ON THE MINIMAL SYNCHRONISM NEEDED FOR DISTRIBUTED CONSENSUS

Danny Dolev*
Computer Science Department
Hebrew University
Jerusalem, Israel

Cynthia Dwork†
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Larry Stockmeyer
Computer Science Department
IBM Research Laboratory
San Jose, CA 95193

Abstract. Reaching agreement is a primitive of distributed computing. While this poses no problem in an ideal, failure-free environment, it imposes certain constraints on the capabilities of an actual system: a system is viable only if it permits the existence of consensus protocols tolerant to some number of failures. Fischer, Lynch and Paterson [FLP] have shown that in a completely asynchronous model, even one failure cannot be tolerated. In this paper we extend their work, identifying several critical system parameters, including various synchronicity conditions, and examine how varying these affects the number of faults which can be tolerated. Our proofs expose general heuristic principles that explain why consensus is possible in certain models but not possible in others.

1. INTRODUCTION

The problem of reaching agreement among separated processors is a fundamental problem of both practical and theoretical importance in the area of distributed systems; see, e.g. [Ag, DRS, DS, LSP, PSL]. We consider a system of N processors $P_1,...,P_N$ ($N \geq 2$) which communicate by sending messages to one another. Initially, each P_i has a binary value x_i. At some point during its computation, a processor can irreversibly *decide* on a binary value v. Each processor follows a deterministic protocol involving the receipt and sending of messages. Even though the individual processor protocols are deterministic, there are three potential sources of nondeterminism in the system. Processors might run at varying speeds, it might take varying amounts of time for messages to be delivered, and messages might be received in an order different from the order in which they were sent.

A protocol solves the *(weak) consensus problem* if

(i) no matter how the systems runs, every nonfaulty processor makes a decision after a finite number of steps,

(ii) no matter how the system runs, two different nonfaulty processors never decide on different values, and

(iii) 0 and 1 are both possible decision values for (possibly different) assignments of initial values. (This condition is needed to avoid the trivial solution where each processor decides 1 regardless of its initial value.)

If the processors and the communication system are completely reliable, the existence of consensus protocols is trivial. The problem becomes interesting when the protocol must operate correctly when some processors can be faulty. The failure mode studied in this paper is *fail-stop*, in which a failed processor neither sends nor receives messages. A consensus protocol is t-*resilient* if it operates correctly when at most t processors fail. The existence of N-resilient consensus protocols is easily established if the processors and the communication system are both synchronous. Intuitively, synchronous processors means that the internal clocks of the processors are synchronized to within some fixed rate of drift. Synchronous communication means that there is a fixed upper bound on the time for a message to be delivered. These two types of synchronism are assumed in much of the research on "Byzantine Agreement", e.g. [DRS, LSP].

Our point of departure and motivation for this paper was the interesting recent result of Fischer, Lynch and Paterson [FLP] which states that in a completely asynchronous system no consensus protocol is 1-resilient, that is, even one failure cannot be tolerated. In reading the proof of this result one sees that three different types of asynchrony are used:

Processor asynchrony allows processors to "go to sleep" for arbitrarily long finite amounts of time while other processors continue to run;
Communication asynchrony precludes an *a priori* bound on message delivery time;
Message order asynchrony allows messages to be delivered in an order different from the order in which they were sent.

One major goal of this work was to understand whether all three types of asynchrony are needed simultaneously to obtain the impossibility result. We find they are not. In fact, we prove the impossibility of a 1-resilient consensus protocol even if the processors operate in lock-step synchrony, thus strengthening the main result of [FLP]. In this result we retain Fischer, Lynch and Paterson's definition of an "atomic step" in which a processor can attempt to receive a message and depending on the value received, if any, it can change its internal state and send messages to all the

*Work performed in part at Stanford University, supported in part by DARPA under Grant No. MDA903-80-C-0107, and in part at IBM, San Jose.

†Work performed at Cornell University, supported in part by NSF Grant MCS81-01220.

other processors. In contrast, using this same definition of an atomic step, we prove that either synchronous communication alone or synchronous message order alone is sufficient for the existence of an N-resilient consensus protocol.

These two N-resilient protocols are fairly delicate and depend on the definition of an "atomic step" of a processor. For example, in the case where we have synchronous communication, if receiving and sending are split into two separate operations so that an unbounded amount of time can elapse in between, then the N-resilient protocol falls apart, and in fact we can prove that there is no 1-resilient protocol in this case. Similarly, if a processor can send a message to at most one other processor in an atomic step (we call this *point-to-point* transmission) then there is a 1-resilient protocol but no 2-resilient protocol.

We identify five critical parameters:
(1) processors synchronous or asynchronous,
(2) communication synchronous or asynchronous,
(3) message order synchronous or asynchronous,
(4) broadcast transmission or point-to-point transmission,
(5) atomic receive/send or separate receive and send.
In defining the system parameters, even informally as we do now, it is useful to imagine that one is standing outside the system holding a "real time clock" that ticks at a constant rate. At each tick of the real clock, at most one processor can take a step. The processors are modeled as infinite-state machines. In the most general definition of "step", a processor can attempt to receive a message, and based on the value of the received message (or based on the fact that no message was received) it can change its state and broadcast a message to all processors. Restrictions on this definition of step are given by 4 and 5 below. The letters U and F below refer to situations that are unfavorable or favorable, respectively, for solving the consensus problem; in other words, the possible behaviors of the system in a favorable situation are a subset of the possible behaviors in the corresponding unfavorable situation.

1. Processors.
U. Asynchronous. Any processor can wait an arbitrarily long, but finite, amount of real time between its own steps. (In the language of distributed systems, there is no bound on the rate of drift of the internal clocks of the processors.) However, if a processor takes only finitely many steps in an infinite run of the system, then it has failed.
F. Synchronous. There is a constant $\Phi \geq 1$ such that in any time interval in which some processor takes $\Phi + 1$ steps, every nonfaulty processor must take at least one step in that interval.

2. Communication.
U. Asynchronous. Messages can take an arbitrarily long, but finite, amount of real time to be delivered. However, in any infinite run of the system, every message is eventually delivered; i.e., messages cannot be lost.
F. Synchronous. There is a constant $\Delta \geq 1$ such that every message is delivered within Δ real time steps.

3. Message Order.
U. Asynchronous. Messages can be delivered out of order.
F. Synchronous. If p sends m_1 to r at real time t_1 and q sends m_2 to r at real time $t_2 > t_1$, then r receives m_1 before m_2. (p, q, and r are not necessarily distinct. For example, we could have $r = p$.)

4. Transmission Mechanism.
U. Point-to-point. In an atomic step, a processor can send to at most one processor.
F. Broadcast. In an atomic step, a processor can broadcast messages to all processors.

5. Receive/Send.
U. Separate. In an atomic step, a processor cannot both receive and send.
F. Atomic. Receiving and sending are part of the same atomic step.

