Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems

P. RAMANATHAN, DILIP D. KANDLUR, AND KANG G. SHIN, SENIOR MEMBER, IEEE

Abstract—**Clock synchronization in the presence of faults has been studied extensively in recent years and several software and hardware solutions have been proposed. Software solutions require nodes to exchange and adjust their individual clock values periodically. Since the clock values are exchanged via message passing, the time overhead induced by the software solutions can be substantial, especially if a tight synchronization is desired. Hardware solutions, on the other hand, use additional hardware to achieve a very tight synchronization with minimal time overhead. However, the prohibitive cost of the additional hardware limits their usefulness to small systems.**

In this paper, we propose a synchronization scheme that strikes a balance between the hardware requirement and the clock skews attainable. The proposed scheme is a software algorithm that uses minimal additional hardware to achieve reasonably tight synchronization. Unlike other software solutions, the guaranteed worst case skews can be made insensitive to the maximum variation of message transit delay in the system. The scheme is particularly suitable for large partially-connected distributed systems with topologies that support simple point-to-point broadcast algorithms. Examples of such topologies include the hypercube and the mesh interconnection structures.

Index Terms—**Clock skew, distributed systems, hexagonal mesh, hypercube, synchronization.**

I. INTRODUCTION

A GLOBAL time base has been widely recognized as an important requirement in distributed computing systems. It can be used to simplify the solutions to several design problems such as checkpointing, interprocess communication, resource allocation, and transaction processing, especially in the presence of faults [11]. It can also be used to implement features such as deadlines and timeout that are essential for correct operation of distributed real-time systems.

A global time base can be established by synchronizing all the local clocks in the distributed system. This synchronization would not have been much of a problem had all the clocks including the faulty ones behaved consistently with one another. However, when some of the faulty clocks can behave in any arbitrary manner, clock synchronization can pose some serious problems. For instance, a faulty clock can make it difficult for the other clocks to synchronize themselves by either omitting or sending conflicting clock information during the course of synchronization. This kind of arbitrary behavior is referred to as a *Byzantine fault*, and Lamport and Melliar-Smith were the first to develop a solution to the problem of synchronizing clocks in the presence of Byzantine faults [12], [13]. Since then, the clock synchronization problem has been studied extensively, and several software and hardware solutions have been proposed.

The software solutions are flexible and economical but require additional messages to be exchanged solely for synchronization [7], [13], [15], [20]. Due to their dependence on message exchanges, the worst case skews guaranteed by most of these algorithms are greater than the maximum variation in message transit delay in the system. The hardware solutions, on the other hand, can achieve a very tight synchronization with minimal time overhead, but require special additional hardware at each node of the distributed system [9], [10], [19], [22]. The cost of additional hardware precludes their use in large distributed systems, unless a very tight synchronization is essential. The hardware solutions also require a separate network of clocks that is usually different from the interconnection network between the nodes of the distributed system [19].

In this paper, we propose a synchronization scheme that strikes a balance between the hardware requirement and the clock skews attainable. Specifically, we shall develop a software solution which requires some hardware support at each node of the distributed system. The additional hardware required at each node is minimal as compared to pure hardware solutions and our solution does not require a separate network of clocks. However, due to the hardware support, the worst case skews in our solution are insensitive to the maximum time required for a broadcast and are about two or three orders of magnitude tighter than most software solutions (see Section IV). Furthermore, the distribution of clock values by different nodes is spread over the entire resynchronization interval. This should be contrasted to the approach used by other software solutions where all the nodes broadcast their clock values during a specific window at the end of the resynchronization interval. Staggering the distribution of clock values over the entire resynchronization interval is a highly desirable feature because there is no abrupt degradation in the message

Manuscript received July 1, 1989; revised November 20, 1989. This work was supported in part by NASA Grant NAG-1-296 and ONR Contract N00014-85-K-0122. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

P. Ramanathan is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.

D. D. Kandlur and K. G. Shin are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8933893.

deliver times caused by the almost simultaneous broadcast of clock values by all the nodes in the system.

A hardware-assisted software scheme has been proposed earlier by Kopetz and Ochsenreiter [24]. However, their scheme was mainly intended for a system in which the nodes are interconnected through a broadcast bus. In contrast, the scheme proposed here is intended for a distributed system with point-to-point interconnection topology. The problem here is more complicated because the clock messages have to be relayed through several intermediate before reaching their destination. There are also some schemes in which the worst case skews do not depend on the maximum variation in the message transit delay during a broadcast [2], [6]. In [6], clocks are intended for simulating the concept of rounds for distributed agreement algorithms rather than to keep track of elapsed time intervals. Hence, the definition of their clock is different from the definition in most other schemes [7], [9], [10], [13], [15], [20], [22]. The scheme in [2] assumes that given a delay bound one can accurately estimate the probability of the message transit delay being greater than that bound. The main idea is to let the clock process at each node make several attempts to read the clock at another node. At the end of each attempt, the clock process can evaluate the maximum error that might occur if the clock value obtained in that attempt were used to estimate the skew. By retrying a sufficient number of times,[1] a clock process can read the clock at another node to any given precision with probability as close to one as desired. This scheme is particularly suitable for systems that have a master–slave arrangement in which one clock has been designated or elected as a master and the other clocks act as slaves. However, this scheme is not suitable for synchronizing large distributed systems because the algorithms to detect a failure of the master and to elect a new one are complex and time-consuming, especially in the presence of Byzantine faults. There is also a nonzero probability of not being able to synchronize the nonfaulty clocks, due to the probabilistic nature of the clock reading mechanism.

