On the Minimal Synchronism Needed for Distributed Consensus

DANNY DOLEV
Hebrew University, Jerusalem, Israel

CYNTIA DWOR
Cornell University, Ithaca, New York

AND

LARRY STOCKMEYER
IBM Almaden Research Center, San Jose, California

Abstract. Reaching agreement is a primitive of distributed computing. Whereas 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 et al. have shown that in a completely asynchronous model, even one failure cannot be tolerated. In this paper their work is extended: Several critical system parameters, including various synchrony conditions, are identified and how varying these affects the number of faults that can be tolerated is examined. The proofs expose general heuristic principles that explain why consensus is possible in certain models but not possible in others.

Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Systems—distributed applications, distributed databases, network operating systems; C.4 [Performance of Systems]: reliability, availability, and serviceability; F.1.2 [Computation by Abstract Devices]: Modes of Computation—parallelism; H.2.4 [Database Management]: Systems—distributed systems

General Terms: Algorithms, Reliability, Theory, Verification

Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, reliability

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., [1], [7], [8], [11] and [12]). We consider a system of \( N \) processors

A preliminary version of this paper appears in the Proceedings of the 24th Annual Symposium on Foundations of Computer Science, November 7-9, 1983, Tucson, Ariz., pp. 393-402. © 1983 IEEE. Any portions of this paper that appeared in the original version are reprinted with permission. The work of D. Doel was performed in part at Stanford University, supported in part by DARPA under grant MDA903-80-C-0107 and in part at the IBM Research Laboratory, San Jose, Calif. The work of C. Dwork was supported in part by National Science Foundation grant MCS 81-01220.

Authors' present addresses: D. Doel, Computer Science Department, Hebrew University, Jerusalem, Israel; C. Dwork, Department K33/802, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120; L. Stockmeyer, Department K33/802, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0004-5411/87/0100-0077 $00.75

$p_1, \ldots, p_N$ ($N \geq 2$) that 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 (nontrivial) consensus problem if

(i) no matter how the system 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;
(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 synchrony are assumed in much of the research on "Byzantine Agreement," (e.g., [7] and [11]).

Our point of departure and motivation for this paper was the interesting recent result of Fischer et al. [10], 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:

(i) Processor asynchrony allows processors to "go to sleep" for arbitrarily long finite amounts of time while other processors continue to run.
(ii) Communication asynchrony precludes an a priori bound on message delivery time.
(iii) 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 lockstep synchrony, thus strengthening the main result of [10]. In this result we retain Fischer et al.'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 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 in which 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 on the basis of the value of the received message (or on the basis of 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.

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; that is, messages cannot be lost.
   
   F. Synchronous. There is a constant $\Delta \geq 1$ such that every message is delivered within $\Delta$ 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; e.g., 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.
TABLE I. MAXIMUM RESILIENCIES

<table>
<thead>
<tr>
<th>mb</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>0</td>
<td>1</td>
<td>N</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

For each setting of the five system parameters, processors (p), communication (c), message order (m), transmission mechanism (b), and receive/send (s), to either favorable (1) or unfavorable (0), the corresponding table entry gives the maximum resiliency for that case and indicates the theorem(s) from which the resiliency bound follows.

"FLP" refers to the impossibility result of Fischer, Lynch, and Paterson [10].

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 [10].

To obtain the strongest possible results we make some further conventions. Whenever we assume synchronous processors in an impossibility result, we actually take $\Phi = 1$, that is, the processors operate in rounds, which is essentially the same as lockstep synchrony. Whenever we assume synchronous communication in an impossibility result, we actually take $\Delta = 1$, that is, 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 nontrivial 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 that can elapse between the two steps.

Varying these five parameters yields 32 cases, and we have found the maximum resiliency for each case (see Table I). More interestingly, we have identified four cases in which 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.

The delineation of these boundaries is the main contribution of this paper.
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.

Another goal of this paper has been to understand intuitively why Fischer et al. 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 “buffer” an event or the relative ordering among several events, then no consensus protocol is \( t \)-resilient. In the original proof in [10], the event is a “critical step,” where one processor \( p \) takes a step that 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, by the same 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 [3, 4], Rabin [13], Bracha [5], and Toueg [14] have shown that consensus in the presence of faults can be achieved in various asynchronous environments by probabilistic protocols that reach consensus in finite time with probability 1. 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 in which 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. [7, 11]).

