18

Synchronizing Time Servers

by Leslie Lamport

June 1, 1987

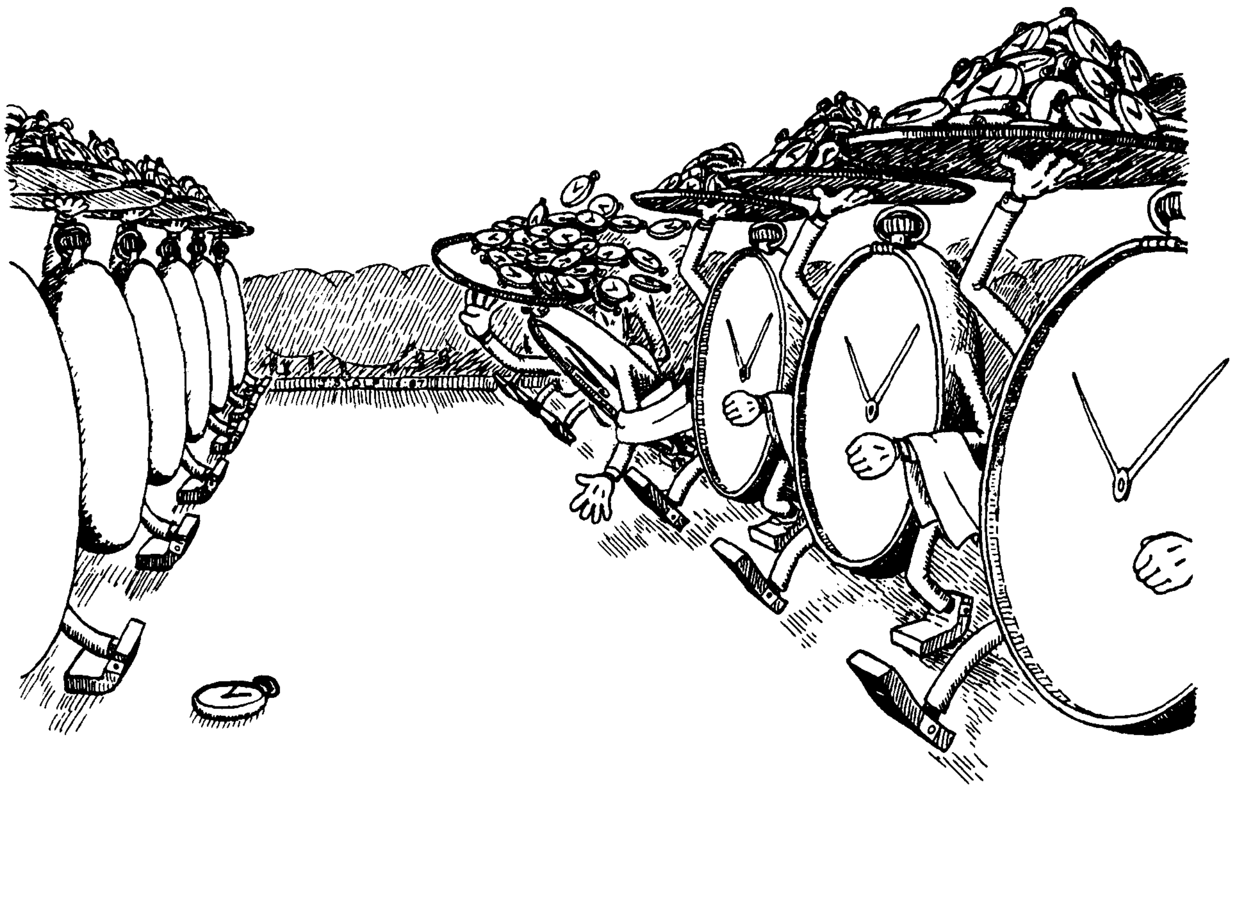
digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

Synchronizing Time Servers

Leslie Lamport

June 1, 1987



Author's Abstract

A time service in a distributed system may be used both for multiprocess synchronization and for simply finding out what time it is. For synchronization, the time provided by different servers should be closely synchronized. For telling time, the time provided by each server should be a close approximation to Universal Time (the international time standard). Algorithms are presented for implementing a fault-tolerant time service that meets both requirements.

Capsule Review

Users of electronic mail are not surprised to see messages that are time-stamped *after* they were received. The naive blame the probably non-existent operator, "who didn't get the time right." The wary know that many computer systems go for long periods without stopping; during those periods adjustments are difficult or impossible because almost all time-dependent processes (for example, file systems, mailers, audit trails, back-up mechanisms, etc.) assume that the time provided by the local operating system is increasing smoothly. Hence a good time service not only must be close to the real time, but also must increase and maintain only a bounded rate of change.

For some applications (for example, distributed data bases), there is a third requirement: the times maintained by various processors within a network must be very close to each other. This requirement is so stringent that it is not enough simply to ensure that the time provided by each processor is within a certain limit of the real time. Resynchronizations with the real time must therefore be coordinated.

The final requirement of a good time service is that the resynchronization protocol must allow a certain number of faulty links or faulty processors, since network protocols ought to work in the presence of partial failures.

Previous authors have presented algorithms that satisfy some of these requirements, but the algorithm described here is the first that satisfies all four simultaneously. Another attribute of the paper is that both the problem and the solution are precisely formulated.

Andrei Broder

Contents

1 Introduction

A time service provides the "current time" to its users. It performs two functions:

- Telling a user the current time and date.
- Allowing different users to synchronize their activities.

Though related, these two functions are distinct. The first requires that the time provided be approximately equal to *Universal Time*—the ideal standard that is approximated by the National Bureau of Standards' broadcasts on station WWV. The second requires that the time provided to different users be approximately the same.

Marzullo [6] devised algorithms for providing an accurate time and date, and a number of fault-tolerant synchronization algorithms have been proposed [2,4,5], but there has apparently been no previous work that considered both functions at the same time. In this paper, I consider the problem of implementing a fault-tolerant time service that provides a single time value to perform both functions—more precisely, a time value that permits close to optimum synchronization and is a reasonable approximation to the correct time and date.

The problem to be solved is stated somewhat informally in this introduction. Section 2 states the assumptions and conditions more precisely and defines some helpful notation. (A glossary is provided at the end of the paper to help the reader follow the notation.) Section 3 describes Marzullo's algorithm for computing the best possible approximation to the correct time and date, and Section 4 develops an algorithm for a time service that provides the two functions.

The algorithms assume a network of processes, in which each node has a local clock that runs at approximately the correct rate, and some nodes also have direct access to Universal Time, perhaps obtained by "listening" to WWV. A time server is implemented by synchronizing all the nodes' clocks, using the available information about Universal Time.

The nodes providing the time service may be a subset of the nodes in the complete system—the other nodes interrogating the time servers to obtain time information. However, nodes that do not act as time servers or providers of Universal Time are ignored. The time service must function properly despite the failure of some network components.

A time service could provide two distinct values to satisfy its two different functions. For the function of providing the current time and date, a process

p would provide a time interval I_p that is its best approximation to Universal Time. More precisely, I_p would be the smallest interval that p knows to contain UT, the current Universal Time. (It is most common to write a value with error bound in the form $t \pm \epsilon$. However, it is more convenient to work with the interval $[t - \epsilon, t + \epsilon]$ within which the correct value is known to lie.)

For the second time-service function, p must provide a time T_p, called *service time*, that is close to the service time T_q provided by any other node q. More precisely, it should provide T_p and, for each other node q, a number δ_{pq} such that $|T_p - T_q| < \delta_{pq}$. However, this condition is not sufficient, since it is trivially met by letting each T_p always be zero. To represent time, T_p should change at approximately the same rate as UT, so the value of T_p increases by about one second with the passage of one second of Universal Time.

While not strictly necessary, it is convenient to have T_p not only change at approximately the same rate as UT, but be approximately equal to it. Consider, for example, the problem of generating creation times for files. One might want to use the creation time to decide which version of a file is the current version. Since versions may be created at different nodes, a file generated at node p should use T_p as its creation time to minimize the likelihood that a version created at one node receives an earlier creation time than a version created before it at a different node. However, one might also want a creation time to tell the Universal Time at which the file was created, so the user can determine the actual date and time of creation. This could be accomplished by recording a separate "universal creation time" derived from I_p. However, this additional value is not needed if T_p provides an acceptable approximation to UT.

We therefore state the following three requirements for the time T_p provided by node p, where κ_p, ϵ_p, and the δ_{pq} are values provided by p. (They could be constants that are announced when the system is "turned on", or they could be provided in response to user requests.)

correct rate The rate of change of T_p with respect to UT lies between $1 - \kappa_p$ and $1 + \kappa_p$.

synchronization For every other node q: $|T_q - T_p| < \delta_{pq}$.

correct time $|UT - T_p| < \epsilon_p$.

The synchronization requirement follows from the correct-time requirement by letting $\delta_{pq} = \epsilon_p + \epsilon_q$. However, this may not provide close enough synchronization. Universal Time is of interest mainly to humans; synchronization algorithms depend only upon the differences between the values of T_p at the different nodes. Humans seldom need to know the value of UT to better than a few seconds, so an ϵ_p of several seconds is acceptable. On the other hand, synchronization algorithms may have more stringent requirements for δ_{pq}. When nodes p and q are using the time service for synchronization, p generally incurs a delay of $O(\delta_{pq})$ seconds because of the lack of synchrony of T_p and T_q. For example, if q announces that it will release a resource at time t (that is, when T_q equals t), then p must wait until T_p reaches $t + \delta_{pq}$ before acquiring the resource. Some applications might require that this delay be kept to within a few microseconds.

