ is a Sperner coloring of B, Sperner's Lemma 4 implies that there is a primitive simplex S in B whose corners are colored by the $k+1$ distinct values $0, \ldots, k$. Let $\mathcal{L}_0, \ldots, \mathcal{L}_k$ be the reduced local communication graphs labeling the corners of S.

Lemma 3 implies that there are distinct processors $q_0, \ldots, q_k$ and a communication graph $\mathcal{G}$ in which all the q_i are nonfaulty and each q_i has reduced local communication graph $\mathcal{L}_i$. This implies that in the execution associated with communication graph $\mathcal{G}$, the $k+1$ processors $q_0, \ldots, q_k$ collectively produce the $k+1$ distinct output values $0, \ldots, k+1$. But this contradicts the assumption that P solves the k-set agreement problem. □

6 Generalizing to the Partially Synchronous Model

As we mentioned in the introduction, one important problem left open by this paper is the generalization of our lower bound from the synchronous model to the partially synchronous model. The key to this problem may lie in work by Attiya et al. [ADLS93]. They generalize the well-known lower bound of $f+1$ rounds for consensus in the synchronous model [FL82] to $(f-1)d + Cd$ time units in the partially synchronous model, where f is the number of processor failures allowed, d is the upper bound on message delivery time, and C is the ratio between the fastest and slowest processor step times c_1 and c_2. We hope that their proof technique will help us to generalize our lower bound of $\lfloor f/k \rfloor + 1$ rounds for k-set agreement in the synchronous model to something like $(\lfloor f/k \rfloor - 1)d + Cd$ time units in the partially synchronous model, so we end this paper with a sketch of their proof.

Consider the consensus problem in which $\{0, 1\}$ is the set of input values, and suppose P is a protocol for consensus that halts in time less than $(f-1)d + Cd$. Given a finite execution α of P, a *fast, failure-free extension* of α is one in which all processors run using the fastest step time c_1 and no additional processors fail. The execution α is *v-valent* if v is the output value in every fast, failure-free extension of α, in which case v is the *valence* of α. The execution α is *univalent* if it is v-valent for some v, and *bivalent* otherwise.

The key idea in the proof is the notion of "retiming" executions—taking one execution with processors running at one speed and transforming it into another execution with processors running at another speed—and this idea is captured within a single key lemma. Let α_0 and α_1 be two executions of length $t \geq (f-1)d$, and suppose that a total of at most $f-1$ processors fail in the two executions, and that p is the only processor with different views in the two executions. The lemma states that if α_0 and α_1 are both univalent, then they have the same valence. To see this, suppose α_0 and α_1 are 0- and 1-valent, respectively. Extend both executions by failing all the processors that failed in either α_0 and α_1, plus the one processor to which the executions appear different. Allow the remaining processors to take steps using the slowest step time c_2. By the definition of consensus and the execution time of P, within an additional time less than Cd, both extensions must yield outputs. Furthermore, these outputs must be identical; without loss of generality, suppose the outputs are 0. Now modify the extension of α_1 to get a contradiction to the 1-valence assumption. Namely, shrink the slow extension so that all processors run using the fastest step time c_1; this means that the extension takes time less than d. Also, instead of failing any new processors in the extension, keep them alive but allow their messages to take the maximum delivery time d. This means that they will not arrive in time to cause any change in the

output value 0. It follows that 0 results from a fast failure-free extension of α_1, which contradicts the 1-valence of α_1, and the lemma follows.

This lemma is applied twice to prove the lower bound. First, we prove that there is a bivalent execution α of length at least $(f-1)d$ in which at most $f-1$ processors fail. If not, then all such executions α are univalent, and we can use techniques like the ones in this paper and in [FL82] to prove the existence of 0- and 1-valent executions α_0 and α_1 satisfying the hypothesis of the key lemma; but the key lemma says that they must have the same valence, which is a contradiction. Second, given the existence of a bivalent execution α, there must be a "maximal" bivalent execution α that has no bivalent extension. Using the assumption that all extensions of α terminate in an additional Cd time, there are two extensions of α of the same length, one 0-valent and the other 1-valent, and again we can use techniques like the ones in this paper and [FL82] to prove the existence of 0- and 1-valent extensions that are nearly identical. The resulting pair of executions satisfy the hypothesis of the key lemma, so they must have the same valence, which is a contradiction.

This retiming technique is interesting because it exploits the need to time out failed messages—the need for a processor to wait up to Cd time to ensure itself that d time has actually passed, and hence ensure itself that an expected but undelivered message will never arrive—which is the primary difficulty of programming in this model. We believe this technique will be helpful in the case of k-set agreement, but we have been unsuccessful so far, and this remains for future work.

References

[ADLS93] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *Journal of the ACM*, 1993. To appear. An earlier version appeared in ACM STOC, 1991.

[BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[Cha91] Soma Chaudhuri. Towards a complexity hierarchy of wait-free concurrent objects. In *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing.* IEEE, December 1991. Also appeared as Technical Report No. 91-024, Iowa State University, 1991.

[Cha93] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105:132–158, July 1993. A preliminary version appeared in ACM PODC, 1990.

[CHLT93] Soma Chaudhuri, Maurice Herlihy, Nancy Lynch, and Mark R. Tuttle. A tight lower bound for k-set agreement. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science.* IEEE, October 1993. To appear.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.

[Dol82] Danny Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, March 1982.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In Marek Karpinsky, editor, *Proceedings of the 10th International Colloquium on Automata, Languages, and Programming*, pages 127–140. Springer-Verlag, 1983. A preliminary version appeared as Yale Technical Report YALEU/DCS/RR-273.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty processor. *Journal of the ACM*, 32(2):374–382, 1985.

[Her88] Maurice Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 276–290. ACM, August 1988.

[HS93] Maurice P. Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 111–120. ACM, May 1993.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[MT88] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[Spa66] E.H. Spanier. *Algebraic Topology*. Springer-Verlag, New York, 1966.

[W+78] J. H. Wensley et al. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.