2. Definitions

In this section we extend the formal framework of Fischer et al. [10] to handle our various system parameters. A consensus protocol is a system of \( N \) processors \( P = \{ p_1, \ldots, p_N \} \), each with a special initial bit. 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:

1. \( \text{Send}(p, m) \) places message \( m \) in \( p \)'s buffer;
2. \( \text{Receive}(p) \) 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 in which message order is synchronous. Each processor \( p \) is specified by a state transition function \( \delta_p \) and a sending function \( \beta_p \), where

\[
\begin{align*}
\delta_p : Z \times M^* & \rightarrow Z, \\
\beta_p : Z \times M^* & \rightarrow \{ B \subseteq P \times M \mid B \text{ is finite} \}.
\end{align*}
\]

The pair \( (q, m) \) in the range of \( \beta_p \) 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 \( p, q \in P, z \in Z, \mu \in M^* \) 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 \( p, z, \mu \). 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 \( \beta_p(z, \mu) = \emptyset \)), 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, that is, all transitions from states in \( Z_R \) must go to states in \( Z_S \), and vice versa. If receive/send is atomic, then a processor can both receive and send messages from any state in \( Z \).

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;
(ii) \( N \) strings \( \text{buff}(p_i, C) \in M^* \) 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 \( \lambda \) (the empty string).
An event is a pair \((p, \mu)\), where \(p \in P\) and \(\mu \in M^*\). Think of the event \((p, \mu)\) as the receipt of message string \(\mu\) by \(p\). Processor \(p\) is said to be the agent of the event \((p, \mu)\). 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 \(\text{st}(p, C) \in Z_s\), then \((p, \mu)\) is applicable to \(C\) only if \(\mu = \lambda\).

The remaining conditions apply when \(\text{st}(p, C) \in Z_r\) if receive/send is separate, or in general if receive/send is atomic.

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

3. If communication is synchronous, \((p, \mu)\) is applicable to \(C\) only if \(\mu\) is a prefix of \(\text{buff}(p, C)\).

4. If communication is immediate, \((p, \mu)\) is applicable to \(C\) only if \(\mu = \text{buff}(p, C)\).

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 \(z = \text{st}(p, C)\) 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 \(\text{buff}(q, C)\);
(c) delete \(\mu\) from the left end of \(\text{buff}(p, C)\).

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 \(M\), \(\text{buff}(p, C)\) is a finite subset of \(M\), events have the form \((p, \mu)\), where \(\mu\) is a finite subset of \(M\), and the empty set \(\emptyset\) takes the place of \(\lambda\). 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, \((p, \mu)\) is applicable only if \(\mu \subseteq \text{buff}(p, C)\) 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 \(I\) under the following conditions:

1. the events of \(\sigma\) can be applied in turn starting from \(I\), that is, \(\sigma_1\) is applicable to \(I\), \(\sigma_2\) is applicable to \(\sigma_1(I)\), etc.;
2. if processors are \(\Phi\)-synchronous (constant \(\Phi \geq 1\)), then, for every consecutive subsequence \(\tau\) of \(\sigma\), if some processor takes \(\Phi + 1\) steps in \(\tau\) and if the processor \(p\) takes no step in \(\tau\), then \(p\) takes no steps in the portion of \(\sigma\) following \(\tau\) (this says that once a processor fails it cannot restart at a later time);
3. if communication is \(\Delta\)-synchronous (constant \(\Delta \geq 1\)), then, for every \(j\), if \(\sigma_j = (p, \mu)\), if message \(m\) was sent to \(p\) by the event \(\sigma_i\) with \(i \leq j - \Delta\), if the state of \(p\) in configuration \([\sigma_1 \ldots \sigma_{j-1}]I\) is a receiving state, and if none of the events \(\sigma_k\) with \(i < k < j\) is the receipt of \(m\) by \(p\), then \(m\) belongs to \(\mu\) (this says that messages must be delivered within \(\Delta\) real-time steps, after which \(p\) will receive \(m\) at its next Receive step, if it has not already received \(m\) earlier).

If \(\Phi = 1\) in (2), the processors are said to be in lockstep.
If $\sigma$ is finite, $\sigma(I)$ denotes the resulting configuration that is said to be \textit{reachable} from $I$. A configuration reachable from some initial configuration is said to be \textit{accessible}. Henceforth, all configurations mentioned are assumed to be accessible. If $Q$ is a set of processors, the schedule $\sigma$ is \textit{Q-free} if no $p \in Q$ takes a step in $\sigma$. A schedule, together with the associated sequence of configurations, is called a \textit{run}.

We assume that there are two disjoint sets of \textit{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 \textit{decision value} $v$ if st$(p, C) \in Y_v$ for some $p$. A consensus protocol is \textit{partially correct} if

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

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

1. the associated schedule is applicable to $I$;
2. at most $t$ processors are faulty;
3. all messages sent to nonfaulty processors are eventually received.

