# Synchronizing Clocks in the Presence of Faults

LESLIE LAMPORT AND P. M. MELLIAR-SMITH

*SRI International, Menlo Park, California*

Abstract. Algorithms are described for maintaining clock synchrony in a distributed multiprocess system where each process has its own clock. These algorithms work in the presence of arbitrary clock or process failures, including "two-faced clocks" that present different values to different processes. Two of the algorithms require that fewer than one-third of the processes be faulty. A third algorithm works if fewer than half the processes are faulty, but requires digital signatures.

Categories and Subject Descriptors: D.4.5 **[Operating Systems]:** Reliability—*fault tolerance*; D.4.7 **[Operating Systems]:** Organization and Design—*real-time systems*

General Terms: Theory

Additional Key Words and Phrases: Byzantine failures, synchronization

## 1. *Introduction*

In a fault-tolerant multiprocess system, it is often necessary for the individual processes to maintain clocks that are synchronized with one another [4, 5, 9]. Since physical clocks do not keep perfect time, but can drift with respect to one another, the clocks must periodically be resynchronized. Such a fault-tolerant system needs a clock synchronization algorithm that works despite faulty behavior by some processes and clocks. This paper describes three such algorithms.

It is easy to construct fault-tolerant synchronization algorithms if one restricts the type of failures that are permitted. However, it is difficult to find algorithms that can handle arbitrary failures—in particular, failures that can result in "two-faced" clocks. As an example, consider a network of three processes. We would like an algorithm in which a fault in one of the processes or in its clock does not prevent the other two processes from synchronizing their clocks. However, suppose that

—Process 1's clock reads 1:00.
—Process 2's clock reads 2:00.
—Process 3's clock is faulty in such a way that when read by Process 1 it gives the value 0:00 and when read by Process 2 it gives the value 3:00.

Processes 1 and 2 are in similar positions; each sees one clock reading an hour earlier and one clock reading an hour later than its own clock. There is no reason

This work was supported in part by NASA under contract number NAS1-15428 and the National Science Foundation under Grant MCS-8104459.

Authors' address: SRI International, Computer Science Laboratory, 333 Ravenswood Ave., Menlo Park, CA 94025.

why Processes 1 and 2 should change their clocks in such a way that would bring their values closer together.

The algorithms described in this paper work in the presence of any kind of fault, including such malicious, two-faced clocks. The first one is called an *interactive convergence algorithm.* In a network of at least $3m + 1$ processes it will handle up to $m$ faults. Its name is derived from the fact that the algorithm causes correctly working clocks to converge, but the closeness with which they can be synchronized depends upon how far apart they are allowed to drift before being resynchronized.

The final two algorithms are called *interactive consistency algorithms*, so named because the nonfaulty processes obtain mutually consistent views of all the clocks. The closeness with which clocks can be synchronized depends only upon the accuracy with which processes can read each other's clocks and how far they can drift during the synchronization procedure. They are derived from two basic interactive consistency algorithms presented in [6]. The first one requires at least $3m + 1$ processes to handle up to $m$ faults. The second algorithm assumes a special method of reading clocks, requiring the use of unforgeable digital signatures, to handle up to $m$ faults with as few as $2m + 1$ processes. The latter algorithm seems to be of little practical value, since Halpern, Simons, and Strong [3] have recently developed a more efficient algorithm based upon the same method of clock reading. However, we feel that the way our algorithm is derived from the Byzantine Generals algorithm is interesting enough to warrant its description.

Strong and Halpern [9] have recently proved that $3m + 1$ processes are required to allow clock synchronization in the presence of $m$ faults if digital signatures are not used. Hence, our first two algorithms use the minimal number of processes.

## 2. *An Informal Discussion*

Before stating and analyzing our algorithms in detail, we give an informal description of how they work. We make no attempt at rigor here; the purpose of this section is to provide the intuition needed to understand the more rigorous exposition of the succeeding sections. The assumptions, conditions, and algorithms stated here are restated more precisely later. The reader who is not interested in all the details may wish to read only this section and skip the rigorous treatment, going directly from the end of this section to the conclusion.

2.1. THE PROBLEM. Implementing reliable clock synchronization presents many problems—from building accurate hardware clocks to designing programming language primitives for reading the clocks. The purpose of this paper is to present algorithms to solve one of these problems: maintaining clock synchronization once the clocks are initially synchronized. As discussed briefly in [5], the type of message passing used by our clock-synchronization algorithms seems to require that the sender and receiver have clocks that are already synchronized. Achieving initial synchrony is a separate problem, whose solution will depend strongly upon the details of how clock reading and interprocess communication are implemented. We do not address that problem, and instead make the following assumption.

*A*0. All clocks are initially synchronized to approximately the same value.

We assume that processes are provided with reasonably accurate clocks—an assumption that we state as

*A*1. A nonfaulty process's clock runs at approximately the correct rate.

Of course, the "correct rate" for a clock is one second of clock time per second of real time. We make no assumptions about faulty processes' clocks.

Assumptions A0 and A1 leave us with the problem of correcting for the slow drifting apart of the clock values caused by slightly differing clock rates. The easiest way to do this is for processes periodically to reset their clocks. A process will have a physical clock that "ticks" continually and a logical clock whose value equals the value of the physical clock plus some offset. It is the logical clocks that are maintained in synchrony by periodically resetting them, a logical clock being reset by simply changing its offset.

Instead of discontinuously changing clock values in this way, it might be desirable to change the clocks gradually. This can be done by making the logical clock value a more complicated function of the physical clock value, effectively spreading the change over a finite interval. Given an algorithm for discontinuously resynchronizing the clocks and a bound on how closely synchronized it keeps them, it is easy to devise an algorithm that spreads out the change and to deduce how well it keeps the clocks synchronized. We therefore consider only resynchronization algorithms that periodically increment the clocks.

Having accurate clocks does no good unless those clocks can be read. Synchronizing the clocks requires that each process be able to read not just its own clock, but other processes' clocks as well. Clock values are different from most other values in a computer system because they are continually changing. This poses a problem for a clock-synchronization algorithm unless the algorithm is so fast that clocks do not change significantly during the resynchronization period. The solution to this problem requires that a process read not another process's clock, but rather the difference between that clock and its own. We therefore make the following assumption:

*A2*. A nonfaulty process $p$ can read the difference $\Delta_{qp}$ between another nonfaulty process $q$'s clock and its own with at most a small error $\epsilon$.

Exactly how A2 is satisfied is of no concern for our first two algorithms. However, our third algorithm is based upon a special method of reading clocks, which will be described below. Assumption A2 asserts that a process can read every other process's clock. In the conclusion, we mention how our algorithms can be extended to work when a process can read the clocks of only some other processes.

Before describing any clock-synchronization algorithms, we should specify what conditions such an algorithm should satisfy. The first and most obvious requirement is

*S1*. At any time, the values of all the nonfaulty processes' clocks must be approximately equal.

While it is obviously necessary, S1 is not a sufficient condition. For example, it is satisfied if all clocks are simply set to zero and stopped. The additional requirement we need must intuitively say that the logical clocks keep a reasonable approximation to real time.

By assuming that clocks are periodically resynchronized, and that a process's logical clock runs at the same rate as its physical clock except for this periodic resynchronization, we have ruled out such trivial "solutions" as stopping the clocks. However, we still have the possibility that each resynchronization causes the clocks to jump arbitrarily far. Our second condition places a bound on the amount that a clock can be incremented.

*S2.* There is a small bound $\Sigma$ on the amount by which a nonfaulty process's clock is changed during each resynchronization.

Condition S2 has two important consequences:

—If $\Sigma$ is much smaller than the resynchronization period, then resynchronization introduces a small error in the average running rate of the clocks. This implies that the processes' clocks maintain a good approximation to absolute real time.

—Resynchronization can cause a process to change its clock's value by some amount $A$. If $A > 0$, then $A$ seconds of clock time have disappeared. Anything that the process should have done during those vanished $A$ seconds cannot be done at the proper time. If $A < 0$, then $A$ seconds of clock time occur twice, which could also cause problems. The easy way out of this difficulty is to let each synchronization interval begin (or end) with an interval of length $\Sigma$ during which nothing is scheduled to happen, so the process is idle for $\Sigma$ seconds during each resynchronization period. This is an acceptable solution if $\Sigma$ is small. (If it is not acceptable, then incrementation by $A$ can be spread across a finite interval of time, as mentioned above.)

2.2. THE INTERACTIVE CONVERGENCE ALGORITHM. Our first solution is the interactive convergence algorithm CNV. It relies heavily upon the assumption that the clocks are initially synchronized, and that they are resynchronized often enough so two nonfaulty processes' clocks never differ by more than $\delta$. How closely clocks can be synchronized depends upon how far apart they are allowed to drift before being resynchronized. At least $3m + 1$ processes are needed to handle up to $m$ faults. The algorithm works essentially as follows.

ALGORITHM CNV. *Each process reads the value of every process's clock and sets its own clock to the average of these values—except that if it reads a clock value differing from its own by more than $\delta$, then it replaces that value by its own clock's value when forming the average.*

To see why this works, let us consider by how much two nonfaulty processes' clocks can differ after they are resynchronized. For simplicity, we ignore the error in reading another process's clock and assume that all processes execute the algorithm instantaneously at exactly the same time.

Let $p$ and $q$ be nonfaulty processes, let $r$ be any process, and let $c_{pr}$ and $c_{qr}$ be the values used by $p$ and $q$, respectively, as process $r$'s clock value when forming the average. If $r$ is nonfaulty, then $c_{pr}$ and $c_{qr}$ will be equal. If $r$ is faulty, then $c_{pr}$ and $c_{qr}$ will differ by at most $3\delta$, since $c_{pr}$ lies within $\delta$ of $p$'s clock value, $c_{qr}$ lies within $\delta$ of $q$'s clock value, and the clock values of $p$ and $q$ lie within $\delta$ of one another.

Let $n$ be the total number of processes and $m$ the number of faulty ones, and assume that $n > 3m$. Processes $p$ and $q$ set their clocks to the average of the $n$ values $c_{qr}$ and $c_{pr}$, respectively. We have $c_{qr} = c_{pr}$ for the $n - m$ nonfaulty processes $r$, and $|c_{qr} - c_{pr}| \leq 3\delta$ for the $m$ faulty processes $r$. It follows from this that the averages computed by $p$ and $q$ differ by at most $(3m/n)\delta$. The assumption $n > 3m$ implies $(3m/n)\delta < \delta$, so the algorithm succeeds in bringing the clocks closer together. Therefore, we can keep the nonfaulty processes' clocks synchronized to within $\delta$ of one other by resynchronizing often enough so that clocks which are initially within $(3m/n)\delta$ seconds of each other never drift further than $\delta$ seconds apart.

It appears that by repeated resynchronizations, each one bringing the clocks closer by a factor of $3m/n$, this algorithm can achieve any desired degree of

synchronization. However, we have ignored two factors:

(1) The time taken to execute the algorithm.
(2) The error in reading another process's clock.

The fact that a process $p$ does not read all other clocks at exactly the same time means that it must average not clock values, but the differences $\Delta_{qp}$ defined in A2. It then increments its clock by the average of the values $\Delta_{qp}$, except with values that are too large replaced by zero.