For the reasons outlined above, we want deterministic algorithms for synchronization which maintain clocks that can be used to measure real time intervals. The scheme proposed in this paper is similar to the interactive convergence algorithm (CNV) in [13]. The differences arise mainly because CNV is intended for a fully-connected topology, as opposed to a partially-connected topology like the hypercube [17] or the mesh [1]. Although CNV could be used in a partially-connected system, it is not well suited for such a system due to the fact that the worst case skews in CNV are greater than the difference between the maximum and the minimum message transit delay in the system. In contrast, as illustrated in Section IV, the worst case skews in the proposed scheme are orders of magnitude smaller than the maximum time required for a reliable broadcast. In addition, the worst case skews do not depend on this maximum time beyond a certain limit.

This paper is organized as follows. In Section II, the clock synchronization problem is formally stated. The proposed software synchronization algorithm and the proof of its correctness are presented in Section III. Several numerical examples are given in Section IV. The paper concludes with Section V.

[1] This will, however, induce additional time overhead.

II. Problem Statement

Before formally stating the clock synchronization problem, it is necessary to introduce the notation and the terminology to be used here, which are similar to the ones in [13]. (They are repeated here for completeness.)

Definition 1: The time that is directly observable in some particular clock is called its *clock time*. This should be contrasted to the term *real time*, which is measured in an assumed Newtonian time frame that is not directly observable.

Definition 2: Let c be a mapping from clock time to real time, where $c(T) = t$ means that at clock time T the real time is t. Then, two clocks c_1 and c_2 are said to be *δ-synchronized* at a clock time T if and only if $|c_1(T) - c_2(T)| \leq \delta$.

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. It is convenient to pretend that clocks run continuously, so a clock is a continuous function on some interval. Also, the nonfaulty clocks are assumed to be monotonic. An implementation that achieves this monotonicity is discussed later in this paper.

Definition 3: 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 $c(T_i) = t_i$, $i = 1, 2$, and for all T in $[T_1, T_2]$

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

for some constant ρ that represents the drift rate of the good clocks.

The problem of clock synchronization can now be formally stated as follows. Consider a distributed system with N nodes in which a maximum of m of these nodes are faulty. Each node executes a clock process (CP) to maintain a time base for all the activities on that node. Let c_p denote the clock at node p. For simplicity, suppose the clocks are resynchronized every R seconds. Let $T^{(i)} = T^{(0)} + iR$ and $R^{(i)}$ be the interval $[T^{(i)}, T^{(i+1)}]$. Due to the correction made at the end of every resynchronization interval, the clock at each node can be viewed as a sequence of logical clocks, one for each resynchronization interval. Let $c_p^{(i)}$ denote the logical clock at node p in the ith resynchronization. Then, $c_p^{(i)}(T) = c_p^{(0)}(T + \xi_p^{(i)})$, where $\xi_p^{(i)}$ is the aggregate correction applied to the clock since the start of the system. For notational convenience, let $\xi_p^{(0)} = 0$ and $c_p^{(0)} = c_p$.

Definition 4: A node p is said to be *nonfaulty up to time $T^{(i+1)}$* if it is nonfaulty during the real time interval $[c_p^{(0)}(T^{(0)}), c_p^{(i)}(T^{(i+1)})]$.

Using the above notation, the two clock synchronization conditions can be stated as

S1) If nodes p and q are nonfaulty up to time $T^{(i+1)}$, then $\forall T \in R^{(i)}$

$$|c_p^{(i)}(T) - c_q^{(i)}(T)| < \delta \qquad \text{for some constant } \delta.$$

S2) If node p is nonfaulty up to time $T^{(i+1)}$, then

$$|\xi_p^{(i+1)} - \xi_p^{(i)}| < \Sigma \qquad \text{for some constant } \Sigma.$$

Intuitively, the first condition states that the difference in clock times on two nonfaulty nodes is always bounded. The second condition states that there is a bound on the amount by which a clock can change its value during one resynchronization interval.

III. Proposed Solution

Since the synchronization scheme proposed in this paper is based on Algorithm CNV, we will first briefly describe the algorithm as proposed in [13]. We will then identify the problems of adapting the algorithm to a distributed system that is not fully connected, and present a solution that eliminates those problems.

Algorithm CNV assumes that the clocks are initially synchronized and that they drift apart only by a bounded amount during a resynchronization interval. In every resynchronization interval, each CP reads the value of the clock at all other nodes. If the value of a clock read differs from the clock at its node by an amount greater than a threshold, CP replaces that value by its own clock value. CP then computes the average of all such values and sets the local clock to that average. In [13], it is shown that this algorithm can achieve synchronization, and requires a minimum of $3m + 1$ nodes to tolerate m faults.