In the next section, these definitions are formalized by modifications to the formal model of [FLP].

To obtain the strongest possible results we make some further conventions. Whenever we assume synchronous processors in an impossibility result, we actually take $\Phi = 1$, i.e., the processors operate in rounds, which is essentially the same as lock-step synchrony. Whenever we assume synchronous communication in an impossibility result, we actually take $\Delta = 1$, i.e., message delivery is *instantaneous*; in this case we assume that whenever a processor attempts to receive, it receives *all* as yet unreceived messages that have been sent to it at previous real times. In results giving a consensus protocol, our protocol actually solves a *strong consensus problem*, defined like the weak consensus problem in the Introduction with the additional condition that if all initial values are the same, say v, then all nonfaulty processors decide on v. Whenever we assume atomic receive/send in a consensus protocol, the definition 5F above can be weakened to say only that whenever a processor executes a receiving step followed by a sending step, there is a fixed upper bound on the amount of real time which can elapse between the two steps.

Varying these five parameters yields 32 cases, and we have found the maximum resiliency for each case. More interestingly, we have identified four cases where N-resilient protocols exist, but any weakening of the system by changing one parameter from favorable to unfavorable is sufficient for a proof that there is no t-resilient protocol where t is either 1 or 2. Thus the boundary between possibility and impossibility of solving the consensus problem is very sharp. These four "minimal" cases are:

(1) synchronous processors and synchronous communication,
(2) synchronous processors and synchronous message order,
(3) broadcast transmission and synchronous message order,
(4) synchronous communication, broadcast transmission, and atomic receive/send.

We find another type of boundary by allowing broadcast to k processors in an atomic step. This "k-casting" is said to be *serializable* if whenever processors p and q k-cast messages m_1 and m_2, respectively, to processors r and s, then both r and s receive the two messages in the same order. In a system with asynchronous processors and asynchronous communication, we show, for any $1 \leq k \leq N$, that serializable k-casting is sufficient for $(k-1)$-resiliency, and that serializable $(k-1)$-casting is not sufficient for $(k-1)$-resiliency. More generally, if for any two *broadcasts* there are at most $k-1$ processors for which we can guarantee that the order of reception is the same as the order of transmission, and we can say nothing at all about the order in which the other processors receive the messages, then there is no $(k-1)$-resilient consensus protocol. The delineation of these boundaries is the main contribution of this paper.

Another goal of this paper was to understand intuitively why Fischer, Lynch and Paterson were able to prove impossibility and to develop heuristic principles to allow one to make an educated guess of the maximum resiliency before actually proving it. The basic intuition is that if letting t processors fail can effectively "hide" an event or the relative ordering among several events, then no consensus protocol is t-resilient. In the original proof in [FLP], the event is a "critical step" where one processor p takes a step which moves the system from some configuration C_0 to some configuration C_1, and there are configurations D_0 and D_1 reachable from C_0 and C_1, respectively, in one step, such that v is the only possible decision value when the system is started in configuration D_v ($v = 0,1$). If p fails then the effect of the critical step can be hidden from the other processors since the communication system can hide all the messages sent by p during the critical step. However, if we have a fixed upper bound on message delivery time or if message order cannot be scrambled, then these messages cannot be hidden for an arbitrarily long time; this explains intuitively why we get N-resiliency in these two cases. The heuristic principle also explains why we get 1-resiliency but not 2-resiliency in the case of bounded message delivery time and point-to-point transmission. In the critical step, p sends a message to a single processor q. In order to hide this event, it is necessary and sufficient that both p and q fail. (Our detailed proofs are more involved than this since we must show in each case that such a critical step exists.)

Finally, we should point out that Ben-Or [BO], Bracha and Toueg [BT] and Rabin [Ra] have shown that consensus in the presence of faults can be achieved in various asynchronous environments by *probabilistic* protocols where there is some small chance that the protocol will operate incorrectly. Even in light of these probabilistic solutions, our boundaries between possibility and impossibility are fundamental to the study of distributed systems. In particular, our results identify cases where probabilistic solutions are needed to reach consensus because deterministic solutions are impossible.

In the next section we give more formal definitions. Section 3 contains the results on possibility and impossibility for the 32 choices of the parameters. In Section 4 we give the results on serializable k-casting. In Section 5, we suggest some directions for future work, such as the extension of our results to Byzantine failures [cf. DS, LSP].

2. DEFINITIONS

In this section we extend the formal framework of Fischer, Lynch and Paterson [FLP] to handle our various system parameters. A *consensus protocol* is a system of N ($N \geq 2$) processors $P = \{p_1, \ldots, p_N\}$. The processors are modeled as infinite-state machines with state set Z. There are two special *initial states* z_0 and z_1. For $v = 0,1$, a processor is started in state z_v if its initial bit is v. Each processor then follows a deterministic protocol involving the receipt and sending of messages. The messages are drawn from an infinite set M. Each processor has a *buffer* for holding the messages that have been sent to it but not yet received. If message order is synchronous, each buffer is modeled as a fifo queue of messages. If message order is asynchronous, each buffer is modeled as an unordered set of messages. The collection of buffers support two operations:

Send(p,m): places message m in p's buffer;
Receive(p): deletes some collection (possibly empty) of messages from p's buffer and delivers these messages to p.

The exact details of what messages can or must be delivered by Receive(p) depend on the choice of system parameters and this is defined precisely below.

First consider cases where message order is synchronous. Each processor p is specified by a *state transition function* δ_p and a *sending function* β_p where

$$\delta_p : Z \times M^* \rightarrow Z$$
$$\beta_p : Z \times M^* \rightarrow \{ B \subseteq P \times M \mid B \text{ is finite} \}.$$

The pair (q,m) in the range of β_p means that p sends message m to processor q. Since we place no constraints on the message set M, we can assume that for each $p,q \in P$, $z \in Z$ and $\mu \in M^*$ there is at most one message m with $(q,m) \in \beta_p(z,\mu)$. It is also convenient to assume that a processor attaches its name and a sequence number to each message so that the same message m is never sent by two different processors nor at two different times.

If transmission is *point-to-point*, then $|\beta_p(z,\mu)| \leq 1$ for every p, z and μ. If transmission is *broadcast* then p can send messages to any number of processors in one step.

If receive/send is *separate*, we assume that Z is partitioned into two disjoint sets Z_R (the *receiving states*) and Z_S (the *sending states*), such that no messages are sent when in a receiving state (formally, if $z \in Z_R$ then $\beta_p(z,\mu) = \phi$), and no messages are received when in a sending state (this condition is formalized below). It is also convenient to assume that receiving and sending states alternate, i.e., all transitions from states in Z_R must go to states in Z_S and vice versa. If receive/send is *atomic*, then any state in Z can both receive and send messages.

A *configuration* C consists of
(i) N states $st(p_i,C) \in Z$ for $1 \leq i \leq N$, specifying the current state of each processor, and
(ii) N strings $buff(p_i,C) \in M^*$ for $1 \leq i \leq N$, specifying the current contents of each buffer.