The clock-reading error $\epsilon$ in Assumption A2 means that if $\delta$ is the maximum true difference between the two clocks, then the difference read by process $p$ could be as great as $\delta + \epsilon$. Therefore, a value of $\Delta_{qp}$ read by process $p$ is regarded as too large, and replaced by zero, if it is greater than $\delta + \epsilon$.

2.3. THE INTERACTIVE CONSISTENCY ALGORITHMS. In the interactive convergence algorithm, a process sets its clock to the average of all clock values. Since a single bad value can skew an average, bad clock values must be thrown away. Another approach is to take a median instead of an average, since a median provides a good value so long as only a minority of values are bad. However, because of the possibility of two-faced clocks, the processes cannot simply read each other's clocks and take a median; they must use a more sophisticated method of obtaining the values of other processes' clocks. We now investigate what properties such a method must have.

The median computed by two different processes will be approximately the same if the sets of clock values they obtain are approximately the same. Therefore, the Clock Synchronization Condition S1 will hold (for some suitably small $\delta$) if the following condition holds for every process $r$.

*CC*1. Any two nonfaulty processes obtain approximately the same value for $r$'s clock—even if $r$ is faulty.

While CC1 guarantees that all processes will compute approximately the same clock values, it does not ensure that the values they compute will be reasonable. For example, CC1 is satisfied if every process always obtains the value zero for any process's clock—a procedure yielding an algorithm that violates the Clock Synchronization Condition S2. To ensure that S2 is satisfied, we make the following additional requirement.

*CC*2. If $r$ is nonfaulty, then every nonfaulty process obtains approximately the correct value of $r$'s clock.

If a majority of processes are nonfaulty, then this implies that the median clock value computed by any process is approximately equal to the value of a good clock.[1] Since good clocks do not drift apart very fast, resetting a clock to the value of another good clock ensures that Clock Condition S2 is satisfied for a small value of $\Sigma$.

Conditions CC1 and CC2 are very similar to the requirements for a solution to the interactive consistency or "Byzantine Generals" problem [6, 7]. In this problem, some process $r$ must send a value to all processes in such a way that the following

[1] More precisely, it is either approximately equal to a good clock's value or else lies between the values of two good clocks.

two conditions are satisfied:

*IC*1. All nonfaulty processes obtain the same value.

*IC*2. If process $r$ is nonfaulty, then all processes obtain the value that it sends.

Our two interactive consistency algorithms are modifications of two Byzantine Generals solutions from [6] to achieve conditions CC1 and CC2. The reasons for using these Byzantine Generals solutions, and the possibility of using other solutions, is discussed in the conclusion.

2.3.1. *Algorithm COM.* Our first interactive consistency algorithm, denoted COM($m$), works in the presence of up to $m$ faulty processes when the total number $n$ of processes is greater than $3m$. It is based upon Algorithm OM($m$) of [6].

We first consider the case $n = 4$, $m = 1$, and describe a special case of Algorithm OM(1) in which the value being sent is a number. In this algorithm, process $r$ sends its value to every other process, which in turn relays the value to the two remaining processes. Process $r$ uses its own value. Every other process $i$ has received three "copies" of this value: one directly from process $r$ and the other two from the other two processes.[2] The value obtained by process $i$ is defined to be the median of these three copies.

To show that this works, we consider separately what happens when process $r$ is faulty and when it is nonfaulty. First, suppose $r$ is nonfaulty. In this case, at least two of the copies received by any other process $p$ must equal the value sent by $r$—the one received directly from $r$ and the one relayed by another nonfaulty process. (Since there is at most one faulty process, at least one of the two processes that relay the value to $p$ must be nonfaulty.) The median of a set of three numbers, two of which equal $v$, is $v$, so condition IC1 is satisfied. When process $r$ is nonfaulty, IC1 implies IC2, which finishes the proof for this case.

Next, suppose that process $r$ is faulty. Condition IC1 is then vacuous, so we need only verify IC2. Since there is at most one faulty process, the three processes other than $r$ must be nonfaulty. Each one therefore correctly transmits the value it receives from $r$ to the other processes. All of the other processes thus receive the same set of copies, so they choose the same median, showing that the IC2 is satisfied.

To modify Algorithm OM(1) for clock synchronization, let us suppose that instead of sending a number, a process can send a copy of a clock. (Imagine clocks being sent from process to process, continuing to tick while in transit.) Let us further suppose that sending a clock from one nonfaulty process to another can perturb its value by at most $\epsilon$, but leaves it otherwise unaffected. However, a faulty process can arbitrarily change a clock's value before sending it.

In Algorithm COM(1), we apply Algorithm OM(1) four times, once for each process $r$. However, instead of sending values, the processes send clocks. Exactly the same argument used above to prove IC1 and IC2 proves CC1 and CC2, where "approximately" means to within $O(\epsilon)$.

The more general Byzantine Generals solution OM($m$), which handles $m$ faulty processes, $n > 3m$, involves more rounds of message passing and additional median taking. This algorithm can be found in [6]. Algorithm COM($m$) is obtained from OM($m$) in the same way we obtained COM(1) and OM(1)—namely, by sending clocks instead of messages.

[2] In case a process fails to receive a message, presumably because the sender is faulty, it can pretend to have received any arbitrary message from that process. See [6] for more details.

This completes our description of Algorithm COM($m$), except for one question: how do processes send clocks to one another? The answer is that the processes don't send clocks, they send the clock differences. Process $p$ sends a "copy" of $q$'s clock to another process $r$ by sending a message with the value $\Delta_{qp}$—a message that means "$q$'s clock differs from mine by $\Delta_{qp}$."

Now, suppose $r$ receives a copy of $q$'s clock from $p$ in the form of a message (from $p$) saying "$q$'s clock differs from mine by $x$." How does $r$ relay a copy of this clock to another process? Process $r$ reasons as follows:

—$p$ tells me that $q$'s clock differs from his by $x$.
—I know that $p$'s clock differs from mine by $\Delta_{qr}$.
—Therefore, $p$ has told me that $q$'s clock differs from mine by $x + \Delta_{pr}$.

In other words, then $r$ relays a clock difference sent to him by $p$, he just adds $\Delta_{pr}$ to that difference.

2.3.2. *Algorithm CSM.* It is shown in [7] that, with no assumptions about the behavior of failed processes, the Byzantine Generals problem is solvable only if $n > 3m$. However, we can do better than this by allowing the use of digital signatures. We assume that a process can generate a message that can be copied but cannot be undetectably altered. Thus, if $r$ generates a signed message, and copies of that message are relayed from process to process, then the ultimate recipient can tell if the copy he receives is identical to the original signed message generated by $r$. With digital signatures, we are assuming that a faulty process cannot affix the signature of another process to any message not actually signed by that process. See [6] for a brief discussion of how digital signatures can be generated in practice.

Algorithm SM($m$) of [6] solves the Byzantine Generals problem in the presence of up to $m$ faults for any value of $n$. (The problem is vacuous if there are more than $n - 2$ faults.) We first consider the case $n = 3$, $m = 1$. In Algorithm SM(1), process $r$ sends a signed message containing its value to the other two processes, each of which relays a copy of this signed message to the other. Each process $p$ other than $r$ winds up with a pile containing up to two properly-signed messages: one received directly from process $r$ and another relayed by the third process. Process $p$ may receive fewer than two messages because a faulty process could fail to send a message. The value process $p$ obtains is defined to be the largest of the values contained in this pile of properly signed messages. (If no message is received, then some arbitrary fixed value is chosen.)

For notational convenience, we pretend that $r$ sends a signed message to itself, which it does not relay. It is easy to see that the piles of messages received by the three processes satisfy the following two properties.

*SM*1. For any two nonfaulty processes $p$ and $q$, every value in $p$'s pile is also in $q$'s.

*SM*2. If process $r$ is nonfaulty, then every process's pile has at least one properly signed message, and every properly signed message has the same value.

Note that SM1 holds for $p$ or $q$ equal to $r$ because of our assumption that $r$ sends a properly signed message to itself. Condition IC1 follows immediately from property SM1, and condition IC2 follows immediately from property SM2, proving that SM(1) is a Byzantine Generals solution.

In the general Algorithm SM($m$), messages are copied and relayed up to $m$ times, with each relaying process adding its signature. When a process $p$ receives a message with fewer than $m$ signatures, $p$ signs the message, copies it, and relays it to every

process that has not already signed the message. The reader can either verify for himself or find the proof in [6] that the stacks of messages received by the processes satisfy conditions SM1 and SM2. (Again, we assume that $r$ sends a signed message to itself, so SM1 is satisfied when $p$ or $q$ equals $r$.) Hence, defining the value obtained by a process to be the largest value in its pile gives an algorithm that solves the Byzantine Generals problem.

To turn the Byzantine Generals solution SM($m$) into the clock-synchronization Algorithm CSM($m$), we again send clocks instead of messages. Moreover, we allow processes to sign the clocks that they send. As before, we assume that a clock's value is perturbed by at most $\epsilon$ when sent by a nonfaulty process. However, instead of allowing a faulty process to set a clock to any value when relaying it, we assume that the process can turn the clock back but not ahead. More precisely, we assume that, when relaying a clock, a faulty process can set it back arbitrarily far, but can set it ahead by at most $\epsilon$.

We now use the same relaying procedure as in Algorithm SM($m$) to send copies of $r$'s clock to all processes. For this intuitive discussion, we assume that all clocks run at exactly the same rate, except for the perturbations they receive when being relayed. Each process keeps a copy of every properly signed clock, so after all the relaying has ended, it has a pile of copies of $r$'s clock. (We assume that $r$ keeps a signed copy of its own clock.) Since a nonfaulty process perturbs a clock's value by at most $\epsilon$ when relaying it, the same reasoning used to prove SM1 and SM2 shows that the following properties are true of these piles of copies of $r$'s clock.

*CSM*1. For any two nonfaulty processes $p$ and $q$, if $p$ has a properly signed clock with value $c$, then $q$ has a properly signed clock whose value is within $m\epsilon$ of $c$.

*CSM*2. If process $r$ is nonfaulty and its clock has the value $c$, then every other process has at least one properly signed clock whose value is within $\epsilon$ of $c$, and every properly signed clock that it has reads no later than $c + m\epsilon$.

The value that a process obtains for $r$'s clock is defined to be the fastest clock in its pile. Conditions CC1 and CC2 then follow immediately from CSM1 and CSM2, where "approximately" means to within $O(m\epsilon)$. Hence, this provides a fault-tolerant clock-synchronization algorithm.

To finish the description of Algorithm CSM($m$), we must describe how clocks can be signed and relayed in such a way that they are disturbed by at most $\epsilon$ when relayed by a nonfaulty clock and can be set forward at most $\epsilon$ by a faulty one. As in Algorithm SM($m$), we require a method for generating unforgeable signed messages.