Three major problems arise when this algorithm is used in a distributed system that is not fully connected. First, the clock message received at a node may be corrupted by a faulty intermediate node through which it was relayed. Second, due to the queueing delay for clock messages, there may be a substantial difference between the real time at which a CP sends the clock value and the real time at which another CP receives that message. Therefore, subtracting the clock value in the received message from the current clock value will not reflect the actual skew that exists between the clocks of the two nodes. This problem, which is present even if the system is fully connected, is aggravated when system is partially-connected because the clock message has to be relayed through intermediate nodes. Third, there may be a delay between the receipt of a clock message from the network and the time at which it is processed. A similar delay is possible between the time at which a CP sends the message and the time at which the message is placed on the network. The first problem is eliminated in the proposed solution by using a broadcast algorithm that delivers multiple copies of the message to all CP's through node-disjoint paths. The second problem is reduced by requiring the CP at each intermediate node to append to the message the delay (according to the local clock) incurred at that node. The third problem is handled by recording the time at which a clock message is sent or received.

A CP broadcasts the local clock value to all other CPs' at a specified time in the resynchronization interval according to its clock. The broadcast algorithm is such that all CP's receive multiple copies of the clock message through node-disjoint paths. The number of copies used in the broadcast algorithm depends on the maximum number of faults to be tolerated and the fault model for the system. When a CP receives a clock message sent by some other CP, it records the time (according to its local clock) at which the message was received. Then, in accordance with the broadcast algorithm, it relays the message to other CP's. Before relaying the message, it appends to the message the time elapsed (according to its own clock) since the receipt of the message. At the end of a resynchronization interval, it computes the skews between the local clock and the clock of the source node for each one of the copies it has received. It then selects the $(m + 1)$th largest value as an estimate of the skew between the two clocks. It is shown in Section III-D that this selected value is a good estimator of the clock skew between the two nodes. The average of the estimated skews over all nodes is used as the correction to the local clock. As in CNV, the proposed scheme requires a minimum of $3m + 1$ nodes to tolerate m Byzantine faults.

The above steps are explained in Section III-C in terms of four concurrent tasks: CLOCK_SEND, CLOCK_RELAY, CLOCK_RECEIVE, and CLOCK_CORRECTION. An efficient implementation of these tasks requires some hardware support and a reliable broadcast mechanism.

A. Hardware Support

A clock message contains the following information: the node at which the broadcast was initiated (namely, the initiator), the node from which the message was most recently relayed (namely, the immediate sender), and five words $W_1, W_2, \cdots, W_5$ shown below. The relative ordering of these words in the message depends on the format being used in the system.

- W_1: Clock time of the initiator when this message was transmitted.
- W_2: Clock time of the immediate sender when it received this message.
- W_3: Clock time of the immediate sender when it relayed this message.
- W_4: Accumulated transit delay not counting the delay at the immediate sender.
- W_5: The time on the local clock when this message was received.

The hardware circuitry at each node helps in updating these words. When a clock message is received it updates W_5 with the local time. An interrupt is then generated to the processor that runs the clock tasks at that node. After the clock tasks have completed their processing, the message is scheduled for forwarding. While the message is being transmitted, the hardware circuitry inserts the local clock time in place of W_3. The other operations performed on these words are explained in the formal description of the clock tasks later in this section.

A special circuit is required to maintain a continuous monotone logical clock at each node. A possible solution is as follows. A high-frequency clock generated by a crystal oscillator is divided by a programmable counter and the divided clock is used to increment a counter that contains the logical clock. The factor used to divided the high-frequency clock depends on the correction that needs to be applied to the logical clock. If at the end of a resynchronization interval, the correction is positive (negative), then the factor used to divide the high-frequency clock is increased (decreased) from its nominal value. For example, a 32 MHz clock can nominally be divided by 32 to get a logical clock whose resolution is 1 μs.

This factor can be changed either to 36 or 28 depending on whether the correction is positive or negative, respectively.

B. Reliable Broadcast

A reliable broadcast mechanism is especially important in a partially-connected distributed system because a process may have to deliver its clock value to the other processes via several intermediate nodes. However, since the nonfaulty CP's need not agree on the clock value at a faulty node, it is not necessary to use an interactive consistency algorithm [5], [14] to correctly deliver the clock values.

In the presence of m Byzantine faults, the following scheme can be used to reliably deliver the clock values. For each pair of nodes, determine a set of $2m + 1$ node-disjoint paths. The CP at a nonfaulty node then individually sends $2m + 1$ copies of the local clock value to the CP at each node through the predetermined paths for that node. As shown in [23] this scheme does not require authentication. It requires: 1) the network to be $2m + 1$ connected, and 2) a process should be able to identify the node from which a received message originated and the node from which it was most recently relayed. These two requirements are necessary for any algorithm that is resilient to m Byzantine faults when authentication is not available [4]. It is shown in Theorem 1 that a CP can reliably estimate the skew between the local clock and the clock at another node from the multiple copies of the clock message it receives from the CP at that node.

The above scheme, albeit very general, requires each CP to send a large number of messages, thus resulting in substantial time overhead. However, for interconnection topologies, like the hypercube [17] and the C-wrapped hexagonal mesh [1], there are reliable broadcast algorithms available [8], [16] to reduce this overhead. As in the scheme described above, they deliver $2m + 1$ copies of the clock message through node-disjoint paths to every node in the system. But they make use of the properties of the topology to reduce the number of messages that have to be sent by a process. In fact, one can show that the number of messages sent by a process is the minimum required to guarantee a reliable delivery without using digital signatures. Due to the importance of a reliable broadcast mechanism for distributed algorithms that are resilient to Byzantine faults, one can argue that most fault-tolerant distributed systems will use topologies that support simple broadcast algorithms similar to the ones in [8] and [16]. Consequently, such a broadcast mechanism will be assumed to exist in the distributed systems being synchronized.