A run is a \textit{deciding run} if every nonfaulty processor enters a decision state. A protocol is a \textit{$t$-resilient protocol for the nontrivial consensus problem} if it is partially correct and every infinite $t$-admissible run from every initial configuration is a deciding run. A protocol is a \textit{$t$-resilient protocol for the strong consensus problem} if it is partially correct and every infinite $t$-admissible run from every initial configuration is a deciding run; moreover, if $I$, is the initial configuration in which all processors have initial value $v$, then all deciding configurations reachable from $I$, 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 the definition must depend not only on $C$ but also on the history of events that led to $C$. If $C$ is reached from initial $I$ by schedule $\tau$, then $\sigma$ is \textit{applicable} to $C$ iff $\tau \sigma$ is applicable to $I$. (To be completely precise, we should include the history $\tau$ as part of the configuration $C$. However, in all our impossibility proofs, the history is clear from context, so we say simply that $\sigma$ is applicable to $C$ rather than to $(C, \tau)$.)

For configuration $C$ let $V(C)$ be the set of decision values of configurations reachable from $C$. Configuration $C$ is \textit{bivalent} if $V(C) = \{0, 1\}$, or \textit{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.

\textbf{Remark.} To reduce the number of different proofs that must be given, when we prove impossibility of a $t$-resilient nontrivial consensus protocol for some model $M$ (specified by the five system parameters), we would like to conclude immediately the impossibility of a $t$-resilient nontrivial consensus protocol in a weaker model $W$ obtained from $M$ by changing either the transmission parameter or the receive/send parameter (or both) from favorable to unfavorable. This conclusion can be made since it is easy to see that, given any protocol $\mathcal{P}_M$ in the model $M$, there is a protocol $\mathcal{P}_W$ in the model $W$ that simulates $\mathcal{P}_M$. So the existence of a $t$-resilient nontrivial consensus protocol in model $W$ would imply the same in model $M$, but we have proved impossibility for model $M$. If $M$ has broadcast transmission and $W$ has point-to-point transmission, the simulation is trivial because point-to-point
transmission is a special case of broadcast transmission, that is, take \( P_M = P_W \). If \( M \) has atomic receive/send and \( W \) has separate receive/send, each processor \( p_i \) in \( P_W \) is simulated by a processor \( q_i \) in \( P_M \). Simulating a receiving step of \( p_i \) is trivial. If \( q_i \) happens to receive messages when simulating a sending step of \( p_i \), then \( q_i \) remembers these messages until the next time it simulates a receiving step of \( p_i \). The details of how the transition and sending functions of \( q_i \) are obtained from \( p_i \) are left to the reader.

3. The Principal Boundaries

Since our impossibility proofs follow the general outline used by Fischer et al. [10], it is worthwhile to review this outline first. The proof assumes the existence of a \( t \)-resilient protocol and reaches a contradiction. There are three steps.

Step I. Show that there is a bivalent initial configuration.

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

Step III. Using I and II, construct an infinite \( t \)-admissible run that is not deciding as follows. Let \( B_i \) be an initial bivalent configuration. In general, if \( B_i \) is bivalent, let \( p = p_i \) where \( j = i \mod N \) and let \( B_{i+1} = \sigma(B_i) \), where \( \sigma \) is obtained from \( B_i \). Moreover, if \( p \)'s buffer is nonempty in \( B_i \), let \( p \) receive a message that 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 [10], in most cases we lose the "commutativity" property of schedules that was used heavily in [10]. Therefore, we have had to develop new techniques beyond those used in [10]. First we review the first step of the outline.

**Lemma 3.1** [10]. For any choice of the system parameters and any \( t \geq 1 \), if a protocol is \( t \)-resilient and solves the nontrivial 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 \( I_i \) is \( i \)-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 \( J_i \) is \( i \)-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 \( \sigma \) be the associated schedule, \( \sigma \) is applicable to \( J_1 \) also, and the same decision 0 is reached in both cases. This contradicts the 1-valency of \( J_1 \). \( \square \)

Another basic lemma from [10], variations of which are used in most of our proofs, is the following.

**Lemma 3.2** [10]. Suppose that processors and communication are both asynchronous. Let \( C \) be a bivalent configuration and let \( e = (p, m) \) be an event applicable to \( C \). Let \( \mathcal{E} \) be the set of configurations reachable from \( C \) without applying \( e \) and let \( \mathcal{D} = \{ e(E) \mid E \in \mathcal{E} \} \). If \( \mathcal{D} \) contains no bivalent configuration, then \( \mathcal{D} \) contains both a 0-valent and a 1-valent configuration.
PROOF. Assume that \( \mathcal{D} \) contains no bivalent configuration, and let \( v \in \{0, 1\} \). Since \( C \) is bivalent, there is a schedule \( \sigma \) such that \( \sigma(C) \) has decision value \( v \). If \( e \) is an event in \( \sigma \), then writing \( \sigma = \sigma_1 \sigma_2 \), where \( e \) does not occur in \( \sigma_1 \), and letting \( D = e(\sigma_1(C)) \), we have \( D \in \mathcal{D} \) and \( D \) must be \( v \)-valent because \( \sigma(D) \) has decision value \( v \). If \( e \) is not in \( \sigma \), then \( \sigma(C) \in \mathcal{C} \) and \( e \) is applicable to \( \sigma(C) \), so \( e(\sigma(C)) \in \mathcal{D} \) and it must be \( v \)-valent because \( \sigma(C) \) has decision value \( v \). \( \square \)

The main result of [10] 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 nontrivial consensus problem. Our first result strengthens this by allowing synchronous, even lockstep, processors. In general, the letters I and E in the names of our theorems refer to impossibility and existence of protocols, respectively.