While it is not necessary to keep the ϵ_p as small as the δ_{pq}, the required synchronization condition can be achieved if ϵ_p could be kept small enough. However, this is not always possible. The closeness with which clocks at different nodes can be directly synchronized depends upon the uncertainty in message transmission time between those nodes. Modern large networks are heterogeneous, and the uncertainty in transmission times may be very different for different pairs of nodes. Typically, a large network consists of a collection of local area networks (LANs) that are interconnected by point-to-point links. Nodes on a single LAN may be directly connected by a fiber-optic link, in which the uncertainty in transmission time can be as small as a few microseconds if the timing functions are performed at a low enough system level. The nodes in different LANs may communicate with one another by a store-and-forward protocol that could have an uncertainty of a second or more in transmission time.[1] If a particular LAN does not include a direct source of Universal Time (such as a WWV receiver), so nodes in the LAN must base their knowledge of UT on messages received from outside the LAN, then the values of ϵ_p and ϵ_q for p and q in the LAN could be several orders of magnitude greater than the best achievable value of δ_{pq}.

Marzullo [6] presented algorithms for obtaining a clock value from a set of clocks, some of which may be faulty. These algorithms can be used to provide the intervals I_p that best approximate Universal Time, but they do not satisfy the synchronization condition if $\delta_{pq} < \epsilon_p + \epsilon_q$. Several Byzantine

[1]Such a large value results not from uncertainty in the physical transmission times, but because the communication involves higher-level protocols, separated from the physical messages by many layers of software. It is quite likely that the timing of transmission delays can be done at a lower system level for intra-LAN messages than for inter-LAN messages.

clock synchronization algorithms have been presented that can be used to satisfy the correct-rate and synchronization conditions in the presence of failures [2,4,5]. However, to my knowledge, there have been no published algorithms to achieve all three of the conditions above.

The major part of this report concerns algorithms for achieving the synchronization condition when δ_{pq} may be much smaller than $\epsilon_p + \epsilon_q$. This is a nontrivial problem in the presence of failures, because even very simple kinds of failure act like malicious, "Byzantine" failures. For example, suppose a node is sending its clock value to all other nodes in the network. It does this by sending a message saying something like "my clock now reads 11:47". Suppose, through some hardware or software error, it pauses for five minutes in the middle of this broadcast. While it sends the same message to all nodes, it has essentially sent descriptions of two clocks that differ by five minutes. Thus, this simple error results in a "two-faced" clock that provides different clock values to different nodes.

The correct-time requirement does not mention the interval I_p, which represents p's knowledge of UT. One might be tempted to replace this requirement by the condition that T_p be in I_p. However, such a condition would be inconsistent with the other two requirements for T_p. It is inconsistent with the correct-rate requirement because new knowledge of the correct value of UT, such as the receipt of a message from a node with a WWV receiver, could suddenly reduce the width of the interval I_p. Keeping the value of T_p within the interval I_p could require a sudden change to T_p, which is prohibited by the correct-rate requirement. It can also be shown that, with malicious failures, requiring T_p to be within I_p could require violating the synchronization requirement if $\delta_{pq} < \epsilon_p + \epsilon_q$.

2 Notation and Assumptions

In the introduction, the term *node* was used to emphasize that each node in the network provides a time service for user processes running at that node. To be consistent with the terminology commonly used in discussing clock synchronization, the term *process* will be used instead of *node*.

2.1 Intervals

The term *interval* is used to denote a closed interval on the real line—that is, an interval of the form $[x, y]$ for $x \leq y$. The width of the interval R is denoted by $||R||$, so $||[x, y]|| = y - x$. The sum of two intervals is defined by

$[x,y]+[z,w]=[x+z,y+w]$. A real number z is considered to be the same as the interval $[z,z]$, so $[x,y]+z$ is defined to be the interval $[x+z,y+z]$ obtained by translating the interval $[x,y]$ to the right a distance of z. For any interval U and number $\delta > 0$, $U \pm \delta$ is defined to equal $U+[-\delta,\delta]$, so $[x,y]\pm\delta=[x-\delta,y+\delta]$.

A *pseudo-metric* d on intervals is a nonnegative, real-valued function on pairs of intervals satisfying the following properties for all intervals U, V, and W: (i) $d(U,V)=d(V,U)$, (ii) $d(U,V)+d(V,W)\geq d(U,W)$, and (iii) $d(U,U+\epsilon)=\epsilon$ for any $\epsilon \geq 0$. A pseudo-metric satisfying the additional property that $d(U,V)=0$ implies $U=V$ is called a *metric.* (A metric is sometimes called a *distance function.*)

Two important pseudo-metrics are

- The *midpoint* pseudo-metric d_m, where $d_m(U,V)$ equals the distance between the midpoints of U and V.
- The *uniform* metric d_u, where $d_u([x,y],[v,w])$ equals the maximum of $|v-x|$ and $|w-y|$.

As its name implies, the uniform metric is a metric. Note that for any intervals U and V, $d_m(U,V)\leq d_u(U,V)$.

A real-valued function F on m-tuples of intervals is said to satisfy the *Lipschitz condition* for a pseudo-metric d if, for any intervals U_i and V_i and any number $\delta > 0$: $d(U_i,V_i)<\delta$ for all i implies $|F(U_1,\ldots,U_m)-F(V_1,\ldots,V_m)|<\delta$. The function F is said to be *translation invariant* if $F(U_1+x,\ldots,U_m+x)=F(U_1,\ldots,U_m)+x$ for any intervals U_i and real number x. Satisfying the Lipschitz condition is a stronger requirement than continuity and a weaker requirement than having a bounded derivative. Translation invariance asserts that translating all arguments by a fixed amount causes the value to be translated by the same amount. We expect any functions appearing in a clock synchronization algorithm to be translation invariant, since increasing all input clock values by a fixed amount should produce a corresponding increase in the clock values computed by the algorithm. Observe that if F satisfies the Lipschitz condition for the pseudo-metric d_m, then it also satisfies the condition for the metric d_u.

2.2 Clocks and Clock Ranges

A *time-dependent value* is any real-valued function of a real variable. If v is a time-dependent value, we interpret $v(t)$ to be the value of v at Universal

Time t. A *clock* is a nondecreasing time-dependent value. If V is a clock, the value $V(t)$ represents the value read by clock V at Universal Time t. The identity function, denoted by UT (so $UT(t) = t$), is a clock. The service time T_p provided by process p is a clock, where $T_p(t)$ represents the time value provided by p to a request for the current service time received at Universal Time t. (Of course, p does not need to know the current value of Universal Time to compute the value of T_p at that time.)

Let $V_1, \ldots, V_n$ be clocks. (Think of V_p as a clock maintained by process p.) The following definitions express the correctness conditions introduced informally in the introduction. They describe conditions on these clocks for an interval of time $[u, v]$, where u and v represent clock values—that is, times indicated by the clocks themselves. Thus, the conditions express properties of the clocks over intervals $[s, t]$ of Universal Time such that $V_p(s) = u$ and $V_p(t) = v$ for a process p. The conditions are expressed in this form because clock times are directly observable, Universal Times are not. The bounds κ_p, δ_{pq}, and ϵ_p are time-dependent values. (In many cases, they will be constants.)

correct rate with bounds κ_p: For each p and any x and y, $x \neq y$ such that $V_p(x)$ and $V_p(y)$ are in $[u, v]$:

$$\int_x^y (1 - \kappa_p(t))\, dt < V_p(y) - V_p(x) < \int_x^y (1 + \kappa_p(t))\, dt$$

synchronization with bounds δ_{pq}: For each p and q, $p \neq q$, and each t such that $V_p(t)$ is in $[u, v]$: $|V_q(t) - V_p(t)| < \delta_{pq}(t)$.

correct time with bounds ϵ_p: For each p and each t such that $V_p(t)$ is in $[u, v]$: $|UT(t) - V_p(t)| < \epsilon_p(t)$.

The correct-rate condition is defined in terms of integrals to avoid requiring that the V_p be differentiable functions of time. When no interval is specified, these conditions are assumed to hold for all intervals.

Each nonfaulty process p is assumed to have a clock C_p, called its *local* clock.[2] It is assumed that the local clocks of all nonfaulty processes satisfy the correct rate condition with bounds $\rho_p \ll 1$, where the ρ_p are constants. (An error in the local clock C_p is considered to be a failure of process p.)

A *p-clock* is a clock of the form $C_p + v$ for some constant v—that is, a clock whose value at time t is $v + C_p(t)$. A p-clock is one that runs at the same

[2]It is sufficient for p to have a cyclic timer, since one can construct a monotonic clock from such a timer. In fact, the algorithms are easily modified to work with only a timer.

rate as p's local clock. Of course, C_p is a p-clock. A p-clock V_p is determined by its value at any single time t_0, since $V_p(t) = V_p(t_0) - C_p(t_0) + C_p(t)$. Note that T_p, the time-service clock provided by p, will not in general be a p-clock.

The following result is an easy consequence of the assumption that C_p satisfies the correct-rate condition. It asserts that the uncertainty ρ_p in the running rate of p's local clock causes its knowledge of Universal Time to degrade at a rate of ρ_p seconds per second of elapsed time on its local clock.

Proposition 1 *If process p is nonfaulty and R is an interval such that $UT(t_0) \in R$, then for all $t \geq t_0$:*

$$UT(t) \in R + (1 \pm \rho_p)(C_p(t) - C_p(t_0))$$

where terms of order $\rho_p^2(t - t_0)$ are neglected.