C. Detailed Description

A CP is comprised of four concurrent tasks. The operations of these four tasks required can be described in terms of the following basic functions.

local_time( )	Returns the current time on the local clock.
btime(p)	Returns the time at which p is supposed to broadcast its clock message in the current resynchronization interval.
initiate_bcast(k, t)	Initiates broadcast of k copies of a clock message with time set at t.
$w1(M), \cdots, w5(M)$	The five clock words in the message M.
receive( )	Blocks until a clock message arrives and returns the received message.
initiator(M)	Returns the node at which the broadcast of message M was initiated.
select(*skews*, l)	*skews* is an array of estimated skews with respect to a particular node computed from copies of clock messages received from the CP at that node. The function select( ) orders these skews and returns the lth value.

```
1. loop
2.     if (local_time( ) = btime(p))
3.         initiate_bcast(2m + 1, local_time( ));
4.     endif;
5. endloop;
```

Fig. 1. CLOCK_SEND task at node p.

CLOCK_SEND: The CLOCK_SEND task is responsible for initiating a reliable broadcast in each resynchronization interval, as shown in Fig. 1. The time at which the broadcast takes place depends on the node and is *staggered* throughout the resynchronization interval. This alleviates the problem of transient loading which would otherwise occur due to the almost simultaneous broadcast of clock messages. The reliable broadcast delivers $2m + 1$ copies of the message to each CP through node-disjoint paths if all CP's adhere to the broadcast algorithm. However, if some nodes are faulty and do not adhere to the broadcast algorithm, then fewer than $2m+1$ copies may be delivered. Given that there are at most m faulty nodes in the system, each CP will receive at least $m + 1$ copies relayed through nonfaulty nodes. It is shown in Theorem 1 that the CP's at all nonfaulty nodes can reliably estimate the clock skew from the copies received.

Due to contention for network resources there may be a time delay between the initiation of a broadcast (i.e., line 3 of Fig. 1) and the time at which the copies are transmitted on the network. To account for this delay, the time on the local clock is inserted in W_1 and W_2 while the broadcast is being initiated. Also, the cumulative transit delay W_4 is initialized to 0. During transmission, the hardware circuitry at that node will insert the current time in place of W_3. The delay between the initiation of the broadcast and the time at which the copy was transmitted, indicated by $(W_3 - W_2)$, will be accumulated in W_4 by the CLOCK_RELAY task of the node that is receiving this message.

CLOCK_RELAY: The CLOCK_RELAY task (Fig. 2) gets activated when a clock message broadcast by some other CP is received. It is responsible for relaying the message to other CP's as per the broadcast algorithm. It is also responsible for updating the words $W_1, \cdots, W_5$ in the clock message.

The CLOCK_RELAY task checks whether the initiator specified in the message is supposed to be broadcasting its clock value at this time instant. This check is used to suppress spurious messages generated from faulty nodes. This is possible if there is a bound on the time to complete a broadcast, say U. If the message is authentic, $(W_3 - W_2)$ is added to W_4 to accumulate the transit delay incurred by the message at

```
constant MAX_BROADCAST_TIME = $U$;
loop
    $M$ := receive( );
    source := initiator($M$);
    if ( |local_time( ) – btime(source)| ≤ MAX_BROADCAST_TIME )
        w4($M$) := w4($M$) + w3($M$) – w2($M$);
        w2($M$) := w5($M$);
        relay the message according to the broadcast algorithm;
    endif;
endloop;
```

Fig. 2. CLOCK_RELAY task.

```
constant MAX_BROADCAST_TIME = $U$;
loop
    $M$ := receive( );
    source := initiator($M$);
    if ( |local_time( ) – btime(source)| ≤ MAX_BROADCAST_TIME )
        copy := Current_copy[source];
        Delta[source][copy] := w5($M$) – (w4($M$) + w3($M$) – w2($M$)) – w1($M$);
        Current_copy[source] := Current_copy[source] + 1;
    endif;
endloop;
```

Fig. 3. CLOCK_RECEIVE task.

```
constant THRESHOLD = $\Delta$;
loop
    if (local_time( ) = resync_time( ))
        total_skew := 0;
        for q = 1 to NUMBER_OF_PROCESSES
            Skew[q] := select(Delta[q], $m$ + 1);
            if (Skew[q] > THRESHOLD)
                Skew[q] = 0;
            endif;
            total_skew := total_skew + Skew[q];
        endfor;
        correction := total_skew / NUMBER_OF_PROCESSES;
    endif;
    initialize Current_copy, etc. for the next resynchronization interval;
endloop;
```

Fig. 4. CLOCK_CORRECTION task.

the immediate sender. This node now becomes the immediate sender for the next step.