**Theorem 10.** In the model with asynchronous communication and asynchronous message order (and the other three parameters favorable), there is no 1-resilient nontrivial consensus protocol. Moreover, this is true if processors are lockstep synchronous.

**Proof.** We introduce the notion of a “failure step” as an expostional convenience. The corresponding event is denoted \((p, \dagger)\); the dagger denotes death. The next configuration \((p, \dagger)(C)\) is identical to \( C \). To the definition of applicable schedule \( \sigma \), add the conditions that if a processor takes a failure step, then all of its subsequent steps are failure steps, and if \( \tau \) is a subsequence of \( \sigma \) in which some processor takes \( \Phi + 1 \) steps, then every processor takes at least one step in \( \tau \) (possibly a failure step). In other words, we force the processors to keep taking steps, but we allow them to fail by taking failure steps from some point on. 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 \( D_i \) has decision value \( v \) for \( v = 0, 1 \).

**Lemma 10.1.** Let \( C \) be a configuration reachable from some initial configuration by a failure-free run. Then \( C \) is ff-bivalent iff \( C \) is bivalent.

**Proof.** Clearly ff-bivalence implies bivalence. 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 \( \sigma \) be the schedule associated with \( R \). If \( \sigma \) contains no failure events, we are done. Say then that \( p \) takes failure steps in \( \sigma \). Consider any schedule \( \sigma' \) identical to \( \sigma \), except that \( p \) takes normal steps of its protocol instead of failure steps. Since communication and message order are asynchronous, \( \sigma' \) is applicable to \( C \) also since any message sent by \( p \) in \( \sigma' \) but not sent by \( p \) in \( \sigma \) 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 \( \sigma' \) to \( C \) leads to decision \( v \). \( \square \)

**Lemma 10.2.** Let \( C \) be a bivalent configuration reachable from some initial configuration by a failure-free run, and let \( p \) be a processor. If \( \text{buff}(p, C) \neq \emptyset \), let \( m \) be an arbitrary message in \( \text{buff}(p, C) \), or let \( m = \emptyset \) if \( \text{buff}(p, C) = \emptyset \). Let \( \mathcal{E} \) be the set of configurations reachable from \( C \) by any failure-free run in which the event \( e = (p, m) \) is not applied, and let

\[ \mathcal{D} = \{ e(E) | E \in \mathcal{E} \text{ and } e \text{ is applicable to } E \}. \]

Then \( \mathcal{D} \) contains a bivalent configuration.
Proof. Suppose for contradiction that $\mathcal{P}$ contains only univalent configurations. Say that processors are $\Phi$-synchronous for some $\Phi \geq 1$. Since the message $m$ is still in $p$'s buffer for every $E \in \mathcal{P}$, $e$ is applicable to $E$ if and only if having $p$ take a step from $E$ does not form a subschedule in which $p$ takes $\Phi + 1$ steps but some other $q$ takes no step. As in the proof of Lemma 3.2, it is easy to show that $\mathcal{P}$ contains both a 0-valent and a 1-valent configuration. In carrying out the details of the proof, first use Lemma 10.1 to conclude that $C$ is $\Phi$-bivalent. Therefore, for each $v \in \{0, 1\}$ there is a failure-free run from $C$ that leads to decision value $v$, so there must be both a 0-valent and a 1-valent configuration in $\mathcal{P}$.

The next step is to argue that there are configurations $C_0, C_1 \in \mathcal{P}$ and a failure-free schedule $\sigma$ such that $C_1 = \rho(C_0)$, $e(C_0)$, and $e(C_1)$ have different univalencies, and $p$ is the agent of no event in $\rho$ except possibly the first. To show this, let

$$\mathcal{P}' = \{E \mid E \in \mathcal{P} \text{ and } e \text{ is applicable to } E \}.$$