A *clock range* is an interval-valued function on the reals of the form $[x, y]$ where x and y are clocks. In other words, R is a clock range if there exist clocks x and y such that $R(t) = [x(t), y(t)]$ for all times t. A ***p-clock range*** is a clock range of the form $[x, y]$ such that x and y are p-clocks. A p-clock range can be written as $U + C_p$ for some interval U. Since a real number x is identified with the interval $[x, x]$, a clock is a special case of a clock range, and a p-clock is a special case of a p-clock range. A p-clock range is determined by its value at any single time.

If F is a real-valued function on m-tuples of intervals, then applying F to an m-tuple of clock ranges produces a time-dependent value. Let $R_1, \ldots, R_m$ be p-clock ranges with $R_i = U_i + C_p$ for intervals U_i. If F is translation invariant, then

$$F(R_1, \ldots, R_m) = F(U_1, \ldots, U_m) + C_p$$

Thus, if the R_i are p-clock ranges, then $F(R_1, \ldots, R_m)$ is a p-clock.

2.3 The Network

I assume a network of processes connected by *channels*, where a channel may connect more than two processes. The two kinds of channels that are of interest are a *point-to-point* channel that has a single sender and a single receiver, and a *broadcast* channel that connects a set of processes so that any one of them can broadcast a message over it to all other processes on the channel. A two-way communication line is a broadcast channel that connects just two processes.

Certain of the processes, called *Universal Time providers*, are assumed to have a direct source of Universal Time. Let $UT^{(j)}$ denote the value of UT obtained by process j. This value is a clock range that is known to contain the correct value of Universal Time, so $UT(t) \in UT^{(j)}(t)$ for any time t. Process j will periodically broadcast $UT^{(j)}$ to other processes.

For each channel c, I assume values τ^c_{min} and τ^c_{max} such that a message sent over c at time t is received between times $t + \tau^c_{\text{min}}$ and $t + \tau^c_{\text{max}}$ if the sender, the receiver, and c are all nonfaulty. More precisely, if an event, such as the receipt of another message, that occurs at the sender at time t causes the sending of a message M over channel c, then M will be received between times $t + \tau^c_{\text{min}}$ and $t + \tau^c_{\text{max}}$. Thus, the minimum and maximum delays τ^c_{min} and τ^c_{max} include the time needed to generate and send the message as well as the time the message was actually in transit along channel c. The values of τ^c_{min} and τ^c_{max} may vary with time, but they are assumed to be known to the receiver. Let γ^c denote $\tau^c_{\text{max}} - \tau^c_{\text{min}}$.

Delivering a message with a delay less than τ^c_{min} or greater than τ^c_{max} constitutes a channel failure. If there is unpredictable variance in transmission delay, due, for example, to variation in the channel loading, then τ^c_{min} and τ^c_{max} should be chosen conservatively to reduce the probability of such a failure. (Note that τ^c_{min} can always be taken to be zero.) However, the time required for fault-tolerant synchronization algorithms depends upon the values of τ^c_{max}, not on the actual delays, and the bounds δ_{pq} on clock synchronization depend upon γ^c, so tradeoffs between reliability and efficiency must be made when choosing the values of τ^c_{min} and τ^c_{max}.

A ***path*** is a sequence of processes and channels, each channel connecting successive processes. The ***null path*** connects process p to itself. If π is a path from p to q and ψ is a path from q to r, then $\pi\psi$ denotes the obvious path from p to r via q.

For a path π from p to q, let τ^π_{min} and τ^π_{max} denote the sum of all τ^c_{min} and τ^c_{max}, respectively, for all channels c in π. Thus, τ^π_{min} and τ^π_{max} represent the minimum and maximum transmission delays for a message relayed from p to q along π. Define γ^π to be $\tau^\pi_{\text{max}} - \tau^\pi_{\text{min}}$, the uncertainty in transmission delay along π.

Fault-tolerant synchronization algorithms require that a process know the values τ^π_{min}, τ^π_{max}, and γ^π for messages it receives over the path π, which usually requires knowledge of the values of τ^c_{min} and τ^c_{max} for each channel c in the path. If these values can change, then new values can be broadcast using the method of [3], which assures that the same values are used by all processes. For simplicity, I assume that the $\tau^{\pi}_{\min}$, $\tau^{\pi}_{\max}$, and γ^{π} are constants for each fixed path π.

If R is an interval and π a path, then R^{π} is defined to be the interval $R+[\tau^{\pi}_{\min}, \tau^{\pi}_{\max}]$. Suppose that π is a nonfaulty path from process p to process q, and R represents p's knowledge of Universal Time at a certain time t—that is, p knows that $UT(t) \in R$. If p sends a message with the value R along π to q and that message arrives at time t', then since the transmission time of the message is in the interval $[\tau^{\pi}_{\min}, \tau^{\pi}_{\max}]$, q knows that $UT(t') \in R^{\pi}$. Observe that if $\pi\phi$ is a path, then $R^{\pi\phi} = (R^{\pi})^{\phi}$.

Synchronization algorithms require processes to send clock ranges to one another. (Remember that a clock is a special case of a clock range.) A process p sends a p-clock range by sending a message with the clock range's current value R along a path π. The receiving process q interprets this message as the receipt of a q-clock range whose value, at the time the message is received, is R^{π}. The following result asserts that transmitting a clock range in this way causes an initial perturbation by a distance of up to γ^{π}, after which the two clock ranges drift apart at a rate of at most $\rho_p + \rho_q$, where distance is measured by the uniform metric d_u on intervals. This result is a simple consequence of the correct-rate assumption for the local clocks and the assumed bounds on message-transmission times.

Proposition 2 *Let R_p be a p-clock range, and suppose that process p sends a message at time t that is received at time t′ over a nonfaulty path q, and let R_q be the q-clock range such that $R_q(t') = R_p(t)^{\pi}$. Then for any $\Delta t \geq 0$:*

$$d_u(R_p(t+\Delta t), R_q(t+\Delta t)) \leq \gamma^{\pi} + (\rho_p + \rho_q)\Delta t$$

where terms of order $\rho_q \tau^{\pi}_{\max}$ are neglected.

3 Obtaining Universal Time

Let us now consider how a time server p could provide the best possible value of I_p, an interval known to contain UT. Assume that each Universal Time provider j maintains a clock range $UT^{(j)}$ that represents its current knowledge of Universal Time. If j is nonfaulty, then $UT(t) \in UT^{(j)}(t)$ for all times t. At various times, provider j broadcasts the current value of $UT^{(j)}$. Let $UT^{(j)}_p$ denote a clock range that represents p's knowledge of the current value of $UT^{(j)}$. More precisely, assume that, if j and p are nonfaulty, then

$UT(t)$ lies within the interval $UT_p^{(j)}(t)$ for all times t. If p receives a message at time t_0 informing it that $UT(t_0)$ lies in the interval R, then Proposition 1 implies that we can define $UT_p^{(j)}$ by

$$UT_p^{(j)}(t) = R + (1 \pm \rho_p)(C_p(t) - C_p(t_0))$$

for $t \geq t_0$. In fact, this is how $UT_p^{(j)}(t)$ should be defined if p did not receive any information about $UT^{(j)}$ during the time interval $[t_0, t]$. The problem of how j broadcasts the value $UT^{(j)}$ is considered later.

The algorithm for computing the best approximation to UT from a set of m intervals, each asserted to contain UT, was first obtained by Marzullo. It appears as Algorithm 4-2 in his thesis [6]. Define $\mathcal{M}_m^f(U_1, \ldots, U_m)$ to be the largest interval whose endpoints belong to at least $m - f$ of the intervals U_i. It is not hard to show that, if one knows only that UT lies in all but at most f of the intervals U_j, then $\mathcal{M}_m^f(U_1, \ldots, U_m)$ is the smallest interval known to contain UT. The value of $\mathcal{M}_m^f(U_1, \ldots, U_m)$ can be computed from the set of m intervals U_i in $O(m \log m)$ time by sorting their endpoints. It can be recomputed in $O(m)$ time if just one of the U_i changes.

If the intersection of U_1 with $\mathcal{M}_m^f(U_1, \ldots, U_m)$ is empty, then $\mathcal{M}_m^f(U_1, \ldots, U_m) = \mathcal{M}_{m-1}^{f-1}(U_2, \ldots, U_m)$. In this case, U_1 is known to be one of the intervals that does not contain UT (there are at most f of them), so it may be thrown away when computing the best approximation to UT. After throwing it away, we are left with $m - 1$ intervals, all but at most $f - 1$ of them containing UT. More generally, if k of the U_i have an empty intersection with $\mathcal{M}_m^f(U_1, \ldots, U_m)$, then $\mathcal{M}_m^f(U_1, \ldots, U_m)$ equals the value obtained by throwing away those k intervals and applying $\mathcal{M}_{m-k}^{f-k}$ to the remaining intervals.

Assume that up to f of the m values $UT_p^{(j)}$ may be incorrect, where an interval $UT_p^{(j)}$ is correct if UT always lies within it. The obvious way to choose I_p, an interval that process p knows to contain UT, is to let it equal $\mathcal{M}_m^f(UT_p^{(1)}, \ldots, UT_p^{(m)})$. However, suppose that at some time when C_p has the value C, $\mathcal{M}_m^f(UT_p^{(1)}, \ldots, UT_p^{(m)})$ equals the interval U. When C_p has the value $C + \Delta C$, UT must lie in the interval $U + (1 \pm \rho_p)\Delta C$. However, the intervals $UT_p^{(j)}$ are spreading out at a rate ρ_p, which could cause the value of $\mathcal{M}$ to spread out at a faster rate—in fact, to make large, discontinuous jumps. Thus, when C_p has the value $C + \Delta C$, $\mathcal{M}(UT_p^{(1)}, \ldots, UT_p^{(m)})$ could be a larger interval than $U + (1 \pm \rho_p)\Delta C$.