CLOCK_RECEIVE: When a message is received, the CLOCK_RECEIVE task (Fig. 3) first verifies the authenticity of the message. It then computes the skew between the local clock and the clock at the node where the broadcast was initiated and adjusts the skew by subtracting the accumulated delay. This operation is performed for each copy that is received and the resulting skews are passed on to the CLOCK_CORRECTION task.

CLOCK_CORRECTION: The CLOCK_CORRECTION task (Fig. 4) corrects the local clock based on the skews calculated by CLOCK_RECEIVE. It first selects the estimated skew for a particular node from the values computed for the copies received from that node. If the estimated skew is greater than a threshold, Δ, it is replaced by zero. The choice of Δ follows from the proof of correctness (see Lemma 5). The correction applied to the local clock is the average of the estimated skews over all nodes.

D. Proof of Correctness

In this section, we show that the proposed solution can ensure synchronization if the following assumptions hold in the system.

A1) For all nodes p and q, $|c_p(T^{(0)}) - c_q(T^{(0)})| < \delta_0$, where δ_0 is a constant.

A2) If node p is nonfaulty during the time interval $[t_1, t_2]$, then c_p is a good clock during the interval, i.e.,

$$|c_p(t) - c_p(t_1)| < \left(1 + \frac{\rho}{2}\right)(t - t_1) \qquad \forall t \in [t_1, t_2].$$

A3) In a reliable broadcast initiated from a nonfaulty node, all copies relayed through nonfaulty nodes are delivered within a time bound U.

A4) Suppose S1 and S2 hold for i and node p is nonfaulty up to time $T^{(i+1)}$. Also suppose that the CP at a nonfaulty node q sends a message containing q's clock value to p at some time T_0 in $R^{(i)}$ according to q's clock. This message reaches p through a relay (possibly empty) of nodes. The CP at each relay node alters the words $W_1, \cdots, W_5$ in the message as described in CLOCK_RELAY. From the message received, the CP at p computes a value Δ_{qp} as described in CLOCK_RECEIVE. If all the relay nodes are nonfaulty, then

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

where $Q = W_5 - (W_4 + W_3 - W_2)$ is an estimate of the total transit delay incurred by the message and ϵ is a constant.

A5) The upper bound U on the time required to complete a reliable broadcast is such that $U \leq R/N$, where R is the resynchronization interval and N is the number of nodes in the system.

Intuitively, A1 states that clocks are initially δ_0-synchronized to each other. A scheme to achieve this initial synchronization is described later. A2 states that if a node is nonfaulty, then so is its clock. In other words, no distinction is made between a faulty clock and a faulty node. A3 states that there is an upper bound U on the time to complete a reliable broadcast. In A4, $c_p^{(i)}(T_0 + \Delta_{qp} + Q)$ is the real time at which the message is received at p and $c_q^{(i)}(T_0)$ is the real time at which the message was sent from q. Since Q is the total delay incurred by the message in transit, $c_p^{(i)}(T_0 + \Delta_{qp} + Q) - Q - c_q^{(i)}(T_0)$ is the error in estimating the skew at p based on this message. Therefore, A4 states that if all CP's adhere to the scheme for broadcasting and relaying the clock messages, then the error in estimating the skew is bounded. This assumption is reasonable because the errors occur, in most part, due to: 1) the transit delay incurred at an intermediate node is measured using a discrete logical clock as opposed to a continuous clock, and 2) the logical clock at an intermediate node drifts from real time while measuring the delay incurred at a node. The existence of a bound then follows from the observation that the time required for a broadcast is bounded, the drift rates of nonfaulty clocks are bounded, and the nonfaulty clocks have a known finite resolution.

A5 states that a broadcast initiated by a CP will always complete before the next CP initiates its broadcast. As a result, in the proposed scheme, at most one CP will be broadcasting its clock value at any given time. It is shown later in this paper that for typical values of all relevant parameters in the system, A5 is not a very restrictive assumption.

Given these assumptions, it can be proved that the nonfaulty clocks will remain synchronized throughout the operation of the system. Due to the inherent complexity of the notation, an informal explanation of the lemmas is first presented. The lemmas are similar to the ones in [13]. The differences arise mainly because the worst case skews in the proposed scheme are shown to be less than the maximum transit delay during a broadcast and because of the modeling assumption that a CP may have to use a few relay nodes to convey its clock value to other CP's. A few additional lemmas are required to show that the faulty relay nodes cannot prevent synchronization.

Lemma 1 states that for a good clock Π units of clock time elapse in approximately Π units of real time. Lemma 2 deals with the perceived skew between two nonfaulty nodes, stating that there is a bound on the estimated skew between two nonfaulty nodes. This bound is used in Lemma 6 to prove that the impact of faulty nodes can be controlled. In Lemma 4 it is shown that it does not matter (to a limited extent) at what time the CP at a nonfaulty node initiates a broadcast of its clock value in a resynchronization interval. Lemma 5 states that the difference between the skew perceived at p between p and r and the skew perceived at q between q and r is bounded when p, q, and r are nonfaulty. Theorem 1 and Lemma 3 deal with the case when some of the relay nodes are faulty. Finally, Theorem 2 combines these lemmas to prove the correctness of the proposed scheme.