Initially, each state is either z_0 or z_1 as described above, and each buffer contains λ (the empty string).

An *event* is a pair (p,μ) where $p \in P$ and $\mu \in M^*$. Think of the event (p,μ) as the receipt of message string μ by p. Processor p is said to be the *agent* of the event (p,μ). We now define conditions under which an event can be *applied* to a configuration to yield a new configuration. The first condition applies only if receive/send is separate.

(1) If $st(p,C) \in Z_S$, then (p,μ) is *applicable* to C only if $\mu = \lambda$.

The remaining conditions apply when $st(p,C) \in Z_R$ if receive/send is separate, or in general if receive/send is atomic.

(2) If communication is *asynchronous*, (p,μ) is *applicable* to C only if $\mu \in M \cup \{\lambda\}$ and μ is a prefix of buff(p,C).

(3) If communication is *synchronous*, (p,μ) is *applicable* to C only if μ is a prefix of buff(p,C).

(4) If communication is *immediate*, (p,μ) is *applicable* to C only if $\mu = buff(p,C)$.

If the event $e = (p,\mu)$ is applicable to C, then the next configuration $e(C)$ is obtained as follows:

(a) p changes its state from $z = st(p,C)$ to $\delta_p(z,\mu)$ and the states of the other processors do not change,
(b) for all $(q,m) \in \beta_p(z,\mu)$, append m to the right end of buff(q,C),
(c) delete μ from the left end of buff(p,C).

In the case of *asynchronous message order*, the main difference in the above definitions is that each buffer is modeled as an unordered finite set. Therefore in the discussion above, M^* is replaced by the set of finite subsets of M, buff(p,C) is a finite subset of M, events have the form (p,μ) where μ is a finite subset of M, and the empty set ϕ takes the place of λ. Minor modifications to the definition of "applicable" and "next configuration" must also be made, and we leave these obvious modifications to the reader; for example, in the case of asynchronous communication, (p,μ) is applicable only if $\mu \subseteq$ buff(p,C) and $|\mu| \leq 1$.

To define synchronous processors and synchronous (but not immediate) communication and to define correctness of a protocol, we must consider sequences of events. A *schedule* is a finite or infinite sequence of events. A schedule $\sigma = \sigma_1, \sigma_2, \ldots$ is *applicable* to an initial configuration I if:

(1) the events of σ can be applied in turn starting from I, i.e., σ_1 is applicable to I, σ_2 is applicable to $\sigma_1(I)$, etc.;
(2) if processors are Φ-synchronous (constant $\Phi \geq 1$) then for every consecutive subsequence τ of σ, if some processor takes $\Phi + 1$ steps in τ and if the processor p takes no step in τ, then p takes no steps in the portion of σ following τ (this says that once a processor fails it cannot restart at a later time);
(3) if communication is Δ-synchronous (constant $\Delta \geq 1$) then, for every j, if $\sigma_j = (p,\mu)$, if message m was sent to p by the event σ_i with $i \leq j - \Delta$, and if none of the events σ_k with $i < k < j$ is the receipt of m by p, then m belongs to μ (this says that messages must be delivered within Δ real time steps).

If $\Phi = 1$ in (2), the processors are said to be *lock-step*.

If σ is finite, $\sigma(I)$ denotes the resulting configuration which is said to be *reachable* from I. A configuration reachable from some initial configuration is said to be *accessible*. Henceforth, all configurations mentioned are assumed to be accessible. If Q is a set of processors, the schedule σ is *Q-free* if no $p \in Q$ takes a step in σ. A schedule together with the associated sequence of configurations is called a *run*.

We assume that there are two disjoint sets of *decision states* Y_0 and Y_1, such that if a processor enters a state in Y_v ($v \in \{0,1\}$) then it must remain in states in Y_v. A configuration C has *decision value* v if $st(p,C) \in Y_v$ for some p. A consensus protocol is *partially correct* if

(1) no accessible configuration has more than one decision value, and
(2) for each $v \in \{0,1\}$, some accessible configuration has decision value v.

A processor p is *nonfaulty* in an infinite run if it takes infinitely many steps and is *faulty* otherwise. For $0 \leq t \leq N$, an infinite run is a t-*admissible run from* I if:

(1) the associated schedule is applicable to I,
(2) at most t processors are faulty, and
(3) all messages sent to nonfaulty processors are eventually received.

A run is a *deciding run* if every nonfaulty processor enters a decision state. A protocol is a t-*resilient protocol for the weak consensus problem* if it is partially correct and every t-admissible run from every initial configuration is a deciding run. A protocol is a t-*resilient protocol for the strong consensus problem* if it is partially correct, every t-admissible run from every initial configuration is a deciding run, and if I_v is the initial configuration in which all processors have initial value v then all deciding configurations reachable from I_v have decision value v.

For the purposes of our impossibility proofs we would also like to define when a schedule is applicable to a noninitial configuration C. In the case of synchronous processors or synchronous communication it would seem that the definition would have to depend not only on C but also on the history of events that led to C. However, the full history is not needed since in all of the impossibility proofs given in detail in this paper, one of the following two situations hold:

(1) processors are asynchronous and communication is either asynchronous or immediate, or
(2) processors are lock-step and communication is asynchronous.

For situation (1), clearly the history is irrelevant, and the definition of σ being applicable to C is identical to the definition above for initial C. The definitions of $\sigma(C)$ and "reachable" are also identical. For situation (2), the definition requires a little extra technical machinery and this will be developed in the proof of Theorem 10 which is the only place where it is used. Whenever we give an impossibility proof in a model with lock-step processors (resp., immediate communication), a modification of the proof also shows impossibility in the same model but with Φ-synchronous processors, $\Phi > 1$, (resp., Δ-synchronous communication, $\Delta > 1$). These modifications will appear in the final version of this paper.

For configuration C let V(C) be the set of decision values of configurations reachable from C. Configuration C is *bivalent* if $V(C) = \{0,1\}$, or *univalent* otherwise. Univalent configurations are either 0-*valent* if $V(C) = \{0\}$, or 1-*valent* if $V(C) = \{1\}$. The following obvious fact is used often in our proofs:

For $v = 0,1$, if C is v-valent and D is reachable from C then D is v-valent.

3. THE PRINCIPAL BOUNDARIES

Since our impossibility proofs follow the general outline used by Fischer, Lynch and Paterson [FLP], it is worthwhile to first review this outline. The proof assumes the existence of a t-resilient protocol and reaches a contradiction. There are three steps.

I. Show that there is a bivalent initial configuration.

II. Show that if C is a bivalent configuration and p is a processor, then there is a schedule σ such that $\sigma(C)$ is bivalent and p takes a step in σ. Moreover, if p's buffer is nonempty in C, then for any message m in p's buffer there is such a σ in which p receives m. (This is the difficult part.)