If $B \in \mathcal{P}'$ and $e(B)$ is $\nu$-valent, then label $B$ with $\nu$. We have argued above that every member of $\mathcal{P}'$ receives a label of either 0 or 1, and that 0 and 1 both appear as labels of (different) elements of $\mathcal{P}'$. Therefore, there must be a failure-free schedule $\sigma$ and a configuration $C' \in \mathcal{P}'$ such that $C' = \sigma(C)$ and $C$ and $C'$ are labeled differently. On the path of configurations obtained by applying $\sigma$ to $C$, let $C_0$ and $C_1$ be two configurations on this path that are labeled differently, and such that there is no configuration of $\mathcal{P}'$ on the portion of the path strictly between $C_0$ and $C_1$. Let $\rho$ be the portion of the schedule $\sigma$ between $C_0$ and $C_1$ so that $C_1 = \rho(C_0)$. We still must argue that $\rho$ is the agent of no event of $\rho$, except possibly the first. Suppose otherwise that $\rho = \alpha f \beta$, where $\alpha$ is nonempty and $f$ is the agent of $f$. Since $f$ is applicable to $\alpha(C_0)$, it is easy to see that $e$ is applicable to $\alpha(C_0)$; so $\alpha(C_0) \in \mathcal{P}'$. But this contradicts the fact that $C_0$ and $C_1$ were chosen so that no member of $\mathcal{P}'$ lies on the path $\rho$ between them. Say, without loss of generality, that $C_1$ is labeled $v$ for $v = 0, 1$. Let $D_1 = e(C_1)$, so $D_1$ is $\nu$-valent.

There are two cases. If $p$ is the agent of no event in $\rho$, then $\rho$ is applicable to $e(C_0)$ and

$$\rho(D_0) = \rho(e(C_0)) = e(\rho(C_0)) = e(C_1) = D_1,$$

which is a contradiction because $D_1$ is $\nu$-valent. (This is an application of commutativity as in [10].)

If $p$ is the agent of the first event in $\rho$ (but not other), write $\rho = f \tau$, where $f = (p, m')$ and $m \neq m'$, and write

$$\tau = (q_2, m_2), (q_3, m_3), \ldots, (q_k, m_k).$$

Let $\tau' = (q_2, \emptyset), (q_3, \emptyset), \ldots, (q_k, \emptyset)$ and $\rho' = f \tau'$. (If $\tau$ is empty, then $\tau'$ is empty and $\rho = \rho' = f$.) If $e(\rho'(C_0))$ is bivalent, we are done since $e(\rho'(C_0)) \in \mathcal{P}$. If $e(\rho'(C_0))$ is not bivalent, we claim that it is 1-valent. Suppose otherwise that it is 0-valent. For $1 \leq j \leq k$, let

$$\tau_j = (q_2, n_2), \ldots, (q_k, n_k),$$

where $n_i = m_i$ if $i \leq j$ and $n_i = \emptyset$ if $i > j$, and let $\rho_j = f \tau_j$. In particular, $\rho_1 = \rho'$ and $\rho_k = \rho$. 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 $\rho_j$. Since $e(\rho_j(C_0))$ is 0-valent and $e(\rho_k(C_0))$ is 1-valent, there must exist a $j$ such that $e(\rho_j(C_0))$ is 0-valent and $e(\rho_j(C_0))$ is 1-valent. The only difference between $\rho_{j-1}$ and $\rho_j$ is that $\rho_{j-1}$ contains $(q_i, \emptyset)$ where $\rho_j$ contains $(q_i, m_i)$. Let $\eta$ be identical to $\rho_j$, except that the event $(q_i, m_i)$ in $\rho_j$, as well as all events with
agent \( q_i \), following this event, are replaced by \((q_i, \dagger)\) in \( \eta \). Note that \( \eta e \) is applicable to \( C_0 \) also. Let \( \sigma \) be a \( q_i \)-free finite schedule applicable to \( G_2 = e(\eta(C_0)) \) such that the associated run is deciding. Then \( \sigma \) is applicable to \( G_0 = e(\rho_0(C_0)) \) and to \( G_1 = e(\rho_1(C_0)) \). Since \( st(r, G_0) = st(r, G_1) = st(r, G_2) \) for all \( r \neq q_i \), the same decision is reached when \( \sigma \) 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 \( \tau \) to \( \tau' \) is to ensure that no event in \( \tau' \) 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, \([\tau', (p, \dagger)]\) 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 \( \sigma \) be a \( p \)-free schedule applicable to \( E_2 \) such that the associated run is deciding. Since \( \sigma \) 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 \( E_1 \). \( \Box \)

Using Lemmas 3.1 and 10.2, the proof of Theorem 10 is now completed, as described in step III of the general outline. \( \Box \)
Remark. Fischer et al. [10] use a slightly weaker definition of a correct consensus protocol that requires a decision by at least one nonfaulty processor, rather than by all nonfaulty processors as in our definition. In the case of atomic receive/send and broadcast transmission, it is easy to see that the two definitions are equivalent. If we have a protocol where at least one processor must decide, we can turn it into a protocol where every nonfaulty processor decides by the following modification to the transition rules of the processors: At any step when a processor first makes a transition into a decision state in \( Y_v \), it broadcasts “Decide \( v \)” to all others. Any processor receiving a “Decide \( v \)” message immediately decides \( v \). Therefore, Theorem 10 indeed strengthens the impossibility result of [10].