Lemma 1: If clock synchronization condition S1 holds for i and node p is nonfaulty up to time $T^{(i+2)}$, then for any Π such that $|\Pi| < R$ and any T in $R^{(i)}$:

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

Proof: Follows easily from A2. ■

Lemma 2: Suppose S1 holds for i, and nodes p and q are nonfaulty up to time $T^{(i+1)}$. If the CP at q sends its clock value to the CP at p through several nonfaulty relay nodes, then skew Δ_{qp} estimated at p for this message using the proposed algorithm is such that

$$|\Delta_{qp}| \le \frac{\delta + \epsilon + \frac{\rho}{2}\cdot U}{1 - \frac{\rho}{2}}.$$

Proof: Let T_0 be the time according to q's clock at which it sent the message to p. Also let Q denote the total delay encountered by the message as indicated by words $W_1, \cdots, W_5$. Then by repeated application of triangle inequality we get

$$\begin{aligned}|\Delta_{qp}| &= |c_p^{(i)}(T_0) - c_p^{(i)}(T_0 + Q + \Delta_{qp}) + Q \\ &\quad + c_p^{(i)}(T_0 + Q + \Delta_{qp}) - c_p^{(i)}(T_0) - Q - \Delta_{qp}| \\ &\le |c_p^{(i)}(T_0) - c_p^{(i)}(T_0 + Q + \Delta_{qp}) + Q| \\ &\quad + |c_p^{(i)}(T_0 + Q + \Delta_{qp}) - [c_p^{(i)}(T_0) + Q + \Delta_{qp}]| \\ &\le |c_p^{(i)}(T_0) - c_q^{(i)}(T_0)| \\ &\quad + |c_q^{(i)}(T_0) - c_p^{(i)}(T_0 + Q + \Delta_{qp}) + Q| \\ &\quad + \frac{\rho}{2}|Q| + \frac{\rho}{2}|\Delta_{qp}|.\end{aligned}$$

This implies

$$\begin{aligned}\left(1 - \frac{\rho}{2}\right)\cdot|\Delta_{qp}| &\le |c_p^{(i)}(T_0) - c_q^{(i)}(T_0)| \\ &\quad + |c_q^{(i)}(T_0) - c_p^{(i)}(T_0 + Q + \Delta_{qp}) + Q| \\ &\quad + \frac{\rho}{2}|Q| \\ &\le \delta + \epsilon + \frac{\rho}{2}|Q| \qquad \text{(hypothesis and A4)} \\ &\le \delta + \epsilon + \frac{\rho}{2}U \qquad \text{(by A3).}\end{aligned}$$

Hence proved. ■

Informally, Lemma 2 states that the Δ_{qp} calculated from a clock message is bounded if all the intermediate nodes through which the message was relayed are nonfaulty. However, all the intermediate nodes may not necessarily be nonfaulty and a CP may not be able to identify the messages that have been relayed through faulty intermediate nodes. Similarly, A4 holds only if all the intermediate nodes are nonfaulty. These two problems are overcome by using a reliable broadcast that delivers multiple copies of a clock message to all CP's through node-disjoint paths. A receiving CP estimates the skew Δ_{qp} from the multiple copies it has received, as shown in CLOCK_RECEIVE. In Theorem 1 and its corollary, it is shown that the estimated Δ_{qp} satisfies the bound specified in Lemma 2. Furthermore, the bound specified in A4 can be changed to account for the fact that the selected copy may have passed through some faulty nodes.

Theorem 1: Suppose q uses CLOCK_RECEIVE to estimate the skew between p's clock and its own clock based on the messages it receives in a reliable broadcast initiated from p. Then, the estimated skew is either equal to the computed skew from a copy that was relayed only through nonfaulty nodes, or there are two copies that have been relayed only through nonfaulty nodes such that the estimated skew lies between the computed skews for these two copies.

Proof: Since the system is assumed to have a maximum of m faults, and since the reliable broadcast tries to deliver $2m+1$ copies, at least $m+1$ copies are relayed through nonfaulty nodes. By A3 these $m+1$ copies will be delivered within U time units. Consequently, the CP at q can unambiguously determine the number of copies it will receive.

Suppose $2m+1-t$ copies are received at q for some $0 \le t \le m$. There are two possible cases: a) $t < m$ and b) $t = m$. When $t < m$ there are more than $m+1$ copies. Since the skew estimated at q is the $(m+1)$th largest value, there are m computed skews smaller than and greater than the estimated skew. The statement of theorem follows from this observation and the hypothesis that there are only m faults in the system.

When $t = m$ all the copies that have been relayed through faulty nodes are lost. Consequently, all the copies that have been received were relayed only through nonfaulty nodes. Hence proved. ■

Corollary 1: Suppose p uses CLOCK_RECEIVE to estimate the skew, Δ_{qp}, between q's clock and its own clock based on the messages its receives in a reliable broadcast initiated by q. Then

$$|\Delta_{qp}| \le \frac{\delta + \epsilon + \frac{\rho}{2}\cdot U}{1 - \frac{\rho}{2}}.$$

Proof: Follows from Theorem 1 and Lemma 2. ■

Lemma 3: Suppose S1 holds for i, and nodes p and q are nonfaulty up to time $T^{(i+1)}$. Also suppose the CP at q uses a reliable broadcast at time T_0 according to its clock to convey q's clock value and p uses CLOCK_RECEIVE and CLOCK_CORRECTION to estimate the skew Δ_{qp}. If Q is the total transit delay as computed for the copy of the clock message that was used to estimate the skew, then

$$|c_p^{(i)}(T_0 + Q + \Delta_{qp}) - Q - c_q^{(i)}(T_0)| \le \hat{\epsilon} \equiv \epsilon + \frac{\rho(\delta + \epsilon + U)}{\left(1 - \frac{\rho}{2}\right)}.$$