In Marzullo's algorithm, p computes an initial value $I_p(t_0)$ from initial values $UT_p^{(j)}(t_0)$ of the $UT_p^{(j)}$ as follows. It throws away any of the $UT_p^{(j)}(t_0)$ that it decides are incorrect. If $m-k$ intervals are currently believed to be correct, then the interval $I_p(t_0)$ equals $\mathcal{M}_{m-k}^{f-k}$ applied to the $m-k$ correct intervals $UT_p^{(j)}(t_0)$. If no new values are received from the Universal Time providers between times t_0 and t, then, ignoring terms of order $\rho_p^2(t-t_0)$, $I_p(t)$ is defined by

$$I_p(t) = I_p(t_0) + (1 \pm \rho_p)(C_p(t) - C_p(t_0))$$

In other words, when p receives no new information, the interval I_p advances (moves right along the number line) at p's clock rate and widens at the rate of $2\rho_p$ seconds per second of clock time.

When a new value for $UT_p^{(j)}$ arrives, process p adds the value to the set of intervals that it presumes to be correct, throwing away the previous value of $UT_p^{(j)}$. Process p next computes I_p by applying $\mathcal{M}_{m-k}^{f-k}$ to the $m-k$ intervals currently presumed correct. It then declares to be incorrect any of these intervals (the ones it had presumed to be correct) that have empty intersections with I_p.

When no new information is received from the Universal Time providers, the value of I_p "deteriorates" at the rate of ρ_p seconds per second. To maintain the accuracy of I_p, it is necessary for the Universal Time providers to broadcast their values of $UT^{(j)}$ sufficiently often. Suppose that a provider j sends its clock range $UT^{(j)}$ to p at time t by simply sending a message with the interval $UT^{(j)}(t)$ along some path π. If p receives this message at time t', it sets $UT_p^{(j)}(t')$ to $UT^{(j)}(t)^{\pi}$. Suppose that each provider j sends $UT^{(j)}$ in this way at least once every J seconds over a path π with $\gamma^{\pi} \leq \gamma$. It is then easy to show that if $\| UT^{(j)} \| < \epsilon$ for every nonfaulty provider j and at least $m-f$ of the time providers and their paths π are nonfaulty, then Marzullo's algorithm guarantees that for all times t, $I_p(t)$ is contained within the interval $UT(t) \pm (\epsilon + \gamma + \rho_p J)$.

4 Providing the Service Time

4.1 Ideal Time

A time server p periodically receives information allowing it to refine the clock T_p that represents the service time it provides to its clients. Information comes in discrete lumps—usually through the receipt of a message. To maintain the continuity of T_p—more precisely, to maintain the boundedness of its rate of change—the value of T_p cannot change instantly in response to new information. Instead, the rate of change of T_p is modified discontinuously. This section presents a method for computing T_p's rate of change.

Process p computes T_p from its clock C_p by the formula:

$$T_p = a_p C_p + b_p$$

where a_p and b_p are constants that are changed discontinuously—that is, they are piecewise constant functions of time. The value b_p represents a zero-point correction;[3] the value a_p represents a correction to the running rate of C_p. I will discuss the way a_p changes; the change to b_p is determined by the requirement that T_p be continuous and will be ignored.

There are two corrections embodied in the value of a_p: a correction to compensate for the measured inaccuracy of the local clock C_p, and a correction to bring T_p into synchrony with UT and with the times T_q provided by other servers q. The component of a_p that compensates for the inaccuracy of C_p effectively reduces the error ρ_p in its rate. It can be obtained by comparing C_p with I_p for a long enough period of time. I will ignore this component and assume that any difference between a_p and 1 is meant as a correction to achieve synchronization. Such a correction actually increases the error κ_p in the rate of change of T_p. This increase is unavoidable. If clock synchronization is to be maintained, κ_p must be allowed to become larger than ρ_p, the inherent error in the rate of p's local clock.

It is easiest to describe synchronization algorithms in terms of discontinuously resetting the time. There are a sequence of *resynchronization times* $T^{(0)}$, $T^{(1)}$, $T^{(2)}$, ... at which processes resynchronize. Every process p changes a_p when $T_p = T^{(i)}$, so the $T^{(i)}$ represent service times. For convenience, I assume that all processes resynchronize at every time $T^{(i)}$—a process p that does nothing at that time can be thought of as performing a resynchronization in which the new value of a_p equals its old value. Time $T^{(0)}$ represents the service time at which the system is started.

Resynchronization may actually be performed by having every process agree when their time T_p should read $T^{(i)}$. However, for this discussion, it is more convenient to have a process convert this into the inverse information: the time that T_p should read when it actually reads $T^{(i)}$. Assuming $\rho_p \ll 1$, if p discovers that T_p should read $T^{(i)}$ when it actually reads $T^{(i)} + \Delta T$, then it knows that T_p should read approximately $T^{(i)} - \Delta T$ when it actually reads $T^{(i)}$.

[3]If C_p is actually a cyclic timer instead of a monotonic clock, then b_p is incremented every time C_p is reset to zero.

At the i^{th} resynchronization, process p thus learns what the "correct" service time should be when T_p reads $T^{(i)}$, and uses this information to reset a_p. (The resynchronization must be carried out in such a way that p receives this information before its clock reaches $T^{(i)}$.) Here, the "correct" time is the one that is agreed upon by all processes as the one to which they want to resynchronize. The most convenient way to describe resynchronization is to have p maintain an "ideal" p-clock $C_p^{(i)}$, that keeps the "correct" time learned during the i^{th} resynchronization—in other words, $C_p^{(i)}$ is the p-clock that has the "correct" time when T_p equals $T^{(i)}$. Hence, $C_p^{(i)}$ represents p's knowledge, as of time $T^{(i)}$, of what the current service time should be. This knowledge becomes obsolete at service time $T^{(i+1)}$, when the resynchronization provides p with more recent information about the "correct" time. Initially $a_p = 1$ and $T_p = C_p^{(0)}$, so T_p equals $C_p^{(0)}$ from the time $T^{(0)}$ when the system is started until $T^{(1)}$.

To resynchronize T_p at service time $T^{(i)}$, p resets the rate of change of T_p so that, in the absence of any further resynchronization, T_p would equal $C_p^{(i)}$ after exactly J seconds, where J is some fixed constant. In other words, if p learned that, when T_p reads $T^{(i)}$, the service time should be $T_p + \Delta T$, then p sets a_p equal to $1 + (\Delta T/J)$. Thus, T_p is always chasing the current "ideal" clock $C_p^{(i)}$. It is convenient to assume that there is at least one resynchronization every J seconds. (We can always add a null resynchronization in which the new ideal clock $C_p^{(i+1)}$ is the same as the old one $C_p^{(i)}$.) Thus, if there is a nonzero correction during each resynchronization, then T_p is always chasing the current ideal clock but never catches it.

Finally, let us make one minor change to this algorithm. Instead of performing the i^{th} resynchronization when T_p equals $T^{(i)}$, we perform it when $C_p^{(i-1)}$ equals $T^{(i)}$. This is a minor difference, since we expect T_p and $C_p^{(i-1)}$ to be close together when either of them reads $T^{(i)}$. However, as Proposition 5 below indicates, the different ideal clocks $C_p^{(i)}$ will be a little more closely synchronized to one another than the service times T_p, so it is slightly better to use them to control the resynchronization. We can now restate our algorithm formally as follows, where J is a fixed parameter.

Resynchronization Algorithm: Let $T^{(0)}$, $T^{(1)}$, ... be an unbounded increasing sequence of times with $T^{(j+1)} - T^{(j)} \geq J$ for all j, let $C_p^{(0)}$, $C_p^{(1)}$, ... be a sequence of p-clocks; for $i > 0$, let $t_p^{(i)}$ be the Universal Time such that $C_p^{(i-1)}(t_p^{(i)}) = T^{(i)}$; and let $t_p^{(0)}$ be the Universal Time such that

$C_p^{(0)}(t_p^{(0)}) = T^{(0)}$. Then the service clock T_p is defined for $t \geq t_p^{(0)}$by

$$T_p(t) = a_p(t)C_p(t) + b_p(t)$$

where a_p and b_p are defined as follows:

- For $t_p^{(0)} \leq t \leq t_p^{(1)}$: $a_p(t) = 1$ and $b_p(t) = T^{(0)} - C_p(t_p^{(0)})$.
- For $t_p^{(i)} < t \leq t_p^{(i+1)}$, $i > 0$: $a_p(t) = 1 + (C_p^{(i)}(t_p^{(i)}) - T_p(t_p^{(i)}))/J$ and $b_p(t) = b_p(t_p^{(i-1)}) + (a_p(t_p^{(i-1)}) - a_p(t_p^{(i)}))C_p(t_p^{(i)})$.

What conditions are required of the ideal clocks $C_p^{(i)}$ to guarantee that the T_p defined by the Resynchronization Algorithm satisfy the correct-rate, synchronization, and correct-time conditions? Since each $C_p^{(i)}$ is a p-clock, running at the same rate as C_p, we know that the ideal clocks $C_1^{(i)}, \ldots, C_n^{(i)}$ satisfy the correct-rate condition with bounds ρ_p. We expect that they must also satisfy the synchronization and correct-time conditions. If the ideal clocks $C_p^{(i)}$ satisfy these conditions, then the service clocks T_p will too, provided that each T_p remains close to the current ideal clock $C_p^{(i)}$. (Of course, the actual bounds in these conditions will not be the same for the T_p as for the $C_p^{(i)}$.)