III. Using I and II construct an infinite t-admissible run which is not deciding as follows. Let B_1 be an initial bivalent configuration. In general, if B_i is bivalent, let $p = p_j$ where $j \equiv i \pmod N$ and let $B_{i+1} = \sigma(B_i)$ where σ is obtained from II. Moreover, if p's buffer is nonempty in B_i let p receive a message which has been in the buffer for the longest time. The resulting infinite run is 0-admissible. It is not a deciding run because, by partial correctness, a bivalent configuration has no decision value.

Although our proofs follow the general outline of [FLP], in most cases we lose the "commutativity" property of schedules which was used heavily in [FLP]. Therefore, we have had to develop new techniques beyond those used in [FLP]. First we review the first step of the outline.

Lemma 3.1. (Fischer, Lynch, Paterson [FLP]). For any choice of the system parameters and any $t \geq 1$, if a protocol is t-resilient and solves the weak consensus problem then the protocol has a bivalent initial configuration.

Proof. Suppose otherwise that all initial configurations are either 0-valent or 1-valent. Since partial correctness implies that 0 and 1 are both possible decision values, there must be initial configurations I_0 and I_1 such that I_v is v-valent. By changing the initial values in which I_0 and I_1 differ, one at a time, we can find initial configurations J_0 and J_1 such that J_v is v-valent and J_0 and J_1 differ in the initial value of exactly one processor, say p. Consider a finite deciding run from J_0 in which p takes no steps; such a run must exist by t-resiliency. Letting σ be the associated schedule, σ is applicable to J_1 also and the same decision 0 is reached in both cases. This contradicts the 1-valency of J_1. □

Another basic lemma from [FLP], variations of which are used in most of our proofs, is the following.

Lemma 3.2 ([FLP]). Suppose that processors, communication and message order are all asynchronous. Let C be a bivalent configuration and let $e = (p,m)$ be an event applicable to C. Let $\mathscr{C}$ be the set of configurations reachable from C without applying e and let $\mathscr{D} = \{ e(E) \mid E \in \mathscr{C} \}$. If $\mathscr{D}$ contains no bivalent configuration then $\mathscr{D}$ contains both a 0-valent and a 1-valent configuration.

Proof. Suppose otherwise that, for some v, $\mathscr{D}$ contains only v-valent configurations. Since C is bivalent, there is a schedule σ such that $\sigma(C)$ has decision value $\sim$v. If e is an event in σ, then writing $\sigma = \sigma_1 e \sigma_2$ where e does not occur in σ_1 and letting $D = e(\sigma_1(C))$, we have $D \in \mathscr{D}$ but D cannot be v-valent because $\sigma_2(D)$ has decision value $\sim$v. If e is not in σ then $\sigma(C) \in \mathscr{C}$ and e is applicable to $\sigma(C)$ so $e(\sigma(C)) \in \mathscr{D}$ and it cannot be v-valent. □

The main result of [FLP] is that in the model with processors, communication and message order all asynchronous (and the other two parameters favorable) there is no 1-resilient protocol for the weak consensus problem. Our first result strengthens this by allowing synchronous, even lock-step, processors. In general, the letters I and E in the names of our theorems refer to impossibility and existence of protocols, respectively.

Theorem I0. In the model with asynchronous communication and asynchronous message order (and the other three parameters favorable), there is no 1-resilient weak consensus protocol. Moreover, this is true if processors are lock-step synchronous.

Proof. We give the proof for lock-step processors. The proof needs a few new definitions and lemmas. Since processors are lock-step assume that in any schedule or run, the processors take turns in the order $p_1, p_2, \ldots, p_N, p_1, p_2, \ldots$. If C is a configuration, let turn(C) be the number of the unique processor whose turn it is. That is, turn(I) = 1 for every initial I and if C′ follows in one step from C then turn(C′) = turn(C) + 1 (mod N). The definition of a schedule σ being applicable to C is identical to the definition in Section 2 for initial C with the added condition that if turn(C) = i then p_i is the agent of the first event in σ. To allow processors to fail, we introduce the notion of a "failure step" as an expositional convenience. The corresponding event is denoted $(p,\dagger)$. The event $(p_i,\dagger)$ is applicable to C iff turn(C) = i. The next configuration $(p_i,\dagger)(C)$ is identical to C except that its turn is incremented. To the definition of applicable schedule add the condition that if a processor takes a failure step then all of its subsequent steps are failure steps. A *round* is an N-event schedule $\sigma_1, \ldots, \sigma_N$ such that if q_i is the agent of σ_i for $1 \leq i \leq N$ then $q_1, \ldots, q_N$ is a cyclic shift of $p_1, \ldots, p_N$. A schedule or run is *failure-free* if no processor takes a failure step. A configuration C is *ff-bivalent* if there are configurations D_0 and D_1 reachable from C by failure-free runs such that D_v has decision value v for v = 0,1.

Lemma I0.1. C is ff-bivalent iff C is bivalent.

Proof. Clearly ff-bivalency implies bivalency. The other direction follows immediately from the following fact: If C can reach a configuration with decision value v by a finite run R then C can reach a configuration with decision value v by a failure-free run. To see this let σ be the schedule associated with R. If σ contains no failure events we are done. Say then that p takes failure steps in σ. Consider any schedule σ' identical to σ except that p takes no failure steps. Since communication and message order are asynchronous, σ' is applicable to C also since any message sent by p in σ' but not sent by p in σ can be delayed until after the decision is reached. (We are using here the fact that R is finite.) The failure-free run obtained by applying σ' to C leads to decision v. □

Lemma I0.2. Let C be a bivalent configuration reachable from some initial configuration by a failure-free run, and let $e = (p,m)$ be an event applicable to C with $m \in M \cup \{\phi\}$. Let $\mathscr{C}$ be the set of configurations reachable from C by zero or more failure-free rounds in which the event e is not applied, and let $\mathscr{D} = \{e(E) \mid E \in \mathscr{C}\}$. Then $\mathscr{D}$ contains a bivalent configuration.

Proof. Suppose otherwise that $\mathscr{D}$ contains only univalent configurations. If $buff(p,C) = \phi$ then $e = (p,\phi)$ is the only non-failure event applicable to C, so the bivalency of C implies the bivalency of e(C). Say then that $buff(p,C) \neq \phi$ so at least two events are applicable to C.