Let us first assume that processes and transmission lines are infinitely fast, so a message can be relayed from process to process in zero time. We use this assumption to construct a method of relaying clocks for which $\epsilon$ equals zero. The message that $r$ sends, and that all the processes relay, is $r$'s clock value $c_r$. The message $c_r$ acts like a clock whose value is now $c_r$. A nonfaulty process relays this value in zero time, so the clock is sent with no perturbation. A faulty process cannot change the value of the clock, since the value is contained in a signed message; all it can do is delay sending the value. This is equivalent to stopping the clock while holding it, which is tantamount to turning the clock back. Hence, the assumption about sending clocks is satisfied, with zero perturbation.

In practice, processes and transmission lines are not infinitely fast. Instead, we assume that the delay in processing and transmitting a message can be determined to within some small $\epsilon$. If we include the time needed to generate a message as part

of the transmission delay, this can be expressed as:

*A2′*. (a) A message from a nonfaulty process is received at its destination $\gamma \pm \epsilon$ seconds after it is sent, for some constant $\gamma$.
(b) A message from a faulty process is received at its destination at least $\gamma - \epsilon$ seconds after it is sent.

Assumption A2′ permits the implementation of a clock-reading scheme satisfying A2: process $p$ reads process $q$'s clock by having it send a message with the current time. To compute $\Delta_{qp}$, process $p$ adds $\gamma$ to the value in the message and subtracts its own clock value. However, A2′ provides more than a way of implementing A2; combined with digital signatures, it allows a clock value to be relayed from process to process, each process in the chain adding its signature to the message. By counting the number of signatures in the message, a process knows how many times the message has been relayed, so it can correct the clock value in the message by adding the appropriate multiple of $\gamma$. The net effect is to introduce an error of at most $\epsilon$ each time the message is relayed by a nonfaulty process, and to allow a faulty process to set the clock ahead by at most $\epsilon$, as required. Of course, this uses the additional assumption:

*A3*. A process can generate an unforgeable digital signature for any message.

## 3. *The Problem*

We now begin our formal exposition. This section gives the precise statement of the Assumptions A0–A2 and the correctness conditions.

3.1. CLOCKS. Any discussion of clocks involves two kinds of time:

*Real Time.* An assumed Newtonian time frame that is not directly observable.
*Clock Time.* The time that is observed on some clock.

We adopt the convention of using lowercase letters to denote quantities that represent real time and uppercase letters to denote quantities that represent clock time. Thus, we will let the "second" denote the unit of real time and the "SECOND" denote the unit of clock time. Within this convention, we use Roman letters to denote large values and Greek letters to denote small values. In most applications, "large" times may be on the order of milliseconds or more and "small" times on the order of microseconds.

It is customary to define a clock to be a mapping $C$ from real time to clock time, where $C(t) = T$ means that at real time $t$ the clock reads $T$. This is appropriate when clocks are used to measure the time at which some event occurred—for example, when a runner crossed the finish line. Thus, if $t$ is the real time at which the runner finished, then $|C(t) - C'(t)|$ represents the difference in the finishing times recorded by two such clocks $C$ and $C'$.

In the process-control systems for which our clock-synchronization algorithms were devised, systems such as the SIFT avionics computer [10], clocks are used to determine when events are generated—for example, when a valve should be shut. In this case, it is more appropriate to define a clock to be the inverse of the usual function, so it is a mapping $c$ from clock time to real time, with $c(T)$ denoting the real time at which the clock $c$ has the value $T$. Thus, if $T$ is the clock time at which the valve is to be shut, then $|c(T) - c'(T)|$ represents the difference in the real times at which two processors with clocks $c$ and $c'$ issue the command to shut it. We thus consider this kind of clock.

Two clocks $c$ and $c'$ are said to be synchronized to within $\delta$ at a clock time $T$ if $|c(T) - c'(T)| < \delta$, so they reach the value $T$ within $\delta$ seconds of one another. This is the way that we will measure clock synchronization, since it is the appropriate one for the class of applications that immediately concern us. If two processes' clocks are synchronized to within $\delta$ at time $T$, then actions generated by the two processes at that time occur within $\delta$ seconds of one another.

If the clocks are used to measure when events occur, rather than to generate events, then one is concerned with the difference between the inverse clocks—the mappings from real time to clock time. Below, we formally state a result whose intuitive meaning is that if $c$ and $c'$ are good clocks that are synchronized to within $\delta$ seconds on some interval, then their inverse clocks are synchronized to within approximately $\delta$ SECONDS on the inverse interval. Hence, our synchronization algorithms can be used in this situation too.

It is most convenient to pretend that clocks run continuously, so a clock $c$ is a continuous function on some interval. (Otherwise, $c$ would be defined only for a discrete set of clock values.) Of course, real processor clocks advance in discrete steps. We can model the discreteness of a real clock as an error in reading the clock. Thus, discreteness adds a clock-reading error at most equal to the interval between "ticks."

Although we have been calling our clocks "functions," only a monotonically increasing clock can be represented by a single-valued function. This is not a problem, since nonfaulty clocks are assumed to be monotonic. However, some care must be taken when formalizing our concepts. We assume that the inverse of a clock $c$ is a single-valued function, so for any real time $t$ there is a unique $T$ such that $c^{-1}(t) = T$.

*Definition* 1. A clock $c$ is a *good clock* during the real-time interval $[t_1, t_2]$ if it is a monotonic, differentiable function on $[T_1, T_2]$, where $T_i = c^{-1}(t_i)$, $i = 1, 2$, and for all $T$ in $[T_1, T_2]$:

$$\left| \frac{dc}{dT}(T) - 1 \right| < \frac{\rho}{2}.$$

This definition involves an arbitrary fixed value $\rho$, which represents twice the maximum error in a clock's running rate. (A perfect clock has $dc/dT$ equal to one second per SECOND.) Rather than fixing $\rho$, we could define a $\rho$-good clock. However, this would require that all our results include an explicit mention of the parameter $\rho$, which would needlessly complicate their statements. We will introduce several similar quantities; they are all listed in a glossary at the end of the paper, which contains a brief reminder of what they mean.

In terms of Definition 1, we can state the precise relation between the synchronization of clocks and of inverse clocks as the following remark. Its proof is an exercise in elementary calculus.

*Remark.* Let $c$ and $c'$ be good clocks on an interval $[T_1, T_2]$, such that $|c(T) - c'(T)| < \delta$ for all times $T$ in that interval. If $\rho \ll 1$, then the inverse functions $C$ and $C'$ of these clocks exist and are differentiable on the interval $[c(T_1) + \delta, c(T_2) - \delta]$, and $|C(t) - C'(t)| \lesssim \delta$ for all $t$ in that interval.

We consider a network of $n$ processes, where each process $p$ contains a clock $c_p$. We assume that the clocks are initially synchronized to within $\delta_0$ of one another at

the "starting time" $T^{(0)}$, so we have

$A0$. For all processes $p$ and $q$: $|c_p(T^{(0)}) - c_q(T^{(0)})| < \delta_0$.

Our algorithms have the property that if enough processes are nonfaulty, then the clocks of nonfaulty processes remain synchronized. To give a formal proof of such a property, we would need a formal definition of a nonfaulty process. More precisely, we would have to define what it means for a process to be nonfaulty during a certain interval. There are two assumptions that must be made about a nonfaulty process: that it correctly executes the algorithm and that its clock is good. A rigorous statement of the first assumption would be tedious and unrewarding, so we do not make a formal definition of "nonfaulty during an interval." Instead, we informally assume that a nonfaulty process does what it is supposed to during the interval in question. However, we do rigorously analyze the degree of synchronization achieved by our algorithms, and this requires us to state the second assumption precisely. This is done as follows.

$A1$. If process $p$ is nonfaulty during the real time interval $[t_1, t_2]$, then $c_p$ is a good clock during that interval.

3.2. SYNCHRONIZATION. As mentioned in our informal description of the algorithms, clock synchrony is maintained by having processes periodically increment their clocks. Incrementing a clock $c$ by $A$ SECONDS means adding $A$ to the value read from the clock. This is described formally as defining a new clock $c'$ by

$$c'(T) = c(T - A).$$

For simplicity, we assume that clocks are resynchronized every $R$ SECONDS. Let $T^{(i)} = T^{(0)} + iR$, and let $R^{(i)}$ be the interval $[T^{(i)}, T^{(i+1)}]$. The clock $c_p$ represents process $p$'s physical clock. Resynchronizing the clocks every $R$ SECONDS means having process $p$ use a logical clock $c_p^{(i)}$ on the time interval $R^{(i)}$, where

$$c_p^{(i)}(T) = c_p(T + C_p^{(i)}) \tag{1}$$

for some constant $C_p^{(i)}$. For convenience, we assume that $C_p^{(0)} = 0$, so $c_p^{(0)} = c_p$.

No algorithm can maintain clock synchronization in the presence of too many faulty processes, so the condition to be satisfied by our algorithms must contain the hypothesis that there are enough nonfaulty ones. To help state this hypothesis, we introduce the following terminology.

*Definition* 2. A process $p$ is said to be nonfaulty up to time $T^{(i+1)}$ if it is nonfaulty during the real-time interval $[c_p^{(0)}(T^{(0)}), c_p^{(i)}(T^{(i+1)})]$.

Note that this interval runs from the time process $p$ is started until the time its clock reaches the end of the $i$th synchronization interval $R^{(i)}$.

Our informal requirements for clock synchronization were that processes' clocks are synchronized to within a small bound, which we shall call $\delta$, and that resynchronization increments a clock by at most $\Sigma$. These conditions depend upon the parameters $\delta$ and $\Sigma$, so we should talk about a "$\delta$–$\Sigma$ synchronization algorithm." For notational simplicity, we leave implicit the dependence on $\delta$ and $\Sigma$, as well as the dependence on $m$, the number of faulty processes tolerated. The following condition defines correct synchronization on the interval $R^{(i)}$, again leaving $i$ an implicit parameter.

*Clock Synchronization Condition.* For all $p$ and $q$, if all but at most $m$ processes are nonfaulty up to time $T^{(i+1)}$, then

*S*1. If Processes $p$ and $q$ are nonfaulty up to time $T^{(i+1)}$, then for all $T$ in $R^{(i)}$

$$|c_p^{(i)}(T) - c_q^{(i)}(T)| < \delta.$$

*S*2. If Process $p$ is nonfaulty up to time $T^{(i+1)}$, then

$$|C_p^{(i+1)} - C_p^{(i)}| < \Sigma.$$

Our problem is to find an algorithm for choosing the values $C_p^{(i+1)}$ such that if the Clock Synchronization Condition holds for $i$, then it will hold for $i + 1$.

We place no restriction on the clock of a process that has failed. Thus, we are not considering the problem of restarting a failed process and bringing it into synchrony with the other processes. This is a nontrivial problem whose solution depends upon the details of how a process reads other processes' clocks, and is beyond the scope of this paper.

3.3. READING CLOCKS. We now formalize our Assumption A2 about clock reading. All the reading of clocks and transmitting of information in the computation of $C_p^{(i+1)}$ is assumed to take place in the final $S$ seconds of the interval $R^{(i)}$—that is, during the interval $S^{(i)} \equiv [T^{(i+1)} - S, T^{(i+1)}]$. Assumption A2, required by Algorithms CNV and OM, is then stated as follows.