Since T_p is always "chasing" the current $C_p^{(i)}$, we need a bound on how fast the ideal clocks can change as a result of resynchronization. The required condition is that there exist a constant σ_p such that, during any time interval of length J, the total amount by which p's ideal clocks are changed is less than σ_p. (The constant J is the parameter of the Resynchronization Algorithm.)

bounded correction with constants σ_p: For all p and all j, k: if $j < k$ and $T^{(k)} - T^{(j)} < J$, then

$$\sum_{i=j}^{k-1} |C_p^{(i+1)} - C_p^{(i)}| < \sigma_p$$

The following result shows that the bounded-correction condition ensures that T_p stays close to $C_p^{(i)}$. The appearance of e (which equals 2.71828...) is somewhat surprising.

Proposition 3 *If the $C_p^{(i)}$ satisfy the bounded-correction condition with constants σ_p, then the Resynchronization Algorithm ensures that for every Universal Time t such that $T^{(i)} \leq C_p^{(i)}(t) \leq T^{(i+1)}$:*

$$|C_p^{(i)}(t) - T_p(t)| < e\sigma_p/(e-1)$$

where terms of order σ_p^2/J are neglected.

Proof: Define the time-dependent value C by $C(t) = C_p^{(i)}(t)$, for i determined by the condition $T^{(i)} \leq C_t^{(i)} < T^{(i+1)}$. In other words, $C(t)$ is the time read at Universal Time t by the ideal clock $C_p^{(i)}$ being used at time t. Observe that $C(t)$ advances at the same rate as $C_p(t)$ except that it is incremented by $C_p^{(i)} - C_p^{(i-1)}$ at the resynchronization (Universal) times $t_p^{(i)}$. The bounded-correction condition means that the sum of the absolute value of all such corrections performed during an interval of length J as measured by C is less than σ_p. For convenience, I assume that this condition holds when the length of the interval is measured by p's local clock C_p rather than by UT. This introduces an error in the length of the interval of at most σ_p, which will introduce an error of order at most σ_p^2/J in our bounds.

Let $\alpha(t)$ equal $|C(t) - T_p(t)|$. We must show that $\alpha(t) < e\sigma_p/(e-1)$. A rigorous, straightforward proof is obtained by computing $\alpha(t + \Delta t)$ as a function of $\alpha(t)$ and the resynchronizations performed during the interval $(t, t + \Delta t)$. Such a proof is tedious and unenlightening. Instead, a less rigorous but more intuitive proof is given.

Since C_p satisfies the correct-rate condition with bound less than one, we can approximate it arbitrarily closely on the interval $[t, t + \Delta t]$ by a differentiable function C'_p with a strictly positive derivative. We can then replace all functions of Universal Time by their composition with $C_p'^{-1}$. This substitution leads to the same formulas we would have if C_p were a perfect clock, with $C_p(t) = t$ for all t. (In other words, the substitution effects a change of coordinates from Universal Time to local-clock time.) Therefore, we may assume without loss of generality that C_p is the identity function, so C_p is the Universal Time clock UT.

Let $t = t_0 < t_1 < \cdots < t_n = t + J$, and assume that the only resynchronizations with nonzero corrections to C in the interval $[t, t + J]$ occur at the times t_i. Let c_i be the change to C at time t_i, and let $\Delta t_i = t_i - t_{i-1}$. Then $\sum_{i=1}^{n} \Delta t_i = J$ and, by the bounded-correction hypothesis, $\sum_{i=1}^{n} |c_i| \leq \sigma_p$.

In addition to the resynchronizations at time t_i, we can have an arbitrary number of resynchronizations with zero correction. Such a resynchronization has the effect of slowing the rate at which T_p converges towards C. The maximum value for α is achieved by doing as many such resynchronizations as possible. The value of α obtained by any finite number of resynchronizations is less than the value obtained in the limiting case of continual resynchronization, in which $d\alpha/dt = -\alpha/J$ on each interval (t_{i-1}, t_i). A bit of calculus then shows that

$$\alpha(t_i) < (\alpha(t_{i-1}) + c_{i-1})e^{-\Delta t_i/J}$$

from which we deduce

$$\alpha(t+J) < \sigma_p + (\alpha(t)/e)$$

It is easy to show from this that if $\alpha(t) < e\sigma_p/(e-1)$ then $\alpha(t+J) < e\sigma_p/(e-1)$. To complete the proof of the proposition, we need only show that $\alpha(t) < e\sigma_p/(e-1)$ holds for all t in the initial interval $[T^{(0)}, T^{(0)} + J]$. However, this follows from the bounded-correction hypothesis and the fact that T_p initially equals $C_p^{(0)}$, so $\alpha(T^{(0)}) = 0$. ∎

I leave it as an exercise for the reader to show that the bound of Proposition 3 is the best possible one. (Consider a scenario in which there is a resynchronization every J seconds that advances the ideal clock by almost σ_p and a large number of "zero resynchronizations".) Proposition 3 immediately implies the following two results.

Proposition 4 *If the $C_p^{(i)}$ satisfy the bounded-correction condition with constants σ_p, then the T_p chosen by the algorithm above satisfy the correct-rate condition with bounds $\rho_p + e\sigma_p/(e-1)J$ (neglecting terms of order σ_p^2/J^2). If, for each fixed i, the $C_p^{(i)}$ also satisfy the correct-time condition with bounds ϵ_p on the interval $[T^{(i)}, T^{(i+1)}]$, then the T_p satisfy the correct-time condition with bounds $\epsilon_p + e\sigma_p/(e-1)$ (neglecting terms of order σ_p^2/J).*

Proposition 5 *If, for each fixed i, the $C_p^{(i)}$ satisfy the synchronization condition with bounds δ_{pq} on the interval $[T^{(i)}, T^{(i+1)}]$, then the T_p chosen by the algorithm above satisfy the synchronization condition with bounds $\delta_{pq} + (2e\sigma_p/(e-1))$ (neglecting terms of order σ_p^2/J).*

These results are based upon the assumption that T_p initially equals $C_p^{(0)}$. Suppose this is not the case, so T_p initially differs from the ideal clock $C_p^{(0)}$ by some quantity ΔT_0. The argument used in the proof of Proposition 3 shows that at time $T^{(0)} + J$, T_p will differ from its ideal clock by a quantity

ΔT_1 that is less than $\sigma_p + \Delta T_0/e$, at time $T^{(0)} + 2J$, T_p will differ from its ideal clock by $\Delta T_2 < \sigma_p + \Delta T_1/e$, and so on. Thus, T_p will keep getting closer to the ideal clock until it is within $e\sigma_p/(e-1)$.

Similarly, the synchronization condition assumes that initially $|T_q - T_p| < \delta_{pq}$. If this condition is not met, then the bound on $|T_q - T_p|$ will keep getting smaller until it eventually reaches a value less than $\delta_{pq} + (2e\sigma_p/(e-1))$.

4.2 Synchronization and Time-Correctness of Ideal Clocks

Propositions 4 and 5 show that the T_p satisfy the synchronization and correct-time conditions if the ideal clocks $C_p^{(i)}$ satisfy these conditions during the interval $[T^{(i)}, T^{(i+1)}]$ and the sequence of ideal clocks $C_p^{(0)}$, $C_p^{(1)}$, ... satisfies the bounded-correction condition. Moreover, suppose that the $C_p^{(i)}$ satisfy the synchronization and correct-time conditions with bounds δ_{pq} and ϵ_p just at service time $T^{(i)}$—that is, on the interval $[T^{(i)}, T^{(i)}]$. In other words, suppose only that the ideal clocks are synchronized to within δ_{pq} and lie within ϵ_p of Universal Time when they read $T^{(i)}$. It is easy to see that the $C_p^{(i)}$ must then satisfy the synchronization and correct-time conditions on the entire interval $[T^{(i)}, T^{(i+1)}]$ with bounds $\delta_{pq} + (\rho_p + \rho_q)(T^{(i+1)} - T^{(i)})$ and $\epsilon_p + \rho_p(T^{(i+1)} - T^{(i)})$, respectively.

The requirement that the $C_p^{(i)}$ satisfy the correct-time condition places a bound on how much $C_p^{(i)}$ and $C_p^{(i+1)}$ may differ. In particular, if $C_p^{(i)}$ satisfies the correct-time condition on the entire interval $[T^{(i)}, T^{(i+1)}]$ with bound $\epsilon_p + \rho_p(T^{(i+1)} - T^{(i)})$, and $C_p^{(i+1)}$ satisfies the correct-time condition at time $T^{(i+1)}$ with bound ϵ_p, then $|C_p^{(i+1)} - C_p^{(i)}| < 2\epsilon_p + \rho_p(T^{(i+1)} - T^{(i)})$. These inequalities together with the propositions above easily imply the following result, where the hypothesis asserts that there is at least one and at most r resynchronizations performed every J seconds.

Proposition 6 *If, for each i, the clocks $C_p^{(i)}$ satisfy the synchronization condition with bounds δ_{pq} and the correct-time condition with bounds ϵ_p at time $T^{(i)}$, and there is at least one and at most r of the $T^{(i)}$ in any interval of the form $[t, t+J)$, then the Resynchronization Algorithm guarantees that the clocks T_p satisfy:*

- *the correct-rate condition with bounds*

$$\left(1 + \frac{e}{e-1}\right)\rho_p + \frac{2re}{(e-1)J}\epsilon_p$$

(neglecting terms of order $(r\epsilon_p/J)^2$ and ρ_p^2)

- *the synchronization condition with bounds*

$$\delta_{pq} + \left(\left(1 + \frac{2e}{e-1}\right)\rho_p + \rho_q\right) J + \frac{4re}{e-1}\epsilon_p$$

(neglecting terms of order $(r\epsilon_p)^2/J$ and $\rho^2 J$).