As in the proof of Lemma 3.2, it is easy to show that $\mathscr{D}$ contains both a 0-valent and a 1-valent configuration. In carrying out the details of the proof, first use Lemma I0.1 to conclude that C is ff-bivalent, so for each $v \in \{0,1\}$ there is a failure-free run from C that leads to decision value ~v, so the configurations in $\mathscr{D}$ cannot all be v-valent. Now by an easy induction there are configurations $C_0, C_1 \in \mathscr{C}$ and a single failure-free round ρ such that $e(C_v)$ is v-valent for v=0,1 and either $C_0 = \rho(C_1)$ or $C_1 = \rho(C_0)$. Say w.l.o.g. that $C_1 = \rho(C_0)$. For $v \in \{0,1\}$, let $D_v = e(C_v)$ so D_v is v-valent. Write $\rho = f\tau$ where $f = (p,m')$ and $m \neq m'$ and write

$$\tau = (q_2,m_2), (q_3,m_3), ..., (q_N,m_N).$$

Let $\tau' = (q_2,\phi), (q_3,\phi), ..., (q_N,\phi)$, and $\rho' = f\tau'$. If $e(\rho'(C_0))$ is bivalent we are done since $e(\rho'(C_0)) \in \mathscr{D}$. If $e(\rho'(C_0))$ is not bivalent, we claim that it is 1-valent. Suppose otherwise that it is 0-valent. For $1 \leq j \leq N$, let

$$\tau_j = (q_2,n_2), ..., (q_N,n_N)$$

where $n_i = m_i$ if $i \leq j$ and $n_i = \phi$ if $i > j$, and let $\rho_j = f\tau_j$. In particular, $\rho_1 = \rho'$ and $\rho_N = \rho$. Note that $\rho_j e$ is applicable to C_0 for all j because e is applicable to C_0 and e is not applied in ρ_j. Since $e(\rho_1(C_0))$ is 0-valent and $e(\rho_N(C_0))$ is 1-valent, there must exist a j such that $e(\rho_j(C_0))$ is 0-valent and $e(\rho_{j+1}(C_0))$ is 1-valent. The only difference between ρ_j and ρ_{j+1} is that ρ_j contains (q_j,ϕ) where ρ_{j+1} contains (q_j,m_j). Let η be identical to ρ_j except that the event with agent q_j in η is $(q_j,\dagger)$. Note that ηe is applicable to C_0 also. Let σ be a q_j-free finite schedule applicable to $G_2 = e(\eta(C_0))$ such that the associated run is deciding. Then σ is applicable to $G_0 = e(\rho_j(C_0))$ and to $G_1 = e(\rho_{j+1}(C_0))$. Since $st(r,G_0) = st(r,G_1) = st(r,G_2)$ for all $r \neq q_j$, the same decision is reached when σ is applied to G_0, G_1, and G_2. But this contradicts either the 0-valency of G_0 or the 1-valency of G_1 and completes the proof that $E_1 = e(\rho'(C_0))$ is 1-valent.

Our purpose in changing τ to τ' is to ensure that no event in τ' is the receipt of a message sent by the event f. Therefore, $[\tau',(p,\dagger)]$ is applicable to D_0; let E_0 be the resulting configuration. Also, $[(p,\dagger),\tau',(p,\dagger)]$ is applicable to C_0; let E_2 be the resulting configuration. See Figure 1. (We indicate v-valency in figures by writing v inside a small box.) Let σ be a p-free schedule applicable to E_2 such that the associated run is deciding. Since σ is applicable to both E_0 and E_1 and since $st(r,E_0) = st(r,E_1) = st(r,E_2)$ for all $r \neq p$, the same decision is reached in all three cases, which contradicts either the 0-valency of E_0 or the 1-valency of E_1. □

Using Lemmas 3.1 and I0.2, the proof of Theorem I0 is now completed as described in part III of the general outline. □

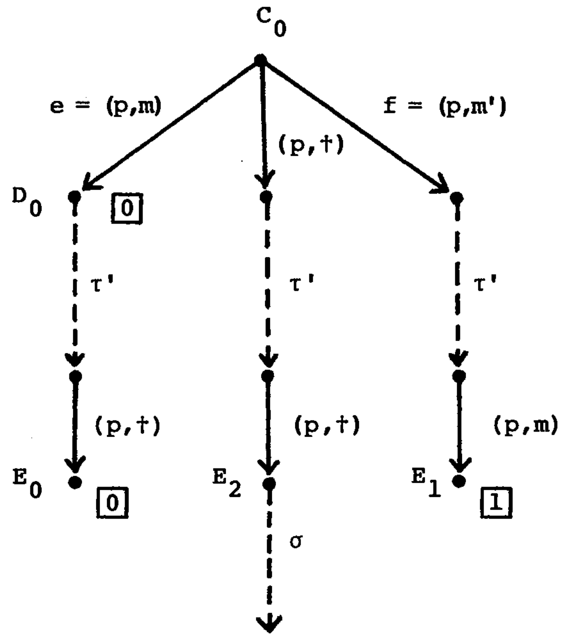

Figure 1.

Another definition and simple lemma will be used often in the remaining proofs.

Definition. Let $X \subseteq P$. Two configurations C and D are *X-equivalent* if $st(p,C) = st(p,D)$ and $buff(p,C) = buff(p,D)$ for all $p \in P-X$.

Lemma 3.3. Assume a model in which processors are asynchronous and communication is either asynchronous or immediate. In any t-resilient consensus protocol, there do not exist a 0-valent D_0, a 1-valent D_1 and a set $X \subseteq P$ with $|X| \leq t$ such that D_0 and D_1 are X-equivalent.

Proof. Suppose otherwise. By X-equivalence, if σ is an X-free finite schedule applicable to D_0 such that the associated run is deciding, then σ is applicable to D_1 also and the same decision is reached in both cases. □

By Theorem I0 we can henceforth restrict attention to models where either communication or message order is synchronous. In each case we identify models where N-resilient protocols exist, but the model cannot be weakened and still have N-resilient protocols. We first consider synchronous communication.

Theorem E1. If communication is synchronous, transmission is broadcast and receive/send is atomic (and the other two parameters are unfavorable), there is an N-resilient strong consensus protocol.

Proof. We describe the protocol for p_i. First p_i broadcasts its name and initial value (i,x_i). p_i then attempts to receive messages for 2Δ of its own steps (where communication is Δ-synchronous). If it receives a message "Decide v", it decides v. If it receives (j,x_j) from some other processor, it remembers x_j and attempts to receive for 2Δ more steps. If at some point it has run for 2Δ steps without receiving any messages it sees if all initial values received from other processors are the same as its own initial value. If so it decides on this common value v; if not it decides v=0. In either case, it broadcasts "Decide v".