*A*2. If the Clock Synchronization Condition holds for $i$, and process $p$ is nonfaulty up to time $T^{(i+1)}$, then for each other process $q$: $p$ obtains a value $\Delta_{qp}$ during the interval $S^{(i)}$. If $q$ is also nonfaulty up to time $T^{(i+1)}$, then

$$|c_p^{(i)}(T_0 + \Delta_{qp}) - c_q^{(i)}(T_0)| < \epsilon \tag{2}$$

for some time $T_0$ in $S^{(i)}$.

For $p = q$, we take $\Delta_{pq} = 0$, so 2 holds in this case too. Remember that $\epsilon$ is an implicit parameter of A2.

The actual method by which $p$ reads $q$'s clock might involve cooperative action by both processes. In this case, determining $\Delta_{qp}$ may require the synchrony of the two processes' clocks, which is why we assume in A2 that the Clock Synchronization Condition holds for $i$.

3.4. APPROXIMATIONS. For real clocks, the maximum rate $\rho$ by which they may drift apart can be reduced to the order of $10^{-6}$ or less. We will simplify our calculations by making approximations based upon the assumption that $n\rho \ll 1$, where $n$ is the number of processes. This means that we will neglect quantities of order $n\rho\epsilon$ and $n\rho^2$ in our calculations. The reader will be able to check the validity of these calculations by showing that for an approximate inequality of the form $x \lesssim y$, the neglected terms are at most of order $n\rho y$. (The inequality $x \lesssim y$ means $x < y'$ for some $y' \approx y$.) We also assume $R \gg \epsilon$, which is the only case of practical interest.

## 4. *The Interactive Convergence Algorithm*

Recall that the interactive convergence algorithm CNV is based upon the following observation: the Clock Synchronization Condition for $i$ implies that if $p$ and $q$ are nonfaulty, then the true difference in their clocks is less than $\delta$, and the observed difference is less than $\delta + \epsilon$. Process $p$ increments its clock by the average of all the $\Delta_{qp}$, with values greater than $\delta + \epsilon$ set to zero. This is expressed precisely as follows, where we assume that the processes are numbered from 1 through $n$.

ALGORITHM CNV. *For all p*:

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p$$

$$\text{where } \Delta_p \equiv \left(\frac{1}{n}\right) \sum_{r=1}^{n} \overline{\Delta}_{rp}$$

$$\overline{\Delta}_{rp} \equiv \textbf{if } r \neq p \textbf{ and } |\Delta_{rp}| < \Delta \textbf{ then } \Delta_{rp} \textbf{ else } 0$$

$$\Delta \approx \delta + \epsilon$$

Note that Algorithm CNV depends explicitly upon $\delta$ and $\epsilon$. We now prove its correctness.

LEMMA 1. *If Clock Synchronization Condition S1 holds for i, and processes p and q are nonfaulty up to time* $T^{(i+1)}$, *then*

$$|\Delta_{qp}| \lesssim \delta + \epsilon.$$

PROOF. Let $T_0$ and $\Delta_{qp}$ be as in A2. Writing

$$c_p^{(i)}(T_0) - c_p^{(i)}(T_0 + \Delta_{qp}) = c_p^{(i)}(T_0) - c_q^{(i)}(T_0) + c_q^{(i)}(T_0) - c_p^{(i)}(T_0 + \Delta_{qp}),$$

it follows easily from S1 and A2 that

$$|c_p^{(i)}(T_0) - c_p^{(i)}(T_0 + \Delta_{qp})| < \delta + \epsilon.$$

The desired result then follows from A1 and the assumption that $\rho \ll 1$. □

THEOREM 1. *If*

$-3m < n$

$-\delta \gtrsim max(n'(2\epsilon + \rho(R + 2S')), \delta_0 + \rho R)$,
*where* $n' \equiv n/(n - 3m)$
$S' \equiv (n - m)S/n$

$-\delta \ll min(R, \epsilon/\rho)$

*then Algorithm CNV satisfies the Clock Synchronization Condition with* $\Sigma = \Delta$.

PROOF. Condition S2 is easy, since $\Delta_p$ is the average of $n$ terms, each less than $\Delta$. We prove Condition S1 by induction on $i$. For $i = 0$, A1 implies that two nonfaulty clocks that are synchronized to within $\delta_0$ up to time $T^{(0)}$ will remain synchronized to within $\delta_0 + \rho R$ at time $T^{(1)} = T^{(0)} + R$. Condition S1 then follows immediately from A0 and the second hypothesis.

We therefore assume that S1 holds for $i$ and prove it for $i + 1$. We begin with the following lemmas.

LEMMA 2. *If Clock Synchronization Condition S1 holds for i and process p is nonfaulty up to time* $T^{(i+2)}$, *then for any* $\Pi$ *such that* $|\Pi| < R$ *and any T in* $S^{(i)}$:

$$|c_p^{(i)}(T + \Pi) - [c_p^{(i)}(T) + \Pi]| < \left(\frac{\rho}{2}\right) \Pi.$$

*Hence, if* $\rho\Pi$ *is negligible, then*

$$c_p^{(i)}(T + \Pi) \approx c_p^{(i)}(T) + \Pi.$$

PROOF. This follows easily from A1. □

LEMMA 3. *If Clock Synchronization Condition S1 holds for i, and p and q are nonfaulty up to time* $T^{(i+2)}$, *then for any T in* $S^{(i)}$ *and any* $\Pi$ *such that* $|\Pi| < R$ *and*

$\rho\Pi \ll \epsilon$:

$$|c_p^{(i)}(T + \Pi + \Delta_{qp}) - c_q^{(i)}(T + \Pi)| \lesssim \epsilon + \rho S.$$

PROOF. Letting $T_0$ be as in A2, we have

$$\begin{aligned}
&|c_p^{(i)}(T + \Pi + \Delta_{qp}) - c_q^{(i)}(T + \Pi)| \\
&\quad = |c_p^{(i)}(T_0 + \Delta_{qp} + T - T_0 + \Pi) - c_q^{(i)}(T_0 + T - T_0 + \Pi)| \\
&\quad \le |c_p^{(i)}(T_0 + \Delta_{qp}) - c_q^{(i)}(T_0)| + \rho|T - T_0 + \Pi| \\
&\qquad\qquad \text{[by two applications of Lemma 2]} \\
&\quad \lesssim \epsilon + \rho|T - T_0| \qquad \text{[by A2 and the hypothesis that } \rho\Pi \text{ is negligible].}
\end{aligned}$$

The result now follows from the hypothesis that $T$ is in $S^{(i)}$. □

LEMMA 4. *If Clock Synchronization Condition S1 holds for i, and processes p, q, and r are nonfaulty up to time $T^{(i+2)}$, then for any T in $S^{(i)}$:*

$$|c_p^{(i)}(T) + \overline{\Delta}_{rp} - [c_q^{(i)}(T) - \overline{\Delta}_{rq}]| \lesssim 2(\epsilon + \rho S).$$

PROOF. It follows from Lemma 1 that $|\Delta_{rp}|$ and $|\Delta_{qp}|$ are both less than $\Delta$, so $\overline{\Delta}_{rp} = \Delta_{rp}$, $\overline{\Delta}_{rq} = \Delta_{rq}$, and $\rho\Delta_{rp}$ and $\rho\Delta_{rq}$ are both negligible. We therefore have

$$\begin{aligned}
&|c_p^{(i)}(T) + \overline{\Delta}_{rp} - [c_q^{(i)}(T) + \overline{\Delta}_{rq}]| \\
&\quad = |c_p^{(i)}(T) + \Delta_{rp} - [c_q^{(i)}(T) + \Delta_{rq}]| \\
&\quad \approx |c_p^{(i)}(T + \Delta_{rp}) - c_q^{(i)}(T + \Delta_{rq})| \qquad \text{[by Lemma 2]} \\
&\quad \le |c_p^{(i)}(T + \Delta_{rp}) - c_r^{(i)}(T)| + |c_r^{(i)}(T) - c_q^{(i)}(T + \Delta_{rq})| \\
&\quad \lesssim 2(\epsilon + \rho S) \qquad \text{[by Lemma 3]}
\end{aligned}$$

proving the result. □

LEMMA 5. *If Clock Synchronization Condition S1 holds for i, and Processes p and q are nonfaulty up to time $T^{(i+2)}$, then for any r and any T in $S^{(i)}$*

$$|c_p^{(i)}(T) + \overline{\Delta}_{rp} - [c_q^{(i)}(T) + \overline{\Delta}_{rq}]| < \delta + 2\Delta.$$

PROOF. By the assumption that S1 holds for $i$, we have

$$|c_p^{(i)}(T) - c_q^{(i)}(T)| < \delta.$$

Since $|\overline{\Delta}_{rp}|$ and $|\overline{\Delta}_{rq}|$ are by definition no larger than $\Delta$, the result follows immediately. □

We now complete the proof of the theorem. Assume that processes $p$ and $q$ are both nonfaulty until time $T^{(i+2)}$. For notational convenience, let $T$ denote $T^{(i+1)}$. For any $T'$ in $R^{(i+1)}$ we have

$$\begin{aligned}
&|c_p^{(i+1)}(T') - c_q^{(i+1)}(T')| < |c_p^{(i+1)}(T) - c_q^{(i+1)}(T)| + \rho R \qquad \text{[by A1]} \\
&\quad = |c_p^{(i)}(T + \Delta_p) - c_q^{(i)}(T + \Delta_q)| + \rho R \qquad \text{[from the algorithm]} \\
&\quad \approx |c_p^{i}(T) + \Delta_p - [c_q^{(i)}(T) + \Delta_q]| + \rho R \qquad \text{[by Lemma 2, since } |\Delta_p|, |\Delta_q| < \Delta] \\
&\quad = \left|\left(\frac{1}{n}\right)\sum_{r=1}^{n}(c_p^{(i)}(T) + \overline{\Delta}_{rp} - [c_q^{(i)}(T) + \overline{\Delta}_{rq}])\right| + \rho R \\
&\qquad\qquad \text{[by definition of } \Delta_p \text{ and } \Delta_q] \\
&\quad \le \left(\frac{1}{n}\right)\sum_{r=1}^{n}|c_p^{(i)}(T) + \overline{\Delta}_{rp} - [c_q^{(i)}(T) + \overline{\Delta}_{rq}]| + \rho R \\
&\quad \lesssim \left(\frac{1}{n}\right)[2(n - m)(\epsilon + \rho S) + m(\delta + 2\Delta)] + \rho R,
\end{aligned}$$

where the last inequality is obtained by applying Lemma 4 to the $n - m$ nonfaulty processes $r$ and Lemma 5 to the remaining $m$ processes. Since $\Delta \approx \delta + \epsilon$, a little algebraic manipulation shows that if

$$\delta \gtrsim n'(2\epsilon + \rho(R + 2S')),$$

then

$$\frac{1}{n}[2(n - m)(\epsilon + \rho S) + m(\delta + 2\Delta)] + \rho R \lesssim \delta.$$

Combining this with the above string of inequalities, we see that for any $T'$ in $R^{(i+1)}$

$$|c_p^{(i+1)}(T') - c_q^{(i+1)}(T')| \lesssim \delta,$$

so S1 holds for $i + 1$. This completes the proof of Theorem 1. □

For any clock synchronization algorithm, we will have $\delta \geq \delta_1 + \rho R$, where $\delta_1$ is the closeness with which the clocks can be resynchronized and $\rho R$ is how far they can drift apart during an $R$-SECOND interval. In the interactive convergence algorithm CNV, there is a term $(n' - 1)\rho R$ in $\delta_1$, so how close the clocks can be resynchronized depends upon how far apart they are allowed to drift.