- *the correct-time condition with bounds*

$$\left(1 + \frac{2re}{e-1}\right)\epsilon_p + \left(1 + \frac{e}{e-1}\right)\rho_p J$$

(neglecting terms of order $(r\epsilon_p)^2/J$ and $\rho^2 J$).

In order to compute I_p and I_q, processes p and q can use the values for Universal Time provided by different subsets of the Universal Time providers. Process p does not care which values are used by process q. However, the following result indicates that to achieve the synchronization condition with $\delta_{pq} < \epsilon_p + \epsilon_q$, it is necessary for p and q to agree to compute the values $C_p^{(i)}$ and $C_q^{(i)}$ using values of UT obtained from the same set of providers j. To apply this proposition in our case, let F and G be the functions used to compute T_p and T_q from the values $UT^{(j)}(t)$ broadcast by the Universal Time providers. As I observed earlier, we expect these functions to be translation invariant. (Recall that $\|U\|$ is the width of the interval U.)

Proposition 7 *Let F and G be translation-invariant functions on n-tuples of intervals such that for any intervals $U_1, \ldots, U_n$: if, for each j, $\|U_j\| < \epsilon$ and the intersection of all the U_j is nonempty, then $|G(U_1, \ldots, U_n) - F(U_1, \ldots, U_n)| < \delta$. If $\delta < \epsilon$, then there is some j such that the values of both $F(U_1, \ldots, U_n)$ and $G(U_1, \ldots, U_n)$ depend upon the value of U_j.*

Proof: We assume that there is no such j and show that $\epsilon \leq \delta$. This assumption implies that we can renumber the arguments so that, for some k, the value of F depends only upon its first k arguments and the value of G depends only upon its last $n-k$ arguments. Let $F'(U,V) = F(U, \ldots, U, V, \ldots, V)$ and $G'(U,V) = G(U, \ldots, U, V, \ldots, V)$, with k copies of U and $n-k$ copies of V. The value of F' depends only on its first argument and the value of G' depends only on its second argument. Let U be an interval of width ϵ', where $\epsilon' < \epsilon$. Without loss of generality, we can assume that $F'(U,U) \geq G'(U,U)$. The hypothesis implies that $|G'(U+\epsilon', U) - F'(U+\epsilon', U)| < \delta$. Since the value of G' does not depend upon its first argument, $G'(U+\epsilon', U) = G'(U,U)$; similarly, $F'(U+\epsilon', U) =$

$F'(U+\epsilon', U+\epsilon')$. Hence, $|G'(U,U) - F'(U+\epsilon', U+\epsilon')| < \delta$. However, the translation invariance of F' implies that $F'(U+\epsilon', U+\epsilon') = F'(U,U)+\epsilon'$. Since $F'(U,U) \geq G'(U,U)$, this allows us to conclude that $\epsilon' \leq \delta$. This is true for any $\epsilon' < \epsilon$, which implies the desired result $\epsilon \leq \delta$. ∎

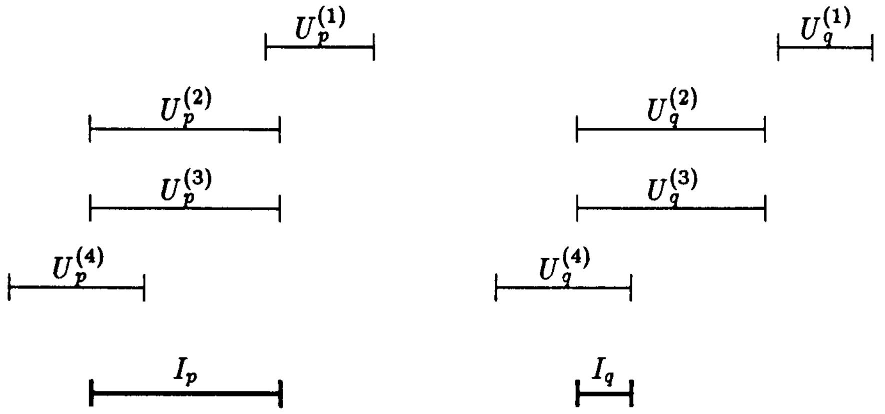
Thus, p and q must agree upon a set of Universal Time providers whose values they will use in computing $C_p^{(i)}$ and $C_q^{(i)}$. This set may change for different values of i (different resynchronizations). The method described in [3] can be employed to obtain agreement on the current set of Universal Time providers that are to be used. Here, let us assume that the values $UT^{(1)}$, ..., $UT^{(m)}$ from m providers are used.

To perform the i^{th} resynchronization, each process p obtains a set of intervals $U_p^{(1)}$, ..., $U_p^{(m)}$, where $U_p^{(j)}$ is the value obtained from Universal Time provider j. (It is the current value of $UT^{(j)}$ "smeared out" by uncertainties in message-transmission time.) Process p sets its p-clock $C_p^{(i)}$ to equal $F(U_p^{(1)}, \ldots, U_p^{(m)})$ (when $C_p^{(i-1)} = T^{(i)}$), where F is some real-valued function of m intervals.

What properties must F have? As we indicated above, we expect F to be translation invariant. We also require that F satisfy the Lipschitz condition for some pseudo-metric. Recall that the Lipschitz condition means that changing each argument by less than δ changes the value of F by less than δ. Translation invariance implies that moving each interval a distance of δ "in the same direction" changes the value of F by δ, so the Lipschitz condition is the strongest "continuity bound" that can be achieved.

The Lipschitz condition implies that we can satisfy the synchronization condition for the $C_p^{(i)}$ by ensuring that $d(U_q^{(j)}, U_p^{(j)}) < \delta_{pq}$ for all j. Intuitively, the Lipschitz condition ensures that p and q will be closely synchronized if, for each Universal Time provider j, the values they obtain from j are almost the same.

Marzullo's function $\mathcal{M}_m^f$ was defined so that $\mathcal{M}_m^f(U_1, \ldots, U_m)$ is the largest interval whose endpoints lie within at least $m - f$ of the intervals U_1, ..., U_m. One might be tempted to define the function F by letting $F(U_1, \ldots, U_m)$ be the midpoint of $\mathcal{M}_m^f(U_1, \ldots, U_m)$. However, this function does not satisfy a Lipschitz condition; in fact, it is not even continuous. Its discontinuity is illustrated in Figure 1, where $m = 4$, $f = 1$, $I_p = \mathcal{M}_4^1(U_p^{(1)}, \ldots, U_p^{(4)})$, and $I_q = \mathcal{M}_4^1(U_q^{(1)}, \ldots, U_q^{(4)})$. In this example, the values of $U_p^{(j)}$ and $U_q^{(j)}$ are ones that could be obtained if Universal Time provider 1 is faulty. Processes p and q see the same values of $U^{(2)}$, $U^{(3)}$, and



(a) Process p's computation. (b) Process q's computation.

Figure 1: Example of discontinuity of Marzullo's function.

$U^{(4)}$, and they see values of $U^{(1)}$ that are almost the same. However, when they apply Marzullo's function to these sets of intervals, they compute very different intervals I_p and I_q. Recall that Marzullo's function computes the optimal value of I_p—that is, the smallest interval that p knows to contain UT. It is this discontinuity in the optimal I_p that makes it impossible, in the presence of malicious faults, to satisfy the synchronization condition with the extra requirement that T_p lies within I_p. More precisely, it can be shown that if a nonfaulty process may assume that it is nonfaulty, then the synchronization condition is incompatible with the requirement that each T_p lies within I_p when the δ_{pq} are smaller than half the widths of the intervals $UT^{(j)}$.

There are a number of functions F that are translation invariant and satisfy the Lipschitz condition for a suitable choice of pseudo-metric. Two such functions are obtained by letting $F(U_1, \ldots, U_m)$ equal the average or the median of the midpoints of the U_i. (These functions satisfy the Lipschitz condition for the midpoint pseudo-metric and therefore for the uniform metric.) A class of functions that includes both of these is defined as follows. Let $A^f(U_1, \ldots, U_m)$ be the average of the multiset of $m - 2f$ numbers obtained by taking the midpoints of all the U_i and omitting the f lowest and f highest of them. Each A^f (with $m > 2l$) is translation invariant and satisfies the Lipschitz condition for the midpoint pseudo-metric. (This follows from the result that, if the numbers x_i and y_i, with $1 \leq i \leq m$, satisfy

$|x_i - y_i| < \delta$, then for any s, the s^{th} largest of the x_i and the s^{th} largest of the y_i differ by at most δ.)

The functions A^f actually give us a somewhat stronger bound on δ_{pq} than that obtained simply from the Lipschitz condition. If $d_m(U_p^{(j)}, U_q^{(j)}) < \delta(j)$, then using A^f to compute $C_p^{(i)}$ and $C_q^{(i)}$ gives an algorithm that satisfies the synchronization condition with bounds δ_{pq} equal to the average of the $m - 2f$ largest of the $\delta(j)$. If the worst-case difference $\delta(j)$ between the values that p and q obtain from Universal Time provider j depends upon j, then the Lipschitz condition guarantees only that δ_{pq} is no larger than the maximum of the $\delta(j)$, while the averaging function A^f can do better. However, the different values of $\delta(j)$ will probably be almost the same in a practical application, so this is not significant.