Proof: If the copy used to estimate the skew was relayed only through nonfaulty nodes, then the lemma follows from A4. So the more interesting case is when it was relayed through at least one faulty node. In this case, it follows from Theorem 1 that there are at least two copies relayed only through nonfaulty nodes, say through paths P_1 and P_2, such that the estimated skew lies between the skews computed from these two copies. Let Q_1 and Q_2 be the accumulated transit delays, and let $\Delta_{qp}(P_1)$ and $\Delta_{qp}(P_2)$ be the computed skews from the clock messages that were relayed through paths P_1 and P_2, respectively.

Without loss of generality, we can assume that $\Delta_{qp}(P_1) \le \Delta_{qp} \le \Delta_{qp}(P_2)$. Adding and subtracting a few terms to the inequality $\Delta_{qp}(P_1) \le \Delta_{qp}$ we get

$$c_p^{(i)}(T_0) + Q + \Delta_{qp}(P_1) \le c_p^{(i)}(T_0) + Q + \Delta_{qp} - c_p^{(i)}(T_0 + Q + \Delta_{qp}) + c_p^{(i)}(T_0 + Q + \Delta_{qp}).$$

Using Lemma 1, A4, Corollary 1, and a few substitutions, the above equation can be shown to be equivalent to

$$c_p^{(i)}(T_0) - c_q^{(i)}(T_0) - \frac{\rho(\delta + \epsilon + U)}{2\left(1 - \frac{\rho}{2}\right)} + \Delta_{qp}(P_1) \le c_p^{(i)}(T_0 + Q + \Delta_{qp}) - Q - c_q^{(i)}(T_0).$$

Now by adding and subtracting a few terms on the left-hand side and then using Lemma 1 and A4, we get

$$-\epsilon - \frac{\rho(\delta + \epsilon + U)}{\left(1 - \frac{\rho}{2}\right)} \le c_p^{(i)}(T_0 + Q + \Delta_{qp}) - Q - c_q^{(i)}(T_0).$$

This proves one side of the required inequality. To prove the other side, we can start with $\Delta_{qp} \le \Delta_{qp}(P_2)$ and show that

$$c_p^{(i)}(T_0 + Q + \Delta_{qp}) - Q - c_q^{(i)}(T_0) \le \epsilon + \frac{\rho(\delta + \epsilon + U)}{\left(1 - \frac{\rho}{2}\right)}.$$

Hence, the lemma follows. ■

Lemma 4: Suppose S1 holds for i, and nodes p and q are nonfaulty up to time $T^{(i+1)}$. Also suppose that the CP at q uses a reliable broadcast algorithm and that the CP at p estimates the skew Δ_{qp} as in CLOCK_RECEIVE and CLOCK_CORRECTION. If Q is the total transit delay as computed from the copy used to estimate the skew, then for any Π such that $|\Pi| < R$ and any T in $R^{(i)}$

$$|c_p^{(i)}(T + \Pi + Q + \Delta_{qp}) - Q - c_q^{(i)}(T + \Pi)| \le \hat{\epsilon} + 2\rho R.$$

Proof: Let T_0 be as in Lemma 3. Then

$$\begin{aligned} &|c_p^{(i)}(T + \Pi + Q + \Delta_{qp}) - Q - c_q^{(i)}(T + \Pi)| \\ &= |c_p^{(i)}(T_0 + T + \Pi + Q + \Delta_{qp} - T_0) - Q - c_q^{(i)}(T_0 + T + \Pi - T_0)| \\ &\le |c_p^{(i)}(T_0 + Q + \Delta_{qp}) - Q - c_q^{(i)}(T_0)| + \rho|T - T_0 + \Pi| \\ &\le \hat{\epsilon} + 2\rho R \qquad \text{(from Lemma 3).} \end{aligned}$$ ■

Lemma 5: Suppose S1 holds for i, and nodes p, q, and r are nonfaulty up to time $T^{(i+2)}$. Also suppose the CP at r uses a reliable broadcast to convey its clock value, and suppose p and q use CLOCK_RECEIVE and CLOCK_CORRECTION to estimate the skew between their clock and r's clock. Let $\bar{\Delta}_{rp}(a)$ and $\bar{\Delta}_{rq}(b)$ be the estimated skews at p and q, respectively, at the end of line 9 in CLOCK_CORRECTION with threshold $\Delta = (\delta + \epsilon + \rho/2 \cdot U)/(1 - \rho/2)$. Let $Q^{(a)}$ and $Q^{(b)}$ denote the total transit delay computed at p and q, respectively, from the copy used to estimate the skew. Then for any T in $R^{(i)}$

$$|c_p^{(i)}(T) + \bar{\Delta}_{rp}(a) - [c_q^{(i)}(T) + \bar{\Delta}_{rq}(b)]| \le 2(\hat{\epsilon} + 2\rho R) + \frac{\rho}{\left(1 - \frac{\rho}{2}\right)}(\delta + \epsilon + U).$$