## 5. *Interactive Consistency Algorithms*

We begin our rigorous discussion of the interactive consistency algorithms by formalizing conditions CC1 and CC2, given in Section 2.3. When $p$ "obtains the value" of $r$'s clock, what it actually finds is a constant $\overline{\Delta}_{rp}$ such that

$$c_r^{(i)}(T) \approx c_p^{(i)}(T + \overline{\Delta}_{rp}).$$

(The values $\overline{\Delta}_{rp}$ are not the same ones defined in the interactive convergence algorithm.) We also allow the possibility that if $r$ is faulty, then $p$ may not be able to read $r$'s clock. This is denoted by letting $\overline{\Delta}_{rp}$ have the special value NULL. Recalling the definition of $\Delta_{qp}$ given by A2, and letting a NULL clock be approximately equal only to another NULL clock, we see that conditions CC1 and CC2 can be restated as follows:

*CC.* For some constant $\Omega \ll R$ and all $i$: if the Clock Synchronization Condition holds for $i$, then for any processes $p$ and $q$ that are nonfaulty up to time $T^{(i+2)}$:

(1) For all $r \neq p, q$: either

  (a) $|\overline{\Delta}_{rp} - [\Delta_{qp} + \overline{\Delta}_{rq}]| < \Omega$, or
  (b) $\overline{\Delta}_{rp} = \overline{\Delta}_{rq} =$ NULL.

(2) $\overline{\Delta}_{qp} \neq$ NULL, and $|\overline{\Delta}_{qp} - \Delta_{qp}| < \Omega$.

For convenience, we let $\overline{\Delta}_{pp} \equiv 0$ for all $p$. Condition CC2 is then equivalent to CC1(a) for $r = q$.

Before stating our next result, we introduce some notation. We let $\mathbf{n}$ denote the set $\{1, \ldots, n\}$. A *multiset* is a set in which the same element can appear more than once. We use ordinary set notation for describing multisets, so the multiset $\{1, 1, 2\}$ contains three elements, two of which are equal. The multiset $\{a_i : i \in \mathbf{n}\}$ contains $n$ elements, not all of which need be distinct. If $\mathbf{M}$ is a multiset, then

"median **M**" denotes the median of **M**, defined by

$$\text{median } \mathbf{M} \equiv a_{\lfloor n/2 \rfloor},$$

where $\mathbf{M} = \{a_1, \ldots, a_n\}$ with $a_1 \leq a_2 \leq \cdots \leq a_n$.

Our two interactive consistency algorithms are based upon the following result.

THEOREM 2. *If* $m \leq \lfloor n/2 \rfloor$, $\delta_0 < \Omega + \epsilon + \rho S$, *and CC holds for all* $i$, *then letting*

$$c_p^{(i+1)}(T) \equiv c_p^{(i)}(T + \Delta_p)$$

*where*

$$\Delta_p \equiv \textit{median}\{\overline{\Delta}_{rp} : r \in \mathbf{n} \textit{ and } \overline{\Delta}_{rp} \neq \textit{NULL}\},$$

*satisfies the Clock Synchronization Condition for all* $i$, *with*

$$\delta \approx \Omega + \epsilon + \rho(R + S),$$
$$\Sigma \approx 2(\Omega + \epsilon) + \rho(R + S).$$

The proof of Theorem 2 requires the following two results about medians.

LEMMA 6. *If* $|a_r - b_r| < \pi$ *for all* $r \in \mathbf{n}$, *then*

$$|\textit{median}\{a_r : r \in \mathbf{n}\} - \textit{median}\{b_r : r \in \mathbf{n}\}| < \pi.$$

PROOF. We prove the stronger result that for any $k$: the $k$th highest values of the multisets $\{a_r\}$ and $\{b_r\}$ lie within $\pi$ of one another. Let the permutations $\alpha$ and $\beta$ be chosen such that:

$$a_{\alpha(1)} \leq a_{\alpha(2)} \leq \cdots \leq a_{\alpha(n)},$$
$$b_{\beta(1)} \leq b_{\beta(2)} \leq \cdots \leq b_{\beta(n)}.$$

We prove that, for all $k$, $|a_{\alpha(k)} - b_{\beta(k)}| < \pi$.

There are at least $k$ values of $i$ such that $a_i \leq a_{\alpha(k)}$. However, the hypothesis implies that, if $a_i \leq a_{\alpha(k)}$, then $b_i < a_{\alpha(k)} + \pi$. Hence there are at least $k$ values of $i$ such that $b_i < a_{\alpha(k)} + \pi$, which implies that $b_{\alpha(k)} < a_{\alpha(k)} + \pi$. A symmetric argument shows that $a_{\alpha(k)} < b_{\alpha(k)} + \pi$, and combining these two inequalities gives the desired result. □

LEMMA 7. *If* $|a_r - a| < \pi$ *for a majority of values* $r$ *in* **n**, *then*

$$|\textit{median}\{a_r : r \in \mathbf{n}\} - a| < \pi.$$

PROOF. It is easy to see that if $A$ is any submultiset containing a majority of the elements of $\{a_r\}$, then

$$\min(A) \leq \text{median}\{a_r : r \in n\} \leq \max(A).$$

Letting $A$ be the multiset $\{a_r : |a_r - a| < \pi\}$, this implies that

$$a - \pi \leq \text{median}\{a_r : r \in \mathbf{n}\} \leq a + \pi,$$

which proves the lemma. □

PROOF OF THEOREM 2. The proof is by induction on $i$. For $i = 0$, the result is trivial. (Note that S2 is vacuous for $i = 0$.) Assume that the theorem is true for $i$. By CC2 and Lemma 1, we see that

$$|\overline{\Delta}_{qp}| < \Omega + \delta + \epsilon$$

for all nonfaulty $p$ and $q$, so S2 follows easily from Lemma 7.

Since $\Delta_p$ is the median of the $\overline{\Delta}_{rp}$, and a majority of the processes $r$ are nonfaulty, the above inequality shows that we can neglect terms of order $\rho\Delta_p$, and likewise

terms of order $\rho\Delta_q$. Lemma 1 implies that we can neglect terms of order $\rho\Delta_{qp}$. Letting $T = T^{(i+1)}$, we then have

$$
\begin{aligned}
&|c_p^{(i+1)}(T) - c_q^{(i+1)}(T)| \\
&\quad = |c_p^{(i)}(T + \Delta_p) - c_q^{(i)}(T + \Delta_q)| && \text{[by hypothesis]} \\
&\quad = |c_p^{(i)}(T + \Delta_{qp} + \Delta_p - \Delta_{qp}) - c_q^{(i)}(T + \Delta_q)| \\
&\quad \approx |c_p^{(i)}(T + \Delta_{qp}) + \Delta_p - \Delta_{qp} - [c_q^{(i)}(T) + \Delta_q]| && \text{[by Lemma 2]} \\
&\quad \lesssim |\Delta_p - \Delta_{qp} - \Delta_q| + \epsilon + \rho S && \text{[by Lemma 3]} \\
&\quad = |\operatorname{median}\{\overline{\Delta}_{rp} - \Delta_{qp} : r \in \mathbf{n} \text{ and } \overline{\Delta}_{rp} \neq \text{NULL}\} \\
&\qquad - \operatorname{median}\{\overline{\Delta}_{rq} : r \in \mathbf{n} \text{ and } \overline{\Delta}_{rq} \neq \text{NULL}\}| + \epsilon + \rho S \\
&\quad \lesssim \Omega + \epsilon + \rho S && \text{[by CC and Lemma 6].}
\end{aligned}
$$

Condition S1 follows easily from this inequality and A1. □

5.1. THE ALGORITHM COM. Achieving Condition CC requires that the processes not only read each other's clocks, but also send values to one another. If $x_p$ is a value that process $p$ sends to the other processes, then we let $x_{pq}$ denote the value that $q$ receives from $p$. The manner in which the value is transmitted is irrelevant—it might be sent as a message from $p$ to $q$, or $p$ might leave it in some register where $q$ can read it. We assume that if $p$ and $q$ are both nonfaulty, then $x_{pq} = x_p$. If $p$ or $q$ is faulty, then $x_{pq}$ may have any value.

We specify Algorithm COM as a recursive algorithm by which a process $q$ obtains a value $\text{COM}(m, \mathbf{P}, x_p, p)_q$ from process $p$, where $x_p$ is the value that $p$ is sending and $\mathbf{P}$ is some set of processes. The value $x_p$ being sent by $p$ represents the difference between some clock and $p$'s clock. If $x_p = 0$, then $p$ is sending its own clock value. The actual clock-synchronization algorithm consists of each process $q$ letting $\text{COM}(m, \mathbf{n}, 0, q)_p$ be the $\overline{\Delta}_{qp}$ of Theorem 2. We write $\mathbf{P} - p$ to denote $\mathbf{P} - \{p\}$, for any set $\mathbf{P}$.

ALGORITHM COM. *For any integer $m \geq 0$, any subset $\mathbf{P}$ of $\mathbf{n}$, any value $x_p$ and any $p, q \in \mathbf{P}$:*

$$
\begin{aligned}
&COM(0, \mathbf{P}, x_p, p)_q \equiv x_{pq} + \Delta_{pq} \\
&COM(m, \mathbf{P}, x_p, p)_q \equiv median\{COM(m - 1, \mathbf{P} - p, x_{pr} + \Delta_{pr}, r)_q : r \in \mathbf{P} - p\}
\end{aligned}
$$

To prove the required properties of Algorithm COM, we need the following result.

LEMMA 8. *If Clock Synchronization Condition S1 holds for $i$, and processes $p$, $q$, and $r$ are nonfaulty up to time $T^{(i+2)}$, then*

$$|\Delta_{pr} + \Delta_{rq} - \Delta_{pq}| \lesssim 3\epsilon + 2\rho S.$$

PROOF. Let $T$ be the time, obtained from A2, such that

$$|c_p^{(i)}(T) - c_q^{(i)}(T + \Delta_{pq})| < \epsilon.$$

We then have

$$
\begin{aligned}
&|\Delta_{pr} + \Delta_{rq} - \Delta_{pq}| \\
&\quad \approx |c_q^{(i)}(T + \Delta_{pr} + \Delta_{rq}) - c_q^{(i)}(T + \Delta_{pq})| && \text{[by Lemma 2]} \\
&\quad \leq |c_q^{(i)}(T + \Delta_{pr} + \Delta_{rq}) - c_r^{(i)}(T + \Delta_{pr})| \\
&\qquad + |c_r^{(i)}(T + \Delta_{pr}) - c_p^{(i)}(T)| + |c_p^{(i)}(T) - c_q^{(i)}(T + \Delta_{pq})| \\
&\quad \lesssim 3\epsilon + 2\rho S && \text{[by A2 and Lemma 3]}
\end{aligned}
$$

which is the required result. □

The following result is the analogue of Lemma 1 of [6].