Another set of definitions and a simple lemma are 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 \( \text{buff}(p, C) = \text{buff}(p, D) \) for all \( p \in P - X \).

Definition. Let \( e = (p, \mu) \) be an event applicable to the configuration \( C \). If receive/send is separate, assume that \( st(p, C) \) is a receiving state. Event \( e \) (when applied to \( C \)) is a total reception if \( \mu = \text{buff}(p, C) \).

Lemma 3.3. Assume any model in which processors are asynchronous. 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. Let \( \sigma \) be an \( X \)-free schedule applicable to \( D_0 \) such that the associated run is deciding and every receiving event in \( \sigma \) is a total reception. If no such \( \sigma \) existed, then there would be an infinite nondeciding run with \( t \) faulty processors (those in \( X \)), contradicting \( t \)-resilience. Since all receptions in \( \sigma \) are total and processors are asynchronous, \( \sigma \) is applicable to \( D_1 \) also, and the same decision is made in both cases.

By Theorem 10 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 \( 2\Delta \) of its own steps (where communication is \( \Delta \)-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 \( 2\Delta \) more steps. If at some point it has run for \( 2\Delta \) steps without receiving any messages, it sees whether 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 $\bar{v}$ (the negation of $v$), let $q$ be the processor
that decides $\bar{v}$ earliest. Before deciding, both $p$ and $q$ attempt to receive for $2\Delta$ steps
but receive 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 $\Delta$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 $d_q > d_p > s_q + \Delta$, 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 $\bar{v}$, $q$ has not received a message “Decide $\bar{v}$” from some
other processor. Therefore, $p$ and $q$ must decide on the same value.

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 unexpected
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 (i.e., communication is synchronous, receive/send is atomic, and
the other three parameters are unfavorable), 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 transmission 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 11.1.** Assume $N \geq 3$. If the model of Theorem E1 is weakened by
making transmission point-to-point (i.e., communication is synchronous, receive/
send is atomic, and the other three parameters are unfavorable), then there is no
2-resilient nontrivial consensus protocol. Moreover, this is true even if message
order is synchronous and communication is immediate.

**Proof.** Assume that communication is $\Delta$-synchronous for some $\Delta \geq 1$ and
that message order is synchronous. Assume there is a 2-resilient protocol in this
model.

We let $p$ denote the total reception event with agent $p$ (so the messages received
depend on the configuration to which this event is applied). In particular, $p(C)$
denotes the configuration reached when the event $(p, \text{buff}(p, C))$ is applied to $C$.

**Lemma 11.1.1.** There do not exist a bivalent $C$ and $v$-valent $D_r$ 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
$\text{st}(q, C) = \text{st}(q, D_0)$ and $\text{buff}(q, C) = \text{buff}(q, D_0)$, $q$ 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 $D_r$
are $\{p, r\}$-equivalent, again a contradiction.
Lemma 11.1.2. Let C be a bivalent configuration and let p and q be distinct processors. There is a configuration E reachable from C and an event e whose agent is either p or q such that e(E) is bivalent.

Proof. If either p(C) or q(C) is bivalent, we are done. By Lemma 11.1.1, 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 R 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 an event f such that B = f(A), p(A) and q(A) are both 0-valent, but p(B) and q(B) are not both 0-valent (see Figure 2). 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. If we assume then that p(B) and q(B) are both univalent, Lemma 11.1.1 and the choice of B imply that p(B) and q(B) must both be 1-valent. Let r be the agent of f. There are two cases.

(1) Δ = 1. Since f must be a total reception in this case, note that r ≠ p and r ≠ q, since p(A) and q(A) are both 0-valent but B (= r(A)) is not 0-valent. Let s be the processor to which r sends when it takes the step from A to B. Say without loss of generality that s ≠ p. Now it is easy to see that p(A) and p(B) are |r, s|-equivalent, which contradicts Lemma 3.3.

(2) Δ ≥ 2. If r ∈ {p, q}, the proof is identical to case (1). Say then that r = p. Let s be as above. If s ≠ q, then q(A) and q(B) are |p, s|-equivalent. If s = q, let g be the event (q, buff(q, A)). Since Δ ≥ 2, g is applicable to B (i.e., the message sent from p to q by the event f does not have to be received when q takes a step from B). If g(B) is 0-valent, then B is bivalent and we are done by taking E = A and e = f. If g(B) is 1-valent, then we have another contradiction to Lemma 3.3, since q(A) and g(B) are |p, q|-equivalent.