Termination of the protocol is obvious: since at most N messages of the form (i,x_i) are sent, a processor can run for at most $2\Delta N$ of its own steps before deciding. It is also clear that if all initial bits have the same value v, then all nonfaulty processors decide v. If the initial bits are not all the same, it remains to show that there is no run with two different decision values. Suppose there is such a run. Number the steps 1,2,3,.... These are the "times" at which the steps occur. Let p be the processor that makes the earliest broadcast of a message "Decide v" for some v. If some other processor decides ~v, let q be the processor that decides ~v earliest. Before deciding, both p and q attempted to receive for 2Δ steps but received no messages. Let s_p be the time of p's first such attempt to receive, and let d_p be the time when p decides. Let s_q be the time of q's first attempt, let m_q be the time of q's Δ^{th} attempt, and let d_q be the time when q decides. If $m_q > d_p$ then q will receive the message "Decide v" from p before time d_q, so q will also decide v. Therefore, we must have $m_q < d_p$. Since $m_q \geq s_q + \Delta$, it follows that $d_p > s_q + \Delta$, so any message received by q before time s_q will be received by p before time d_p. Similarly, since $d_q > d_p > s_p + \Delta$, any message received by p before time s_p will be received by q before time d_q. Therefore, p at time d_p has collected exactly the same set of initial values as q has collected at time d_q. Since q is the earliest processor to decide ~v, q has not received a message "Decide ~v" from some other processor. Therefore, p and q must decide the same. □

The next two results show the effect of replacing broadcast transmission by point-to-point transmission in the protocol of Theorem E1. We find the unusual phenomenon that 1-resiliency is possible but 2-resiliency is not.

Theorem E1.1. If the model of Theorem E1 is weakened by having point-to-point transmission, then there is a 1-resilient strong consensus protocol.

Proof. The proof follows easily from Theorem E1. Let p_1 and p_2 run the protocol of Theorem E1. In this protocol, a processor need never send to itself, so atomic sending to N–1, or "(N–1)-casting" suffices. Since there are only two processors participating, point-to-point is equivalent to (N–1)-casting. When one of p_1 or p_2 (or both) decides, it sends the decision to all others. □

Theorem I1.1. Assume $N \geq 3$. If the model of Theorem E1 is weakened by making transmission point-to-point, there is no 2-resilient weak consensus protocol. Moreover, this is true even if message order is synchronous and communication is immediate.

Proof. We give the proof for immediate communication and synchronous message order. Assume there is a 2-resilient protocol in this model. Since communication is immediate, we can name an event simply by naming the processor that takes the step, since the message string received when p takes a step from C is buff(p,C) by definition. Let p(C) denote the configuration reached when the event (p,buff(p,C)) is applied to C.

Lemma I1.1.1. There do not exist a bivalent C and v-valent D_v for v=0,1, and distinct processors p and q such that $D_0 = p(C)$ and $D_1 = q(C)$.

Proof. Suppose otherwise. Recall that a processor can send to at most one processor in a step. If the set of processors receiving messages sent by the events p and q is contained in {p,q}, then D_0 and D_1 are {p,q}-equivalent, which contradicts Lemma 3.3. If one of p or q sends to some $r \notin \{p,q\}$, say that p sends to r. Since $st(q,C) = st(q,D_0)$ and $buff(q,C) = buff(q,D_0)$, q will act the same when q is applied to D_0 as when q is applied to C. But $q(D_0)$ is 0-valent, and $q(D_0)$ and D_1 are {p,r}-equivalent, again a contradiction. □

Lemma I1.1.2. Let C be a bivalent configuration and let p and q be distinct processors. There is a configuration E reachable from C such that either p(E) or q(E) is bivalent.

Proof. If either p(C) or q(C) is bivalent, we are done. By Lemma I1.1.1, p(C) and q(C) cannot have different univalencies, so say that they are both 0-valent. Since C is bivalent, there is a finite deciding run from C with decision value 1. If D is a configuration in R with decision value 1, then p(D) and q(D) are obviously both 1-valent. Therefore, there are configurations A and B in R and a processor r such that B = r(A), p(A) and q(A) are both 0-valent, but p(B) and q(B) are not both 0-valent. Note that $r \neq p$ and $r \neq q$ since p(A) and q(A) are both 0-valent but B (= r(A)) is not 0-valent. If p(B) or q(B) is bivalent, we are done. We show that p(B) and q(B) both univalent leads to a contradiction. By Lemma I1.1.1 and the choice of B, p(B) and q(B) must both be 1-valent. Let s be the processor to which r sends when it takes the step from A to B. Say w.l.o.g. that $s \neq p$. Now it is easy to see that p(A) and p(B) are {r,s}-equivalent, which contradicts Lemma 3.3. □

To complete the proof of Theorem I1.1, we must slightly modify step III of the outline. Starting from a bivalent initial configuration, we again try to let processors take steps, in turn, while keeping the system in a bivalent configuration. If at some point we cannot let p take a step and maintain bivalency, we use Lemma I1.1.2 to let the other processors take steps, in turn, while staying in bivalent configurations. Even in this case we have constructed a 1-admissible infinite nondeciding run. □

The next result shows the effect on the protocol of Theorem E1 by separating the Receive and Send operations.

Theorem I1.2. If the model of Theorem E1 is weakened by making receive/send separate, there is no 1-resilient weak consensus protocol. Moreover, this is true if communication is immediate.

Proof. As in the previous proof, we specify an event by naming the processor that takes the step. Furthermore, the event Rec(p) (Send(p)) means that p takes a step when in a receiving (sending) state. Given a bivalent C and an event e applicable to C, let $\mathscr{C}$ be the set of configurations reachable from C without applying e and let $\mathscr{D} = \{e(E) \mid E \in \mathscr{C}\}$. We show by contradiction that $\mathscr{D}$ contains a bivalent configuration. If $\mathscr{D}$ does not contain a bivalent configuration, then by a proof very similar to the proof of Lemma 3.2, $\mathscr{D}$ contains both a 0-valent and a 1-valent configuration. Therefore, we can find configurations $C_0, C_1 \in \mathscr{C}$ and an event f such that $C_1 = f(C_0)$ and $e(C_0)$ and $e(C_1)$ have different univalencies. Say w.l.o.g. that $D_v = e(C_v)$ is v-valent. Let p be the agent of e and q be the agent of f. Since processors are deterministic, $p \neq q$. There are four cases, and in each case we find an equality or equivalence that contradicts Lemma 3.3.

(1) e = Send(p) and f = Send(q).

Let $D_0' = f(D_0)$; D_0' is 0-valent. Since message order is asynchronous,

$$D_0' = f(e(C_0)) = e(f(C_0)) = D_1.$$

(2) e = Rec(p) and f = Rec(q).

Again, $f(D_0) = D_1$.

(3) e = Rec(p) and f = Send(q).

Let $D_0' = f(D_0)$. Since $st(q,C_0) = st(q,D_0)$ and $buff(q,C_0) = buff(q,D_0)$, q acts the same when it takes a step from C_0 as when it takes a step from D_0; in particular, the messages sent are the same in the two cases. It follows that D_0' and D_1 are {p}-equivalent.

(4) e = Send(p) and f = Rec(q).

By an argument similar to case (3), D_0 and D_1 are {q}-equivalent.

Having shown that $\mathscr{D}$ must contain a bivalent configuration, the proof is completed by III of the outline. □

Theorem E2. If processors and communication are both synchronous (and the other three parameters are unfavorable), then there is an N-resilient strong consensus protocol.

Theorem E2 is well-known. A processor can tell if another has failed by using "time-outs". Consensus can be reached by simplifying any algorithm which reaches Byzantine agreement, e.g., [DS].