LEMMA 9. *For all m and k, if $n > 2k + m$, Clock Synchronization Condition S1 holds for i, and all but at most k processes in* **P** *are nonfaulty up to time $T^{(i+2)}$, then for any of those nonfaulty processes p and q, and any $x_p$:*

$$|COM(m, \mathbf{P}, x_p, p)_q - [x_p + \Delta_{pq}]| \lesssim m(3\epsilon + 2\rho S).$$

PROOF. The proof is by induction on $m$. The result is trivial for $m = 0$. Assume it for $m - 1$. For any processor $r$ in $\mathbf{P} - p$ that is nonfaulty up to time $T^{(i+2)}$, we have

$$\begin{aligned}
&|\mathrm{COM}(m - 1, \mathbf{P} - p, x_{pr} + \Delta_{pr}, r)_q - [x_p + \Delta_{pq}]| \\
&\quad = |\mathrm{COM}(m - 1, \mathbf{P} - p, x_{pr} + \Delta_{pr}, r)_q - [x_{pr} + \Delta_{pr} + \Delta_{rq}] \\
&\qquad + [\Delta_{pr} + \Delta_{rq} - \Delta_{pq}]| \qquad [\text{since } p \text{ and } r \text{ nonfaulty implies } x_{pr} = x_p] \\
&\quad \lesssim (m - 1)(3\epsilon + 2\rho S) + (3\epsilon + 2\rho S),
\end{aligned}$$

where the last inequality comes from Lemma 8 and the induction hypothesis, which can be applied because $\mathbf{P} - p$ has $n - 1$ elements and $n - 1 > 2k + (m - 1)$. Therefore, for every nonfaulty process $r$, we have

$$|\mathrm{COM}(m - 1, \mathbf{P} - p, x_{pr} + \Delta_{pr}, r)_q - [x_p + \Delta_{pq}]| \lesssim m(3\epsilon + 2\rho S).$$

The lemma now follows from Lemma 7, since $n > 2k + m \geq 2k + 1$. □

Our next lemma is the analogue of Theorem 1 of [6].

LEMMA 10. *If Clock Synchronization Condition S1 holds for i, and* **P** *is a set containing more than 3m processes, all but at most m of which are nonfaulty up to time $T^{(i+2)}$, then for any of the nonfaulty processes p and q:*

(1) *For all r in* **P**,

$$|COM(m, \mathbf{P}, x_r, r)_p - [\Delta_{qp} + COM(m, \mathbf{P}, x_r, r)_q]| \lesssim (2m + 1)(3\epsilon + 2\rho S),$$

(2) $|COM(m, \mathbf{P}, x_q, q)_p - x_q - \Delta_{qp}| \lesssim m(3\epsilon + 2\rho S)$.

PROOF. Part 2 follows immediately from Lemma 9 by letting $k = m$. Part 1 is proved by induction. For $m = 0$, it follows easily from the definition of COM and Lemma 8. Let $m > 0$ and assume it holds for $m - 1$. We consider two cases: (i) $r$ faulty and (ii) $r$ nonfaulty.

If $r$ is faulty, then there are at most $m - 1$ faulty processes in $\mathbf{P} - r$, and we can apply the induction hypothesis to obtain

$$\begin{aligned}
&|\mathrm{COM}(m - 1, \mathbf{P} - r, x_{rs}, s)_p - \Delta_{qp} - \mathrm{COM}(m - 1, \mathbf{P} - r, x_{rs}, s)_q| \\
&\quad \lesssim (2(m - 1) + 1)(3\epsilon + 2\rho S)
\end{aligned}$$

for any $s$ in $\mathbf{P} - r$. The result now follows easily from Lemma 6.

If $r$ is nonfaulty, then we can apply the inequality from part 2 to obtain

$$\begin{aligned}
&|\mathrm{COM}(m, \mathbf{P}, x_r, r)_p - [x_r + \Delta_{rp}]| \lesssim m(3\epsilon + 2\rho S), \\
&|\mathrm{COM}(m, \mathbf{P}, x_r, r)_q - [x_r + \Delta_{rq}]| \lesssim m(3\epsilon + 2\rho S).
\end{aligned} \tag{3}$$

We then have

$$\begin{aligned}
&|\mathrm{COM}(m, \mathbf{P}, x_r, r)_p - \Delta_{qp} - \mathrm{COM}(m, \mathbf{P}, x_r, r)_q| \\
&\quad = |\mathrm{COM}(m, \mathbf{P}, x_r, r)_p - [x_r + \Delta_{rp}] \\
&\qquad - (\mathrm{COM}(m, \mathbf{P}, x_r, r)_q - [x_r + \Delta_{rq}]) + \Delta_{rp} - \Delta_{rq} - \Delta_{qp}| \\
&\quad \lesssim 2m(3\epsilon + 2\rho S) + (3\epsilon + 2\rho S),
\end{aligned}$$

where the last inequality follows from the triangle inequality (3) and Lemma 8. This finishes the proof of part 1 for $m$. □

Taking $x_r = 0$, Lemma 10 yields the following result.

THEOREM 3. *If all but at most m processes are nonfaulty up to time* $T^{(i+2)}$, *and* $n > 3m$, *then Condition CC is satisfied by*

$$\overline{\Delta}_{qp} = COM(m, \mathbf{n}, 0, q)_p,$$

*with* $\Omega \approx (2m + 1)(3\epsilon + 2\rho S)$.

Combining this with Theorem 2 yields our first interactive consistency clock synchronization algorithm, with

$$\delta \approx (6m + 4)\epsilon + (4m + 3)\rho S + \rho R,$$
$$\Sigma \approx (12m + 8)\epsilon + (8m + 5)\rho S + \rho R.$$

This algorithm requires that $n$ be greater than $3m$—that is, that more than two-thirds of the processes be nonfaulty. As shown in [9], this is the best one can do.

5.2. THE ALGORITHM CSM. We begin our formal development of Algorithm CSM with a precise statement of Assumption A2′.

*A2′.* If an event in a process $q$ occurring at (real) time $t_0$ causes $q$ to send a message to a process $p$, then that message arrives at a time $t_1$ such that

(a) if $p$ and $q$ are nonfaulty, then $|t_1 - t_0 - \gamma| < \epsilon$,
(b) $t_1 - t_0 > \gamma - \epsilon$,

for some constant $\gamma$ such that $n\rho\gamma \ll \epsilon$.

In practice, the value of $\gamma$ may depend upon $p$ and $q$ and on the type of event generating the message. To avoid having to cope with all these different values, we assume a single $\gamma$ for all messages. The only restriction we place on the size of $\gamma$ is that $n\rho\gamma \ll \epsilon$, which means that $\gamma$ may be "medium-sized." It will typically be larger than $\epsilon$ but much smaller than $R$.

We next restate our assumption of unforgeable digital signatures. Formally, a digital signature mechanism consists of a function $S_p$ for each process $p$, satisfying the following condition:

*A3.* For any process $p$ and any data item $D$:

(a) No faulty process other than $p$ can generate $S_p[D]$.
(b) For any $X$, any process can determine if $X$ equals $S_p[D]$.

Note that the first assumption is stronger than the one made in [6], since it does not permit one faulty process to forge the signatures of another faulty process.

Assumption A3(a) means that a faulty process cannot generate any arbitrary value, so it restricts the class of faults that may occur. Hence, we can hope to find a clock synchronization algorithm to handle $m$ faults with fewer than $3m + 1$ processes, and, indeed, our second interactive consistency algorithm requires that only a majority of the processes be nonfaulty.

We define the message $M(T, p_0 \cdots p_s)$, for any sequence $p_0, \ldots, p_s$ of processes—including the null sequence $\lambda$—as follows:

$$M(T, \lambda) \equiv (T, p_0, S_{p_0}(T)),$$
$$M(T, p_0 \cdots p_s) \equiv (M(T, p_0 \cdots p_{s-1}), p_s, S_{p_s}[M(T, p_0 \cdots p_{s-1})]).$$

By A3(a), the value $M(T, p_0 \cdots p_s)$ can be generated only as the result of process $p_0$ sending the message $M(T, p_0)$ to process $p_1$, which sends the message $M(T, p_0p_1)$ to process $p_2 \cdots$ which sends the message $M(T, p_0 \cdots p_{s-1})$ to process $p_s$, which generates $M(T, p_0 \cdots p_s)$. Moreover, A3(b) implies that any process can determine whether a given data item $X$ equals $M(T, p_0 \cdots p_s)$ for some $T$ and $p_0 \cdots p_s$.

In Algorithm CSM, for some time $T_p$ in $S^{(i)}$, process $p$ sends the message $M(T_p, p)$ to all other processes when its clock reaches $T_p$. Immediately upon receiving this message, each other process $p_1$ sends the message $M(T_p, pp_1)$ to all processes other than itself and $p$, and so forth. Any process $q$ will therefore receive messages $M(T_p, pp_1 \cdots p_s)$ for many different sequences $p_1 \cdots p_s$. Each such message tells $q$ that $p$'s clock read $T_p$ approximately $(s + 1)\gamma$ SECONDS ago. If $p$ and all the $p_i$ are nonfaulty, then this message is correct. If one or more of the $p_i$ are faulty, then they can either fail to relay the message, so $q$ never receives it, or they can delay it. However, they cannot alter the value of $T_p$ or cause the message to arrive too early. Hence, process $q$ believes the message indicating the earliest time at which $p$'s clock reached $T_p$.

There is a practical problem in implementing this approach. In order to perform the appropriate message relaying, a process must be prepared to receive the incoming message. This may require that the process not do anything else while waiting, so it should know when the message will arrive and be able to ignore the message if it does not arrive when it should. Since the uncertainty in message transmission time is $\epsilon$, and the difference between $p$'s clock and $q$'s clock is $\delta$, $q$ can expect to receive the message $M(T_p, p)$ within about $\epsilon + \delta$ seconds of when its clock reads $T_p + \gamma$. If $q$ relays this message only if it arrives when it should, then another process $r$ can expect to receive the message $M(T_p, pq)$ within about $2(\epsilon + \delta)$ of when its clock reads $T_p + 2\gamma$. Continuing, this leads us to the following definition:

*Definition* 3. The message $M(T, p_0 \cdots p_s)$ is said to arrive *on time* at process $q$ if either

(1) $s \geq 0$ and the message arrives at (real) time $c_q^{(i)}(T')$, or
(2) $s = -1$ (so $p_0 \cdots p_s$ is the null sequence) and $T' = T$

and $| T' - T - (s + 1)\gamma | \leq (s + 1)(\delta + \epsilon)$.

The $\delta$ in this definition is the same one as in the Clock Synchronization Condition. Its value will be given later.

The following algorithm describes how each process $q$ determines the value $\overline{\Delta}_{pq}$ for every $p \neq q$.