To complete the proof of Theorem 11.1, 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 11.1.2 to let the other processors take steps, in turn, while staying in bivalent configurations. Note that since communication is synchronous, every message sent to a nonfaulty processor is received at some time in this infinite run. Therefore, 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 11.2. If the model of Theorem E1 is weakened by making receive/send separate (i.e., communication is synchronous, transmission is broadcast, and the other three parameters are unfavorable), then there is no 1-resilient nontrivial consensus protocol. Moreover, this is true if communication is immediate.
Lemma 11.2. Let $C$ be a bivalent configuration and let $p$ be a processor. Let $\mathcal{C}$ be the set of configurations reachable from $C$ without letting $p$ take a step and let

$$\mathcal{D} = \{e(E) \mid E \in \mathcal{C} \text{ and } e \text{ is an event with agent } p \text{ which is applicable to } E\}.$$  

Then $\mathcal{D}$ contains a bivalent configuration.

Proof. Suppose that $\mathcal{D}$ contains only univalent configurations. By a proof very similar to the proof of Lemma 3.2, $\mathcal{D}$ contains both a 0-valent and a 1-valent configuration. Therefore, we can find configurations $C_0, C_1 \in \mathcal{C}$ and events $e_0, e_1, f$ such that $C_1 = f(C_0)$, $p$ is the agent of $e_0$ and $e_1$, and $e_0(C_0)$ and $e_1(C_1)$ have different univalencies. Say, without loss of generality, that $D_r = e_r(C_r)$ is $r$-valent (see Figure 3). Let $q$ be the agent of $f$. By definition of $\mathcal{C}$, $p \neq q$. There are four cases, depending on whether $p$ or $q$ are in receiving or sending states in $C_0$, and in each case we find an equality or equivalence that contradicts Lemma 3.3.

1. $st(p, C_0) \in Z_0$ and $st(q, C_0) \in Z_1$.

Since $p$ is deterministic, $e_0 = e_1$. Let $D_0 = f(D_0)$; $D_0$ is 0-valent. Since message order is asynchronous,

$$D_0 = f(e_0(C_0)) = e_0(f(C_0)) = e_1(C_1) = D_1.$$  

2. $st(p, C_0) \in Z_1$ and $st(q, C_0) \in Z_0$.

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 $f$ is applied to $C_0$ as when $f$ is applied to $D_0$; in particular, the messages sent are the same in the two cases. Also, $st(p, C_1) \in Z_0$. It follows that $D_0$ and $D_1$ are $|p|$-equivalent.

3. $st(p, C_0) \in Z_0$ and $st(q, C_0) \in Z_1$.

By an argument similar to case (2), $D_0$ and $D_1$ are $|p|$-equivalent.

4. $st(p, C_0) \in Z_1$ and $st(q, C_0) \in Z_1$.

Since $e_0 = e_1$ in this case, $D_0$ and $D_1$ are $|q|$-equivalent.

Starting from an initial bivalent configuration and using Lemma 11.2 as in step III of the outline, we construct an infinite nondeciding run in which every processor takes infinitely many steps. Since communication is synchronous, all messages must be received; so this is a 0-admissible nondeciding run. □
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 that reaches Byzantine agreement, for example, [7].

**Theorem E2.** If the model of Theorem E2 is weakened by making processors asynchronous (i.e., communication is synchronous and the other four parameters are unfavorable), then there is no 1-resilient nontrivial 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 11.2. 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') st(p, C0) ∈ Zs and st(q, C0) ∈ Zs.

Let D0 = f(D0). Note again that e0 = e1. If p and q do not send to the same processor in the two events e0 and f, then D0 = D1. If p and q send to the same processor r, then D0 and D1 are |r|-equivalent. □

**Remark (partial synchrony).** Consider cases in which transmission is point-to-point, receive/send is separate, and message order is asynchronous (a practically realistic situation). Theorems 10 and 12 show that either asynchronous processors or asynchronous communication precludes any nonzero resiliency. Dwork et al. [9] define various types of "partially synchronous models" that lie between the completely synchronous and completely asynchronous cases, and show that resiliency proportional to N is achievable in these models. In one version of partial synchrony, Dwork et al. [9] give partially correct consensus protocols (i.e., no two correct processors decide differently no matter how asynchronously the system behaves); for termination of the protocols, the bounds Δ and Φ in the definitions of synchronous communication and synchronous processors do not have to hold throughout the execution of the protocol, but these bounds only have to hold starting at some real time T, which is unknown a priori to the processors. For example, a special case of partially synchronous processors allows different processors to start the protocol asynchronously, that is, at very different (real) times. The exact resiliency that is possible depends on the fault model and on the particular synchrony or partial synchrony assumptions, and the reader is referred to [9] for more details.

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, and q broadcasts a message m', then
Theorem 13. If the model of Theorem E3 is weakened by making transmission point-to-point (i.e., message order is synchronous and the other four parameters are unfavorable), then there is no 1-resilient nontrivial consensus protocol. Moreover, this is true even if receive/send is atomic.

Theorem 13 is obtained as a corollary of a more general result in the next section.