Theorem I2. If the model of Theorem E2 is weakened by making processors asynchronous, there is no 1-resilient weak consensus protocol. Moreover, this is true even if message order is synchronous and communication is immediate.

Proof. The proof is very similar to the proof of Theorem I1.2. The only difference is in case (1), the only place where we used asynchronous message order. In this proof, message order is synchronous but transmission is point-to-point. The new case (1) is as follows.

(1') e = Send(p) and f = Send(q).

Let $D_0' = f(D_0)$. If p and q do not send to the same processor in the two events e and f, then $D_0' = D_1$. If p and q send to the same processor r, then D_0' and D_1 are {r}-equivalent. □

The next group of results have synchronous message order.

Theorem E3. If message order is synchronous and transmission is broadcast (and the other three parameters are unfavorable), there is an N-resilient strong consensus protocol.

Proof. The first step of each processor is to broadcast its initial value. It then attempts to receive and decides on the first value received. Since message order is synchronous, the first value broadcasted will be the value decided by all. □

Remark. For Theorem E3 a weaker definition of synchronous message order suffices: Whenever p broadcasts a message m_p and q broadcasts a message m_q, then m_p and m_q appear in the same order in the queues of all processors. The exact order does not matter as long as it is the same for all processors.

Theorem I3. If the model of Theorem E3 is weakened by making transmission point-to-point, there is no 1-resilient weak consensus protocol. Moreover, this is true even if receive/send is atomic.

Theorem I3 is obtained as a corollary of a more general result in the next section.

Theorem E4. If message order is synchronous and processors are Φ-synchronous for some constant $\Phi \geq 1$ (and the other three parameters are unfavorable), there is an N-resilient strong consensus protocol.

Proof. The proof is by induction on N. The basis N = 1 is obvious. Say that N > 1. Let $\mathscr{P}$ be the (N–1)-resilient consensus protocol for N–1 Φ-synchronous processors that exists by the induction hypothesis.

On every other one of its own steps, each processor p_i with $1 \leq i \leq N-1$ runs the protocol $\mathscr{P}$. When not running $\mathscr{P}$, p_i sends the message "p_i is alive" to p_N. If p_i decides in the protocol $\mathscr{P}$, it first sends the decision value to p_N and then p_i itself decides. Since $\mathscr{P}$ requires only that processors be Φ-synchronous, it can be seen that $p_1, ..., p_{N-1}$ simulate $\mathscr{P}$ correctly. On every other one of its own steps, p_N sends a message to itself. On its other steps, p_N attempts to receive. If at some point p_N has received a sequence of $\Phi + 1$ of its own messages without receiving a message "p_i is alive" between two of its own messages in the sequence, then p_N concludes that p_i has failed. If p_N concludes that $p_1, ..., p_{N-1}$ have all failed, it decides on its own initial value.

The correctness proof has two cases. (1) If some p_i reaches a decision in $\mathscr{P}$ and sends the decision to p_N before failing, then p_N will receive this decision before p_N can conclude that p_i has failed. (2) If p_1, ..., p_{N-1} all fail before sending a decision to p_N, then p_N will eventually discover this and decide on its own initial value. □

It follows from previous results that any weakening of the model of Theorem E4 cannot tolerate one failure. These theorems cover all 32 cases of choosing the system parameters.

4. ANOTHER TYPE OF BOUNDARY

In this section we consider models where the transmission mechanism is intermediate between point-to-point and full broadcast and where message order is intermediate between synchronous and asynchronous.

Definition. A model supports k-*casting*, $1 \leq k \leq N$, if a processor can send to at most k processors in an atomic step. (In particular, broadcasting as defined previously is N-casting and point-to-point is 1-casting.) The k-casting is s-*serializable*, $1 \leq s \leq k$, if for any two k-casts from p to the set of processors Q_p and from q to the set of processors Q_q, there are at least $\min(s, |Q_p \cap Q_q|)$ processors in $Q_p \cap Q_q$ that must receive the messages in the order in which they were sent, but there are no constraints on the order in which the other processors receive the messages. The set of processors that must receive in the correct order depends only on p, q, Q_p and Q_q.

In this section, processors and communication are asynchronous. The choice of the receive/send parameter is irrelevant.

Theorem E5. For any k, $1 \le k \le N$, k-serializable k-casting is sufficient for (k–1)-resiliency.

Proof. Let $S = \{p_1,...,p_k\}$. Each processor in S k-casts its initial value to all processors in S. Each then attempts to receive and decides on the first value received. This decision value is then sent to p_{k+1}, ..., p_N. Since the k-casting is k-serializable, all processors in S receive the same initial value first. Since at most k–1 in S can fail, at least one will decide and send the decision to P–S. □

Theorem I5. For any k, $2 \le k \le N$, (k–1)-serializable broadcasting is not sufficient for (k–1)-resiliency.

Lemma I5.1. There is no configuration C, events e = (p,m) and f = (q,n) with $p \neq q$, and $v \in \{0,1\}$ such that e(C) is v-valent and e(f(C)) is ~v-valent.

Proof. Since $p \neq q$, f is applicable to e(C). Let D = e(f(C)) and E = f(e(C)) so D is ~v-valent and E is v-valent. Let Q be the set of processors that must receive the messages from the two broadcasts e and f in the same order, so $|Q| \le k-1$. Any Q-free finite deciding run applicable to D is also applicable to E and the same decision is reached in both cases since st(r,D) = st(r,E) for all r. □

Lemma I5.2. Let C be bivalent and let e = (p,m) be an event applicable to C. Let $\mathscr{C}$ be the set of configurations reachable from C without applying e and let $\mathscr{D} = \{ e(E) \mid E \in \mathscr{C} \}$. Then $\mathscr{D}$ contains a bivalent configuration.

Proof. Say that $\mathscr{D}$ contains only univalent configurations. By Lemma 3.2 and a simple induction as in preceding proofs, there are configurations $C_0, C_1 \in \mathscr{C}$ and an event f = (q,n) such that $C_1 = f(C_0)$, $D_0 = e(C_0)$ is 0-valent, and $D_1 = e(C_1)$ $(= e(f(C_0)))$ is 1-valent. By Lemma I5.1, p = q. Let R be a p-free finite deciding run from C_0. If R contains a configuration E with decision value 1, then since $e(C_0)$ is 0-valent and e(E) is 1-valent, there must be configurations A and B in R such that B = g(A) for some event g (the agent of g is not p since R is p-free), e(A) is 0-valent and e(B) (= e(g(A))) is 1-valent. This contradicts Lemma I5.1. Therefore, R has decision value 0.