ALGORITHM CSM($m$). *For each process $p$, and for some clock time $T_p$ in $S^{(i)}$:*

(1) *When its clock $c_p^{(i)}$ reaches $T_p$, process $p$ sends the message $M(T_p, p)$ to every other process.*
(2) *For each process $q \neq p$:*
  (*A*) *Process $q$ initializes $\overline{\Delta}_{pq}$ to $\infty$.*
  (*B*) *If the message $M(T_p, pp_1 \cdots p_s)$ arrives on time at $q$, at time $c_q^{(i)}(T)$, and $T - T_p - (s + 1)\gamma < \overline{\Delta}_{pq}$, then*
    (*a*) *Process $q$ sets $\overline{\Delta}_{pq}$ equal to $T - T_p - (s + 1)\gamma$.*
    (*b*) *If $s < m$, then immediately upon receiving this message, $q$ sends the message $M(T_p, pp_1 \cdots p_sq)$ to every other process $q'$ not contained among the processes $pp_1 \cdots p_s$.*

(C) *At time* $T_p + (m+1)(\gamma + \delta + \epsilon)$, *if* $\overline{\Delta}_{pq} = \infty$, *then* $q$ *sets* $\overline{\Delta}_{pq}$ *equal to* NULL.

We have assumed A2′ instead of A2, so the values $\Delta_{qp}$ are not yet defined. In order to apply Theorem 2, we must define the $\Delta_{qp}$ and prove A2.

LEMMA 11. *Assumption A2 is satisfied, except with the strict inequality replaced by approximate inequality, if* $\Delta_{qp}$ *is defined to equal* $T - T_q - \gamma$, *where* $T$ *is the value such that* $p$ *receives the message* $M(T_q, q)$ *at time* $c_p^{(i)}(T)$, *and to have any value if* $p$ *receives no such message.*

PROOF. If $p$ and $q$ are nonfaulty, then the message $M(T_q, q)$ is sent by $q$ and received by $p$. Taking $T_0 = T_q$, we find

$$\begin{aligned} &|c_p^{(i)}(T_0 + \Delta_{qp}) - c_q^{(i)}(T_0)| \\ &\quad = |c_p^{(i)}(T - \gamma) - c_q^{(i)}(T_q)| \\ &\quad \approx |c_p^{(i)}(T) - \gamma - c_q^{(i)}(T_q)| \qquad \text{[by Lemma 2]} \\ &\quad < \epsilon \qquad \text{[by A2′(a)]} \end{aligned}$$

which proves the lemma. □

Lemma 11 allows us to use the earlier results that assumed A2. (Since these results all involve approximate inequalities, they are not invalidated when the exact inequality in A2 is replaced by an approximate inequality.) We now prove the main result for Algorithm CSM.

THEOREM 4. *If all but at most* $m$ *processes are nonfaulty up to time* $T^{(i+2)}$, *then the values* $\overline{\Delta}_{pq}$ *found by Algorithm CSM*$(m)$ *satisfy condition CC with* $\Omega \approx (m+5)\epsilon + 2\rho S$.

The proof uses the following lemmas.

LEMMA 12. *Let Clock Synchronization Condition S1 hold for* $i$, *and let* $p$ *and* $q$ *be nonfaulty up to time* $T^{(i+2)}$. *If the message* $M(T, p_0 \cdots p_s)$ *arrives on time at* $p$, *and* $s < m$, *then the message* $M(T, p_0 \cdots p_s p)$ *arrives on time at* $q$.

PROOF. Let $c_p^{(i)}(T')$ be the time at which $M(T, p_0 \cdots p_s)$ arrives at $p$, letting $T' = T$ if $p_0 \cdots p_s$ is the null sequence, and let $c_q^{(i)}(T'')$ be the time at which $M(T, p_0 \cdots p_s p)$ arrives at $q$. Then

$$\begin{aligned} &|c_p^{(i)}(T'') - c_p^{(i)}(T' + \gamma)| \\ &\quad \approx |c_p^{(i)}(T'') - c_p^{(i)}(T') - \gamma| \\ &\quad \leq |c_q^{(i)}(T'') - c_p^{(i)}(T') - \gamma| + \delta \qquad \text{[by S1]} \\ &\quad \lesssim \epsilon + \delta \qquad \text{[by A2].} \end{aligned}$$

It follows from A1, and the assumption that $\rho\gamma$ is negligible, that

$$|T'' - T' - \gamma| \lesssim \delta + \epsilon. \tag{4}$$

We then have

$$\begin{aligned} &|T'' - T - (s+2)\gamma| \\ &\quad \leq |T'' - T' - \gamma| + |T' - T - (s+1)\gamma| \\ &\quad \lesssim \delta + \epsilon + (s+1)(\delta + \epsilon) \quad \text{[by 4 and the on-time arrival of } M(T, p_0 \cdots p_s)] \end{aligned}$$

which implies that $M(T, p_0 \cdots p_s p)$ arrives on time at $q$. □

LEMMA 13. *If Clock Synchronization Condition S1 holds for i, all but at most m processes are nonfaulty up to* $T^{(i+2)}$, *and p and q are among the nonfaulty ones, then for any process r: if* $\overline{\Delta}_{rp} \neq$ NULL *then* $\overline{\Delta}_{rq} \neq$ NULL *and*

$$\overline{\Delta}_{rq} + \Delta_{qp} \lesssim \overline{\Delta}_{rp} + (m + 3)\epsilon + \rho S.$$

(*For* $r = q$ *or* $p$, $\overline{\Delta}_{rr}$ *is defined to be* 0.)

PROOF. Let $r_0 = r$, and let $T$ be the time such that

$$\overline{\Delta}_{rp} = T - T_r - (s + 1)\gamma \tag{5}$$

and the message $M(T_r, r_0 \cdots r_s)$ arrived at $p$ (on time) at time $c_p^{(i)}(T)$. (If $r = p$, so $s = -1$, then $T = T_r$.) We consider two cases:

(1) $q = r_j$
(2) $q$ not in the sequence $r_0 \cdots r_s$.

In case 1, it follows from A3(a) that the message $M(T_r, r_0 \cdots r_{j-1})$ must have arrived on time at $q$ at some time $c_q^{(i)}(T')$, so $\overline{\Delta}_{rq} \neq$ NULL, and

$$\overline{\Delta}_{rq} \leq T' - T_r - j\gamma. \tag{6}$$

A simple induction argument using A2′(b) shows that

$$c_p^{(i)}(T) - c_q^{(i)}(T') - (s - j + 1)\gamma > -(s - j + 1)\epsilon. \tag{7}$$

We then have

$$\begin{aligned}
&c_p^{(i)}(T) - c_p^{(i)}(T' + (s - j + 1)\gamma) \\
&\quad \gtrsim c_p^{(i)}(T) - c_q^{(i)}(T' + (s - j + 1)\gamma - \Delta_{qp}) - \epsilon - \rho S && \text{[by Lemma 3]} \\
&\quad \approx c_p^{(i)}(T) - c_q^{(i)}(T') - (s - j + 1)\gamma + \Delta_{qp} - \epsilon - \rho S && \text{[by Lemma 2]} \\
&\quad > -(s - j + 2)\epsilon - \rho S + \Delta_{qp} && \text{[by (7)].}
\end{aligned}$$

By A1, this yields

$$T - (T' + (s - j + 1)\gamma) \gtrsim -(s - j + 2)\epsilon - \rho S + \Delta_{qp},$$

so

$$T' - j\gamma + \Delta_{qp} \lesssim T - (s + 1)\gamma + (s - j + 2)\epsilon + \rho S.$$

Subtracting $T_r$ from both sides of this inequality, we see that (5) and (6) imply

$$\overline{\Delta}_{rq} + \Delta_{qp} \lesssim \overline{\Delta}_{rp} + (s - j + 2)\epsilon + \rho S,$$

which yields the desired result, since $s \leq m$.

For case 2, $q$ not equal to any of the $r_j$, we consider two subcases: $s < m$ and $s = m$. If $s < m$, then $p$ sends $q$ the message $M(T_r, r_0 \cdots r_s p)$, which, by Lemma 12, arrives on time at $q$. Hence, $\overline{\Delta}_{rq} \neq$ NULL and

$$\overline{\Delta}_{rq} \leq T' - T_r - (s + 2)\gamma, \tag{8}$$

where $c_q^{(i)}(T')$ is the time at which the message arrives. We then have

$$\begin{aligned}
&c_p^{(i)}(T' + \Delta_{qp} - \gamma) - c_p^{(i)}(T) \\
&\quad \approx c_p^{(i)}(T' + \Delta_{qp}) - c_p^{(i)}(T) - \gamma && \text{[by A1]} \\
&\quad \lesssim c_q^{(i)}(T') - c_p^{(i)}(T) - \gamma + \epsilon + \rho S && \text{[by Lemma 3]} \\
&\quad < 2\epsilon + \rho S && \text{[by A2′(a)].}
\end{aligned}$$

This implies that

$$T' + \Delta_{qp} - \gamma - T \lesssim 2\epsilon + \rho S,$$

which can be rewritten as

$$T' - (s + 2)\gamma + \Delta_{qp} \lesssim T - (s + 1)\gamma + 2\epsilon + \rho S.$$

Subtracting $T_r$ from both sides of this inequality shows that the desired result follows immediately from (5) and (8).

Finally, we consider the case $s = m$, where $q$ is not one of the $r_j$. Since there are at most $m$ faulty processes, there is at least one process $r_j$ that is nonfaulty up to time $T^{(i+2)}$. Let $c_q^{(i)}(T')$ be the time at which $r_j$ received the message $M(T_r, r_0 \cdots r_{j-1})$—or at which it sent the message $M(T_r, r)$, if $j = 0$. The same argument as before shows that (7) again holds.