**Theorem E4.** If message order is synchronous and processors are \( \Phi \)-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 \( \mathcal{P} \) be the \((N - 1)\)-resilient consensus protocol for \( N - 1 \) \( \Phi \)-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 \( \mathcal{P} \). When not running \( \mathcal{P} \), \( p_i \) sends the message "\( p_i \) is alive" to \( p_N \). If \( p_i \) decides in the protocol \( \mathcal{P} \), it first sends the decision value to \( p_N \) and then \( p_i \) itself decides. Since \( \mathcal{P} \) requires only that processors be \( \Phi \)-synchronous, it can be seen that \( p_1, \ldots, p_{N-1} \) simulate \( \mathcal{P} \) 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 \( p_1, \ldots, p_{N-1} \) 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 \( \mathcal{P} \) 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 \( p_1, \ldots, p_{N-1} \) all fail before sending a decision to \( p_N \), then \( p_N \) will eventually discover this and decide on its own initial value. \( \square \)

It follows from previous results that any weakening of the model of Theorem E4 cannot tolerate one failure. By the remark at the end of Section 2, these theorems together with the impossibility result of [10] cover all 32 cases of choosing the system parameters. Table I summarizes the maximum resiliency and the relevant theorem(s) for each of these cases.

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 $k$, $1 \leq k \leq N$, $k$-serializable $k$-casting is sufficient for $(k - 1)$-resiliency.

PROOF. Let $S = \{p_1, \ldots, p_k\}$. 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 $p_{k+1}, \ldots, p_N$. 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 $P \setminus S$. □

THEOREM 15. For any $k$, $2 \leq k \leq N$, $(k - 1)$-serializable broadcasting is not sufficient for $(k - 1)$-resiliency.

LEMMA 15.1. 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 $f(e(C))$ 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 $|Q| \leq k - 1$. Any $Q$-free finite deciding run applicable to $D$ is also applicable to $E$, and the same decision is reached in both cases since $\text{st}(r, D) = \text{st}(r, E)$ for all $r$. □

LEMMA 15.2. Let $C$ be bivalent and let $e = (p, m)$ be an event applicable to $C$. Let $\mathcal{E}$ be the set of configurations reachable from $C$ without applying $e$, and let $\mathcal{D} = \{e(E) \mid E \in \mathcal{E}\}$. Then $\mathcal{D}$ contains a bivalent configuration.

PROOF. Say that $\mathcal{D}$ contains only univalent configurations. By Lemma 3.2 and a simple argument as in preceding proofs, there are configurations $C_0$, $C_1 \in \mathcal{E}$ 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 15.1, $p = q$. 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 15.1. Therefore, $R$ has decision value 0.

By a similar argument, there must be configurations $B_i$ and $B_0$ in $R$ and an event $b$ such that $B_0 = b(B_0)$, $[e](B_0)$ is 0-valent, and $[f](B_1)$ is 1-valent. Since the agent of $b$ is not $p$, the schedule $[be]$ is applicable to $f(B_1)$ (see Figure 4). If $[fbe](B_i)$ is bivalent, we are done because this configuration belongs to $\mathcal{D}$. Let $Q$, with $|Q| \leq k - 1$, 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 $[fbe](B_i)$ is also applicable to $[fbe](B_i)$, and the same decision is reached in both
cases. Therefore, \([\mathcal{F}(B_1)\) is 0-valent because \([\mathcal{F}(B_1)\) is 0-valent. Letting \(B_2 = f(B_1), \mathcal{E}(B_2)\) is 1-valent and \(\mathcal{E}(b(B_2))\) is 0-valent, which contradicts Lemma 15.1. \(\square\)

The proof of Theorem 15 is now completed as in step III of the outline. \(\square\)

Since the model with point-to-point transmission and synchronous message order is a special case of the model with \(1\)-serializable broadcasting, Theorem 13 is an immediate corollary of Theorem 15 with \(k = 2\). Also note that the completely asynchronous model of [10] is precisely \(0\)-serializable broadcasting with receive/send atomic, so the impossibility result of [10] also follows from Theorem 15 and the remark after Theorem 10.

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 unforgeable signature to any message) it is known that there is a consensus protocol that is resilient to any number of Byzantine failures [7, 12]; without authentication, \(t\)-resilient consensus is possible if \(t \leq (N-1)/3\) [11, 12]. 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. Attiya et al. [2] have established bounds on the maximum resiliency possible in the model of Theorem E1 under Byzantine failures with authentication.

(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., [6], [7]). 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 nontrivial consensus problem, there is a similar protocol in which every nonfaulty processor decides after at most \(2\Delta\) 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?

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

Acknowledgment. We thank Joe Halpern, Nancy Lynch, Michael Merritt, and Dale Sweeney for helpful discussions and comments.
REFERENCES


Received March 1984; revised January 1986; accepted February 1986.