Next, we consider the correct-time condition: $|UT - C_p^{(i)}| < \epsilon_p$. Suppose that at most f of the Universal Time providers may be faulty. If $m \leq 2f$, so at least half the Universal Time providers are faulty, there is not much hope of finding any algorithm that satisfies the correct-time condition, since all the faulty providers could give the same incorrect value.[4] Therefore, we can assume $m > 2f$. It is then easy to show that there exist nonfaulty providers j and j' such that $A^f(U_p^{(1)}, \ldots, U_p^{(m)})$ lies between the midpoints of $U_p^{(j)}$ and $U_p^{(j')}$. Combining this with the result above for the synchronization condition, it is easy to prove the following result.

Proposition 8 *With the notation of the Resynchronization Algorithm, let $U_p^{(j)}$ be intervals such that, for all j:*

1. *For all p and q: $d_m(U_p^{(j)}, U_q^{(j)}) < \delta_{pq}$.*
2. *For all p: if Universal Time provider j is nonfaulty, then $t_p^{(i)} \in U_p^{(j)}$.*

where each $C_p^{(i)}$ is chosen so that $C_p^{(i)}(t_p^{(i)}) = A^f(U_p^{(1)}, \ldots, U_p^{(m)})$. If there are at most f faulty Universal Time Providers, then the $C_p^{(i)}$ satisfy the synchronization condition with bounds δ_{pq} (neglecting terms of order $(\rho_p + \rho_q)\delta_{pq}$) and the correct-time condition with bounds $\max\{\| U_p^{(j)} \|/2 : \text{provider } j \text{ nonfaulty}\}$ *at time $T^{(i)}$.*

Observe that $A^f(U_1, \ldots, U_m)$ depends only upon the midpoints of the U_i, so $A^f(U_1, \ldots, U_m) = A^f(U_1 \pm \xi_1, \ldots, U_m \pm \xi_m)$ for any numbers ξ_j. While

[4]However, even with more than half the providers faulty, it is still possible to satisfy the synchronization condition.

the averaging function A^f gives reasonable worst-case behavior, it does not make the best use of the available information because it ignores the widths of intervals. Very wide intervals are given the same weight as narrow ones, even though they provide less information. One can construct examples in which the function A^f does not provide the best possible approximation to UT. However, I know of no simple function F satisfying the Lipschitz condition that does better.

4.3 Broadcasting Universal Time

By Proposition 8, the synchronization and correct-time conditions for nonfaulty processes can by met by broadcasting values from the Universal Time providers such that the following two conditions are satisfied, where d_m is the midpoint pseudo-metric on intervals and $U_p^{(j)}$ is the value obtained by p from server j during resynchronization i.

1. If processes p and q are nonfaulty, then for every Universal Time provider j: $d_m(U_q^{(j)}, U_p^{(j)}) < \delta_{pq}$.

2. If process p and Universal Time provider j are both nonfaulty, then $UT(t_p^{(i)})$ lies in the interval $U_p^{(j)}$.

These conditions are very similar to those of the approximate Byzantine agreement problem [1], in which each process p begins with a real value v_p and must choose a real value v'_p such that: (i) for nonfaulty processes p and q: $|v'_q - v'_p| < \delta$, and (ii) v'_p lies within the interval I determined by the largest and smallest of the values v_q. For $\delta \ll ||I||$, the approximate Byzantine agreement problem is known to require $f + 1$ rounds of message passing to handle f failures, even for simple halting failures in a completely connected network.

We can apply lower-bound results for the approximate Byzantine agreement problem to the problem of broadcasting a Universal Time provider's value by letting v_p be the midpoint of the value $U^{(j)}$ that process p obtains directly from provider j. The broadcast problem then becomes a special case of the approximate Byzantine agreement problem. Since a faulty provider may send very different values to different processes, the result for the approximate Byzantine agreement problem implies that $f+1$ rounds of message passing are needed to handle f process failures in a completely connected network.

We assume that if process p sends a message at time t to process q over a path π, and p, q, and π are nonfaulty, then the message is received at some time in the interval $t + [\tau^{\pi}_{\mathrm{min}}, \tau^{\pi}_{\mathrm{max}}]$. However, what if one or more of the processes and/or channels on the path π are faulty? If we rule out "malicious" process behavior and garbled messages, the only type of failure possible is for a message sent over a channel c to take longer than τ^{c}_{max} seconds to be delivered. (A lost message is considered to take very much longer than τ^{c}_{max} seconds.) Even with malicious failures, one can guarantee that, with suitably high probability, a faulty process or channel can do no more than delay a message. Such a guarantee is achieved by using digital signatures, so a faulty process cannot falsify the information contained in a message, and by choosing the value of τ^{c}_{min} so that it is physically impossible for a message to be sent over channel c in less than τ^{c}_{min} seconds—for example, by letting $\tau^{c}_{\mathrm{min}} = 0$. In practice, how one achieves this guarantee depends upon the class of failure one is willing to tolerate. In most cases, it suffices to add simple redundancy to messages. However, tolerating malicious failures requires that a process relay a clock value by appending a digital signature to it without removing other process's signatures [2,4].

The following algorithm by which a Universal Time provider j broadcasts a set of clocks to all processes rests upon the assumption that faults can only delay (or lose) messages. However, if j is faulty, it may send different values to different processes. The choice of the constant k is discussed later.

Byzantine Clock-Broadcast Algorithm: A Universal Time provider j broadcasts a p-clock to every process p as follows. (The sets C_p of p-clocks are initially empty.)

1. j sends an interval $U^{(j)}$ to all its neighbors.
2. If process p receives the interval R along path π at time t, then it adds to C_p the p-clock I^{π}_{p} whose value at time t equals R^{π}, and it relays R to each of its neighbors q unless one or more of the following conditions holds:
    - q is on the path π.
    - C_p already contained p-clocks U and V such that the left endpoint of U is greater than or equal to the left endpoint of R^{π}_{p} and the right endpoint of V is greater than or equal to the right endpoint of I^{π}_{p}.
    - The length of π equals k.

3. When no more messages can arrive, process p sets $UT_p^{(j)}$ to be the p-clock whose left and right endpoints are the maxima of the left and right endpoints of all the p-clocks I_p^{π}.

Note that in the second condition of step 2, U and V could be the same p-clock. This condition can be strengthened so that p need not relay R to q if it knows that q has already added to $\mathcal{C}_q$ a q-clock approximately equal to I_p^{π}. For example, suppose p received R by an Ethernet message and q is on the same Ethernet. If one is willing to assume benign failure modes for the Ethernet, then p could assume that q received the same message at approximately the same time, so there is no need for p to relay it. However, the resulting algorithm would then tolerate only benign Ethernet faults.

Define the *delay* τ and the *variance* γ of a Byzantine Clock-Broadcast Algorithm to be the maximum of $\tau_{\max}^{\pi}$ and γ^{π} for all paths π from j of length at most k. Suppose the clocks of any two nonfaulty processes differ by at most ζ. Since messages along faulty paths can be ignored, it is easy to see that if a Byzantine clock broadcast is initiated by provider j when its clock equals T, then a process can ignore any message that reaches it over a path π of length l when its clock reads later than $\tau_{\max}^{\pi} + l\zeta$. Hence, each process p can compute its p-clock $UT_p^{(j)}$ at time $\tau + k\zeta$, where τ is the delay of the algorithm.

Suppose that, in executing the Byzantine Clock-Broadcast Algorithm, p receives R along path π. Let r be a node on this path such that ϕ is the subpath of π going from j to r, and η is the subpath going from r to p, so p received R because r relayed R along η. Suppose that r also relays R to q along ψ. If p, q, r, ψ, and η are nonfaulty, then Proposition 2 implies that if p received R at time t, then

$$d_u(I_p^{\pi}(t), I_q^{\phi\psi}(t)) < \gamma^{\eta} + \gamma^{\psi}$$

(neglecting terms such as $\rho_p \tau_{\max}^{\pi}$). The following result follows easily from this.

Proposition 9 *Assume that for every pair of nonfaulty processes p, q there are paths $\phi\eta$ from Universal Time provider j to p and $\phi\psi$ from j to q of length at most k such that η and ψ are nonfaulty. If a Byzantine Clock-Broadcast algorithm with variance γ is started at time t to broadcast an interval $U^{(j)}$, then:*

1. *If p and q are nonfaulty, then $d_u(UT_p^{(j)}(t), UT_q^{(j)}(t)) < 2\gamma$*

2. If j and p are nonfaulty and $UT(t) \in U^{(j)}$ then $UT(t) \in UT_p^{(j)}(t)$.

(neglecting terms of order $\rho_p \tau_{\max}^{\pi}$ for paths π of length at most k from j to p).

For any particular network, the hypothesis of Proposition 9 can be satisfied by making k large enough if every pair of nonfaulty processes are connected by some nonfaulty path. The choice of k and the actual set of channels to use for the broadcast will be a compromise between the conflicting desires to increase reliability and reduce the number of messages sent.

The conclusion of Proposition 9 is almost but not quite in the form necessary for implying the hypothesis of Proposition 8. This is because the value $U_p^{(j)}$ used in the actual algorithm will be $UT_p^{(j)}(t_p^{(i)})$ rather than $UT_p^{(j)}(t)$. However, the conclusion of Proposition 9 remains valid after replacing t by the values $t_p^{(i)}$ if we can neglect terms of order $\rho_p|t_p^{(i)} - t|$. These terms will be negligibly small if the time t, when the clock broadcast is begun, is close to the resynchronization time $t_p^{(i)}$. The broadcast needs to be begun early enough so that every process p receives its value before time $t_p^{(i)}$, which is the time when its clock $C_p^{(i)}$ reads $T^{(i)}$. If the $C_p^{(i-1)}$ satisfy the correct-time condition with bounds ϵ_p and τ is the maximum of $\tau_{\max}^{\pi}$ for all paths from j of length k, then we get a minimum value for $|t_p^{(i)} - t|$ on the order of $\epsilon_p + \tau$.