By Lemma 12, the message $M(T_r, r_0 \cdots r_j)$ arrives on time at $q$, so $\overline{\Delta}_{rq} \neq$ NULL, and

$$\overline{\Delta}_{rq} \leq T'' - T_r - (j + 1)\gamma, \tag{9}$$

where $c_q^{(i)}(T'')$ is the time at which the message arrives. By A2′(a) we have

$$c_q^{(i)}(T'') - c_q^{(i)}(T') - \gamma < \epsilon,$$

and combining this with (7) yields

$$c_p^{(i)}(T) - c_q^{(i)}(T'') - (s - j)\gamma > -(s - j + 2)\epsilon.$$

Using Lemma 3, we can deduce from this that

$$c_p^{(i)}(T) - c_p^{(i)}(T'' + \Delta_{qp}) - (s - j)\gamma \gtrsim -(s - j + 3)\epsilon - \rho S.$$

Since $\rho(s - j)\gamma$ is negligible, this implies by A1 that

$$T - T'' - \Delta_{qp} - (s - j)\gamma \gtrsim -(s - j + 3)\epsilon - \rho S,$$

which can be rewritten as

$$T'' - (j + 1)\gamma + \Delta_{qp} \lesssim T - (s + 1)\gamma + (s - j + 3)\epsilon + \rho S.$$

Subtracting $T_r$ from both sides, we obtain the desired result from (9) and (5). □

PROOF OF THEOREM 4. Let $p$, $q$ be as in Condition CC. It follows easily from Lemma 12 that $\overline{\Delta}_{qp} \neq$ NULL. Condition CC2 then follows from CC1(a) for $r = q$. It therefore suffices to prove CC1 for all $r$. This requires proving that if $\overline{\Delta}_{rp} \neq$ NULL, then $\overline{\Delta}_{rq} \neq$ NULL and

$$\overline{\Delta}_{rp} - \Delta_{qp} - \overline{\Delta}_{rq} \lesssim (m + 5)\epsilon + 2\rho S,$$
$$\overline{\Delta}_{rq} + \Delta_{qp} - \overline{\Delta}_{rp} \lesssim (m + 5)\epsilon + 2\rho S.$$

The fact that $\overline{\Delta}_{rq} \neq$ NULL and the second inequality follow from Lemma 13. Reversing $p$ and $q$ in Lemma 13, we obtain

$$\overline{\Delta}_{rp} + \Delta_{pq} - \overline{\Delta}_{rq} \lesssim (m + 3)\epsilon + \rho S.$$

To prove the theorem, we therefore need only show that

$$|\Delta_{pq} + \Delta_{qp}| \lesssim 2\epsilon + \rho S.$$

We write

$$\begin{aligned} &|c_p^{(i)}(T_0 + \Delta_{qp}) - c_p^{(i)}(T_0 - \Delta_{pq})| \\ &\quad \leq |c_p^{(i)}(T_0 + \Delta_{qp}) - c_q^{(i)}(T_0)| + |c_q^{(i)}(T_0) - c_p^{(i)}(T_0 - \Delta_{pq})| \\ &\quad \lesssim \epsilon + \epsilon + \rho S \qquad \text{[by A2 and Lemma 3]} \end{aligned}$$

where $T_0$ is as in A2, and the result follows from A1. □

Combining Theorem 4 with Theorem 2 gives an interactive consistency algorithm with

$$\delta \approx (m + 6)\epsilon + 3\rho S + \rho R,$$
$$\Sigma \approx (2m + 12)\epsilon + 5\rho S + \rho R.$$

This is the value of $\delta$ that should be used in the definition of on-time arrival.

## 6. *Conclusion*

We have described three clock synchronization algorithms. The interactive convergence Algorithm CNV is the simplest, requiring only that every process read every other process' clock. The interactive consistency algorithms are more complex, requiring a great deal of message passing. These algorithms are based upon two Byzantine Generals solutions from [6]. A number of different Byzantine Generals solutions have been proposed; a survey of them can be found in [8]. The solutions we have used as the basis for our clock-synchronization algorithms are optimal in the sense that they require the fewest "rounds" of message passing—$m + 1$ rounds being required to handle $m$ faults. Since each round adds an $O(\epsilon)$ term to the synchronization error, minimizing the number of rounds is a reasonable criterion for choosing an algorithm.

In addition to minimizing the number of rounds, one also wants to reduce the number of messages generated. Algorithm COM generates approximately $n^{m+1}$ messages. All Byzantine Generals solutions we know of that generate fewer messages either use more rounds or require digital signatures. Fortunately, in process-control applications, $n$ and $m$ tend to be small enough so that $n^{m+1}$ is not an unreasonable number of messages. However, there may be other applications in which one would be willing to use more rounds in order to generate fewer messages. While there are Byzantine Generals solutions that do this, it is not clear how they can be converted to clock-synchronization algorithms. Our method of deriving Algorithm COM depended upon the specific details of Algorithm OM. We have not tried to derive clock-synchronization algorithms from the other Byzantine Generals solutions.

The situation is different if we allow digital signatures. Algorithm CSM generates almost as many messages as Algorithm COM. However, it is possible to reduce the number of messages. In [2], Dolev and Strong reduced the number of messages generated by Algorithm SM to $2n^2$ by simply eliminating redundant messages—for example, a process never sends the same value twice to the same process. In our intuitive description of Algorithm CSM, we can reduce the number of "clocks" sent by having each process $p$ obey the following two rules:

(1) $p$ never sends a clock if it has already sent a faster clock.
(2) $p$ never sends a clock that is approximately the same as one it has already sent.

In view of the improved algorithm of [3], which is not based upon a Byzantine Generals solution, there seems little point in investigating these improvements to Algorithm CSM.

To compare the closeness of synchronization achieved by these algorithms, we assume that $\rho S \ll \epsilon$. This is a reasonable assumption, since, for most practical applications, $\epsilon$ will be on the order of microseconds, $S$ at most a few milliseconds, and $\rho \lesssim 10^{-6}$. To simplify comparisons with the interactive convergence algorithm, in which $\delta$ depends upon $n$, we assume that $n = 3m + 1$. This will be the case if the only reason for having multiple processes is to achieve fault-tolerance through

redundancy. We then get the following values of $\delta$:

Algorithm CNV: $(6m + 2)\epsilon + (3m + 1)\rho R$,
Algorithm COM: $(6m + 4)\epsilon + \rho R$,
Algorithm SCM: $(m + 6)\epsilon + \rho R$.

We have proved only that the synchronization errors of the algorithms are less than these quantities; we do not know if the errors can really become this large. However, for want of an alternative, we use these bounds in comparing the algorithms.

From these numbers, Algorithm CSM appears to be superior. However, this is misleading because the $\epsilon$ for Algorithm CSM is not necessarily the same as the $\epsilon$ of the other two algorithms, since it may come from a very different way of reading the clocks. The Algorithms CNV and COM can use any method of reading clocks, but clock reading in Algorithm CSM requires measuring the arrival times of messages and knowing the delay in processing and sending a message. For the tightly-coupled multiprocessors typical of process-control applications, we believe that $\epsilon$ is likely to be much larger for CSM than for COM. In this case, Algorithm CSM is to be preferred only because it requires fewer processes to achieve the same degree of fault-tolerance.

The algorithm of [3] reduces the term $(m + 6)\epsilon$ of Algorithm CSM to $\epsilon$. However, that algorithm involves the same form of clock reading as Algorithm CSM, so its $\epsilon$ may be larger than that of our other two algorithms. For process-control applications, Algorithms CNV and COM may provide closer synchronization than any algorithm requiring digital signatures.

Since Algorithms CNV and COM can use the same method of clock reading, the above error bounds provide a meaningful comparison for them. If $R$ is small enough—that is, if the clocks are resynchronized often enough—then Algorithm CNV can achieve slightly better synchronization than Algorithm COM. However, one usually wants to resynchronize only as often as is necessary to achieve a desired value of $\delta$. If this value of $\delta$ is much larger than $6m\epsilon$, then it is necessary to synchronize $3m + 1$ times as often with Algorithm CNV than with the interactive consistency algorithms. For example, if $m = 2$, $\epsilon = 2$ microseconds, $\rho = 10^{-6}$, and $\delta = 50$ microseconds—values that are reasonable for process-control systems in which one process can directly read another's clock—we obtain the following resynchronization intervals $R$:

Algorithm CNV: 3.1 seconds,
Algorithm COM: 18 seconds.

We suspect that in most applications, Algorithm CNV will provide sufficiently short resynchronization times.

We have assumed a system in which each process can communicate with all the others, and have considered only process failure, not communication failure. Our interactive consistency algorithms can be generalized to incompletely connected networks of processes. In the same way that Algorithm COM was derived from Algorithm OM($m$), Algorithm OM($m$, $p$) of [6] and the algorithm of [1] can be used to obtain clock synchronization algorithms for incompletely connected networks. Algorithm CSM also works, with obvious modifications, in the more general case. It can be shown that for a network of diameter $d$, Algorithm CSM($m$) satisfies Theorem 4 except with the maximum number of faulty processes reduced to $m - d + 1$.

In algorithms not using digital signatures, the failure of a communication line joining two processes must be considered a failure of one of the two processes. Indeed, a two-faced clock is perhaps more likely to be caused by communication failure than by failure of the clock itself. For Algorithm CSM, assuming that a faulty communication line cannot "forge" properly signed messages, a faulty communication line is equivalent to a missing one. Hence, Algorithm CSM($m + d - 1$) can handle up to $m$ process faults plus any number of communication line failures, so long as the remaining network of nonfaulty processes and communication lines has diameter at most $d$.

*Glossary*

- $\delta$ The maximum error in clock synchronization—Clock Synchronization Condition S1.
- $\delta_0$ The maximum initial difference between the values of different processes' clocks—A0.
- $\epsilon$ The maximum error in reading clocks—A2.
- $\gamma$ The "normal" message-transmission time—A2′.
- $\rho$ The rate at which nonfaulty clocks can drift apart—Definition 1.
- $\Delta_{qp}$ The difference between process $q$'s clock and process $p$'s clock, as read by $p$.
- $\Sigma$ The maximum amount by which a clock is advanced during resynchronization—Clock Synchronization Condition S2.
- $\Omega$ Defined in Condition CC.
- $m$ The maximum number of faulty processes.
- $n$ The total number of processes.
- $R$ The length of a synchronization interval—that is, the time between successive clock resynchronizations.
- $R^{(i)}$ The $i$th synchronization interval.
- $S^{(i)}$ The period at the end of the $i$th synchronization interval during which the resynchronization algorithm is executed.
- $T^{(i)}$ Ending time of the $i$th synchronization period.

ACKNOWLEDGMENTS. We wish to thank our fellow members of the SRI Computer Science Laboratory, especially Robert Shostak, for their assistance and encouragement, and Nancy Lynch for simplifying the proof of Lemma 6 and pointing out many flaws in an earlier version.

REFERENCES

1. DOLEV, D. The Byzantine Generals strike again. *J. Algor. 3*, 1 (1982), 14–30.
2. DOLEV, D., AND STRONG, R. Authenticated algorithms for Byzantine Agreement. *SIAM. J. 12*, 4 (Nov. 1983), 656–666.
3. HALPERN, J., SIMONS, B., AND STRONG, R. An efficient fault-tolerant algorithm for clock synchronization. IBM Tech. Rep. RJ-4094, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1983.
4. LAMPORT, L. The implementation of reliable distributed multiprocess systems. *Comput. Netw. 2* (1978), 95–114.
5. LAMPORT, L. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Prog. Lang. Syst.*, to appear.
6. LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Byzantine Generals problem. *ACM Trans. Prog. Lang. Syst. 4*, 3 (July 1982), 382–401.
7. PEASE, M., SHOSTAK, R., AND LAMPORT, L. Reaching agreement in the presence of faults. *J. ACM 27*, 2 (Apr. 1980), 228–234.
8. STRONG, H. R., AND DOLEV, D. Byzantine Agreement. In *Intellectual Leverage for the Information Society (Compcon)*. New York: IEEE Computer Society Press, pp. 77–82.

9. DOLEV, D., HALPERN, J. Y., AND STRONG, H. R. On the possibility and impossibility of achieving clock synchronization. In *Proceedings of 16th Annual ACM Symposium on Theory of Computing* (Washington, D.C., Apr. 30–May 2). ACM, New York, 1984, pp. 504–511.
10. WENSLEY, J., ET AL. SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE 66,* 10 (Oct. 1978).

RECEIVED JULY 1981; REVISED MARCH 1982, FEBRUARY 1984, AND JULY 1984; ACCEPTED AUGUST 1984