By a similar argument, there must be configurations B_1 and B_0 in R and an event b such that $B_0 = b(B_1)$, $[fe](B_0)$ is 0-valent and $[fe](B_1)$ is 1-valent. Since the agent of b is not p, the schedule [be] is applicable to $f(B_1)$. See Figure 2. If $[fbe](B_1)$ is bivalent, we are done because this configuration belongs to $\mathscr{D}$. Let Q, with $|Q| \le k-1$, be the set of processors that receive the messages from the two broadcasts b and f in the same order. Any Q-free finite deciding run applicable to $[fbe](B_1)$ is also applicable to $[bfe](B_1)$ and the same decision is reached in both cases. Therefore, $[fbe](B_1)$ is 0-valent because $[bfe](B_1)$ is 0-valent. Letting $B_2 = f(B_1)$, $e(B_2)$ is 1-valent and $e(b(B_2))$ is 0-valent, which contradicts Lemma I5.1. □

The proof of Theorem I5 is now completed as in part III of the outline. □

Since the model with point-to-point transmission and synchronous message order is a special case of the the model with 1-serializable broadcasting, Theorem I3 is an immediate corollary of Theorem I5 with k=2. Also note that the completely asynchronous model of [FLP] is precisely 0-serializable broadcasting, so the impossibility result of [FLP] also follows from Theorem I5.

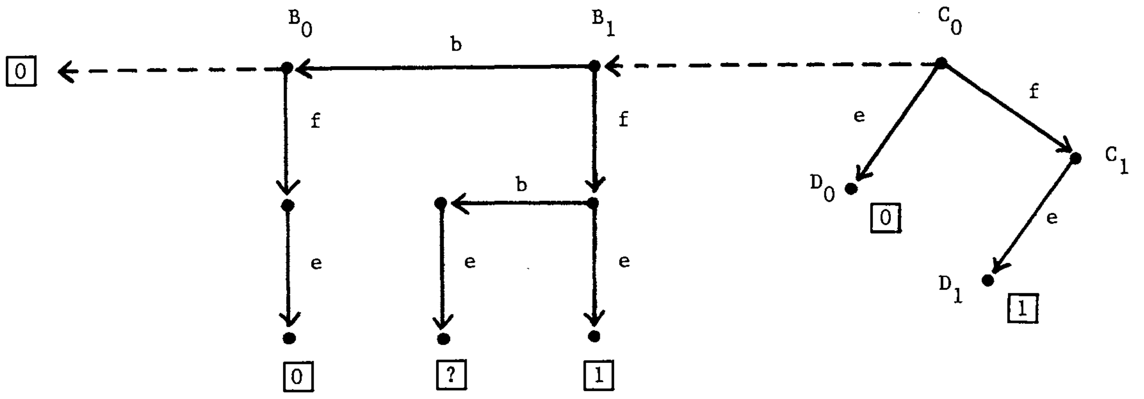

Figure 2.

5. OPEN QUESTIONS

A number of directions for future research come easily to mind.

(1) Byzantine failures.

Most of the research on reaching agreement in the "standard" synchronous model with synchronous processors and synchronous communication (Theorem E2) has dealt with Byzantine failures where processors can send erroneous messages. For example, given a reasonable definition of correctness of a consensus protocol in the case of Byzantine failures, then with authentication (i.e., a processor can attach its unforgable signature to any message) it is known that there is a strong consensus protocol which is resilient to any number of Byzantine failures [DS]; without authentication, t-resilient consensus is possible iff $t \leq \lfloor (N-1)/3 \rfloor$ [LSP]. What is the effect of Byzantine failures on our other protocols? If the model has broadcast (or k-cast) transmission (Theorems E1, E3 and E5), the answer might depend on whether a Byzantine processor is forced to broadcast (or k-cast) a message, including erroneous messages, whenever the correct action calls for a broadcast (or k-cast). In contrast, a possibly more destructive type of behavior would be to send a message to some processors but not to others. One model where this might matter is the model of Theorem E3. If Byzantine processors are forced to broadcast, a trivial modification to the protocol of Theorem E3 is still correct and N-resilient, since the only thing that matters in this protocol is that the same message is broadcast to all processors at the same time. However, if a Byzantine processor can send to some but not others, then this particular protocol is not correct. We are presently pursuing the problem of introducing Byzantine failures into the model of Theorem E1.

(2) Complexity.

Once one knows that a consensus protocol exists in a certain model, it then becomes interesting to place bounds on various measures of efficiency such as the time to reach agreement and the number of messages that must be sent. In the "standard" synchronous model, considerable work has been done on these efficiency issues, e.g., [DR,DS]. One specific open question for the model of Theorem E1 is the following. We noted in the proof of Theorem E1 that every nonfaulty processor decides after at most $2\Delta N$ of its own steps. If we only want to solve the weak consensus problem, there is a similar protocol in which every nonfaulty processor decides after at most 2Δ steps. Is there a constant c such that for any number N of processors, there is an N-resilient strong consensus protocol for this model in which every nonfaulty processor decides within $c\Delta$ of its own steps?

(3) Asynchronous start.

In the two protocols that assume synchronous processors (Theorems E2 and E4), we also assume that the processors all start running the protocol at roughly the same time. A weaker assumption would be to place no constraint on when the processors start (except that every nonfaulty processors must start at some time in any infinite run), but once a processor does start it must then respect the definition of Φ-synchronicity (until it fails). Are there N-resilient consensus protocols in the models of Theorems E2 and E4 using this weaker assumption?

Another general direction is to study the effect of network topology and network failures.

Acknowledgment. The second author thanks Nancy Lynch and Dale Skeen for helpful discussions.

REFERENCES

Ag. H. Aghili, M. Astrahan, S. Finkelstein, W. Kim, J. McPherson, M. Schkolnick, and R. Strong, A prototype for a highly available database system, Report RJ3755, IBM Research Division, San Jose, CA, 1983.

BO. M. Ben-Or, Another advantage of free choice: completely asynchronous agreement protocols, *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, 1983.

BT. G. Bracha and S. Toueg, Resilient consensus protocols, *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, 1983.

DR. D. Dolev and R. Reischuk, Bounds on information exchange for Byzantine agreement, *Proc. ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1982, 132-140.

DRS. D. Dolev, R. Reischuk, and H. R. Strong, Eventual is earlier than immediate, *Proc. 23rd Annual IEEE Symp. on Foundations of Computer Science*, 1982, 196-203.

DS. D. Dolev and H. R. Strong, Polynomial algorithms for multiple processor agreement, *Proc. 14th ACM Symp. on Theory of Computing*, 1982, 401-407.

FLP. M. J. Fischer, N. A. Lynch and M. S. Paterson, Impossibility of distributed consensus with one faulty process, *Proc. 2nd ACM Symp. on Principles of Database Systems*, 1983.

LSP. L. Lamport, R. Shostak and M. Pease, The Byzantine generals problem, *ACM Trans. on Programming Languages and Systems* 4 (1982), 382-401.

PSL. M. Pease, R. Shostak and L. Lamport, Reaching agreement in the presence of faults, *J.ACM* 27 (1980), 228-234.

Ra. M. O. Rabin. Randomized Byzantine generals, *these proceedings*.