4.4 The Complete Algorithm

We now have all the pieces necessary to construct an algorithm to compute the T_p. First, one must select disjoint sets $\mathcal{P}_i$ of processes such that if p and q want to synchronize their times so that $\delta_{pq} < \epsilon_p + \epsilon_q$, then they both lie within the same set $\mathcal{P}_i$. If the sets $\mathcal{P}_i$ can change, then the algorithm of [3] is used to guarantee that all nonfaulty processes agree upon the current collection of sets.

A process p not in any set $\mathcal{P}_i$ simply chooses T_p to be the midpoint of I_p. All processes p in the same set $\mathcal{P}_i$ choose their service times T_p by the following algorithm.

Service Time Algorithm:

1. The processes in $\mathcal{P}_i$ choose a set of Universal Time providers. The method of [3] is used to ensure that all processes in $\mathcal{P}_i$ agree upon

a set of providers that are thought to be nonfaulty and to provide suitable values.[5] Let this set of providers be numbered from 1 to m.

2. For a sequence of predetermined times $T^{(i)}$ and time providers j_i, when the maximum (right-hand endpoint) of $UT^{(j_i)}$ equals $T^{(i)} - \tau - k(\chi + \rho H)$, provider j_i executes the Byzantine Clock-Broadcast Algorithm, with $U^{(j_i)}$ equal to the current value of $UT^{(j_i)}$, to broadcast a p-clock $UT_p^{(j_i)}$ to every process p in $\mathcal{P}_i$, where
   - τ is the delay of the broadcast algorithm.
   - $\chi \geq \|UT_{\|}^{(j_i)}/2$ for all p.
   - H is a constant such that each provider j broadcasts its value of $UT^{(j)}$ in this way at least once every H seconds.
   - k is the parameter of the broadcast algorithm.
3. Each process p sets $C_p^{(i)}$ equal to the p-clock $A^f(UT_p^{(1)}, \ldots, UT_p^{(m)})$.
4. Process p uses the Resynchronization Algorithm to compute T_p.

It follows from the correct-time condition in Proposition 10 below that provider j_i initiates its broadcast early enough so each process p can compute $C_p^{(i)}$ by the time $C_p^{(i-1)}$ reaches $T^{(i)}$.

Propositions 6, 8, and 9 allow us to deduce that the T_p satisfy the correct-rate, synchronization, and correct-time conditions. However, the bounds in these conditions become rather complex. Therefore, only the simpler conditions for the ideal clocks $C_p^{(i)}$ are given; the corresponding conditions for the service clocks T_p are obtained from these bounds by applying Proposition 6.

Proposition 10 *If at most f of the Universal Time providers are faulty, then the clocks $C_p^{(i)}$ constructed in step 3 of the Service Time Algorithm satisfy*

- *the clock-synchronization condition with bounds $2\gamma + (\rho_p + \rho_q)H$.*
- *the correct-time condition with bounds $\chi + \rho_p H$.*

on the interval $[T^{(i)}, T^{(i+1)}]$ (neglecting terms of order $\rho_p \tau$).

[5]Since agreement takes time, a provider will have to remain in the set of chosen providers for some period of time after it is discovered to be faulty before the processes agree to eliminate it. Thus, even if we assumed that faulty Universal Time providers can be detected, the algorithm for choosing T_p still has to tolerate faulty providers.

Glossary

C_p: Process p's local clock.

$C_p^{(i)}$: A mythical ideal clock that runs at the same rate as C_p and maintains the "correct" time, as learned by p in the i^{th} resynchronization.

d_m: The midpoint pseudo-metric on (bounded) intervals; $d_m(U, V)$ is defined to be the distance between the midpoints of U and V.

d_u: The uniform metric on intervals; $d_u([x, y], [v, w])$ is defined to be the maximum of $|v - x|$ and $|w - y|$.

e: 2.71828182845904523536028747135266249775724709369995957496 69...

H: A parameter of the Service Time Algorithm, chosen to be a length of time such that each Universal Time provider broadcasts its value at least once every H seconds.

i: Used as a superscript to denote a clock-synchronization event.

I_p: A clock range, maintained by time server p, that is guaranteed to contain UT.

j: Used as a sub- or superscript to denote a Universal Time provider.

J: A parameter of the Resynchronization Algorithm, chosen to be a length of time such that there is at least one resynchronization every J seconds. (At the i^{th} resynchronization, process p sets the running rate of T_p so that T_p would equal $C_p^{(i)}$ after exactly J seconds, in the absence of further resynchronization.)

k: The maximum-length path by which messages travel in a Byzantine Clock-Broadcast Algorithm.

Lipschitz condition: F satisfies a Lipschitz condition for pseudo-metric d if $d(U_i, V_i) < \delta$ for all i implies $|F(U_1, \ldots, U_M) - F(V_1, \ldots, V_M)| < \delta$.

pseudo-metric: A nonnegative function d such that: (i) $d(U, V) = d(V, U)$, (ii) $d(U, V) + d(V, W) \geq d(U, W)$, and (iii) $d(U, U + \epsilon) = |\epsilon|$.

R^π: The interval $R + [\tau_{\min}^\pi, \tau_{\max}^\pi]$, where R is an interval and π is a path.

T_p: The time provided by time-service process p.

$T^{(i)}$: The time of the i^{th} resynchronization (as read by the clocks $C_p^{(i-1)}$).

translation invariance: F is translation invariant if $F(U_1 + x, \ldots, U_m + x) = F(U_1, \ldots, U_m) + x$.

UT: Universal Time—the ideal standard, closely approximated by clocks at the National Bureau of Standards and other places throughout the world.

$U^{(j)}$: An interval containing UT broadcast by Universal Time provider j during a particular synchronization.

$U_p^{(j)}$: The interval obtained by process p when $U^{(j)}$ is broadcast by Universal Time provider j.

$UT^{(j)}$: A clock range maintained by Universal Time provider j that contains UT.

$UT_p^{(j)}$: The p-clock obtained by process p when Universal Time provider j broadcasts $UT^{(j)}$.

$t_p^{(i)}$: The value of UT at which process p's ideal clock $C_p^{(i-1)}$ reads $T^{(i)}$.

γ: The variance of a Byzantine Clock-Broadcast Algorithm.

γ^c: When c is a channel, it equals $\tau^c_{\mathbf{max}} - \tau^c_{\mathrm{min}}$, the uncertainty in message-transmission time over channel c. For a path π, γ^π is the sum of the γ^c for all channels c in the path.

δ_{pq}: An upper bound on the difference between time values provided by nodes p and q—e.g., an upper bound for $|T_p - T_q|$.

ϵ: An upper bound on half the width of $UT^{(j)}$ for all nonfaulty Universal Time providers j.

ϵ_p: An upper bound on the difference between Universal Time and a value provided by process p—e.g., an upper bound on $|UT - T_p|$.

κ_p: An upper bound on the error in the rate of change of the service time T_p provided by process p.

ρ_p: An upper bound on the error in the running rate of C_p.

σ_p: The maximum amount by which resynchronization can change p's ideal clocks $C_p^{(i)}$ during any time interval of J seconds duration.

τ: The delay of a Byzantine Clock-Broadcast Algorithm.

$\tau_{\min}^c$: The minimum message delay for a message sent across channel c. (It includes the time needed to generate the message.) For a path π, $\tau_{\min}^{\pi}$ is the sum of the minimum message delays for all channels in path π.

$\tau_{\max}^c$: The maximum message delay for a message sent across channel c (including the time needed to generate the message). For a path π, $\tau_{\max}^{\pi}$ is the sum of the maximum message delays for all channels in path π.

χ: A parameter of the Service Time Algorithm, at least half the maximum width of $UT_p^{(j)}$ for every nonfaulty Universal Time provider j and nonfaulty process p.

$\|\ldots\|$: The width of an interval, defined by $\|[x,y]\| = y - x$.

$+$: The sum of two intervals is defined by $[u,v] + [x,y] = [u+x, v+y]$. The sum of an interval and a number is defined by $[u,v] + x = [u+x, v+x]$.

$\pm$: For an interval $[u,v]$ and a number $\delta \geq 0$, $[u,v] \pm \delta$ is the interval $[u-\delta, v+\delta]$.

Acknowledgments

Michael Fine originally suggested to me the problem of a network time service. Discussions with Andrei Broder helped me formulate the solution. Andrei Broder and Tim Mann discovered many small errors in earlier versions of this document.

References

[1] Danny Dolev, Nancy A. Lynch, Sholmit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1985.

[2] Joseph Y. Halpern, Barbara Simons, and Ray Strong. Byzantine clock synchronization. In Jayadev Misra, editor, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 89–102, Association for Computing Machinery, Inc., New York, August 1984.

[3] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280, April 1984.

[4] Leslie Lamport and P. M. Melliar-Smith. Byzantine clock synchronization. In Jayadev Misra, editor, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 68–74, Association for Computing Machinery, Inc., New York, August 1984.

[5] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In Jayadev Misra, editor, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 75–88, Association for Computing Machinery, Inc., New York, August 1984.

[6] Keith A. Marzullo. *Maintaining Time in a Distributed System*. PhD thesis, Stanford University, March 1984.

Index

SRC Reports

by Leslie Lamport

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301