Proof: It follows from Corollary 1 that when Δ is chosen as in the hypothesis, $|\Delta_{rq}(a)| \le \Delta$ and $|\Delta_{rq}(b)| \le \Delta$. So $\bar{\Delta}_{rp}(a) = \Delta_{rp}(a)$ and $\bar{\Delta}_{rq}(b) = \Delta_{rq}(b)$, where $\Delta_{rp}(a)$ and $\Delta_{rq}(b)$ are the estimate of the skews at the end of line 6 of CLOCK_CORRECTION. For notational simplicity, let $\Delta_{rp} \equiv \Delta_{rp}(a)$ and $\Delta_{rq} \equiv \Delta_{rq}(b)$. Then

$$\begin{aligned} &|c_p^{(i)}(T) + \bar{\Delta}_{rp}(a) - [c_q^{(i)}(T) + \bar{\Delta}_{rq}(b)]| \\ &= |c_p^{(i)}(T) + \Delta_{rp} + Q^{(a)} - Q^{(a)} + Q^{(b)} - [c_q^{(i)}(T) + \Delta_{rq} + Q^{(b)}]| \\ &\le |c_p^{(i)}(T + \Delta_{rp} + Q^{(a)}) - c_q^{(i)}(T + \Delta_{rq} + Q^{(b)}) + Q^{(b)} - Q^{(a)}| \\ &\quad + \frac{\rho}{2}|\Delta_{rp} + Q^{(a)}| + \frac{\rho}{2}|\Delta_{rq} + Q^{(b)}| \\ &\le |c_p^{(i)}(T + \Delta_{rp} + Q^{(a)}) - c_q^{(i)}(T + \Delta_{rq} + Q^{(b)}) + Q^{(b)} - Q^{(a)}| + \frac{\rho}{\left(1 - \frac{\rho}{2}\right)}(\delta + \epsilon + U) \\ &\le |c_p^{(i)}(T + \Delta_{rp} + Q^{(a)}) - c_r^{(i)}(T) - Q^{(a)}| + |c_q^{(i)}(T + \Delta_{rq} + Q^{(b)}) - c_r^{(i)}(T) - Q^{(b)}| \\ &\quad + \frac{\rho}{\left(1 - \frac{\rho}{2}\right)}(\delta + \epsilon + U) \\ &\le 2(\hat{\epsilon} + 2\rho R) + \frac{\rho}{\left(1 - \frac{\rho}{2}\right)}(\delta + \epsilon + U) \quad \text{from Lemma 4.} \end{aligned}$$ ■

Lemma 6: Suppose S1 holds for i, and nodes p and q are nonfaulty up to time $T^{(i+2)}$. Also suppose that p and q use CLOCK_RECEIVE and CLOCK_CORRECTION to estimate the skew between their clock and the clock at node r. Let $\bar{\Delta}_{rp}(a)$ and $\bar{\Delta}_{rq}(b)$ be the estimated skews at p and q, respectively, at the end of line 9 in CLOCK_CORRECTION. Then for any T in $R^{(i)}$

$$|c_p^{(i)}(T) + \bar{\Delta}_{rp}(a) - [c_q^{(i)}(T) + \bar{\Delta}_{rq}(b)]| \leq \delta + 2\Delta.$$

Proof: By the hypothesis that S1 holds for i, we have

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

Since $\bar{\Delta}_{rp}$ and $\bar{\Delta}_{rq}$ are by definition no larger than Δ, the result follows. ■

Theorem 2: If A1–A5 hold and if

a) $3m < N$ and the interconnection network is $2m+1$ connected

2) $$\delta > \max\left\{\frac{(N-m)}{N}\left(2(\hat{\epsilon}+2\rho R)+\frac{\rho(\delta+\epsilon+U)}{\left(1-\frac{\rho}{2}\right)}\right) + \frac{m}{N}(\delta+2\Delta)+\rho(R+\Delta),\ \delta_0+\rho R\right\}.$$

then the algorithm presented in the previous section satisfies Conditions S1 and S2 with $\Sigma = \Delta$.

Proof: Let Δ_p and Δ_q be the correction computed at p and q, respectively, at the end of line 12 in CLOCK_CORRECTION. Condition S2 holds because the correction to the local clock is the average of N terms, each less than Δ. Condition S1 can be proved by induction on i. For $i = 0$, A1 implies that two nonfaulty clocks that are synchronized to within δ_0 at time $T^{(0)}$ will remain synchronized to within $\delta_0 + \rho R$ at time $T^{(1)} = T^{(0)} + R$. Condition S1 follows from A1 and the hypothesis.

Now suppose that S1 holds for i. We need to prove that it holds for $i+1$. Assume that p and q are both nonfaulty up to time $T^{(i+2)}$. For clarify of presentation, we will use $\bar{\Delta}_{rp}$ and $\bar{\Delta}_{rq}$ without explicitly specifying the nodes through which the clock values were relayed from r to p and q. Also let T denote $T^{(i+1)}$. Then for any T' in $R^{(i+1)}$ we have

$$|c_p^{(i+1)}(T') - c_q^{(i+1)}(T')|$$
$$\leq |c_p^{(i+1)}(T) - c_q^{(i+1)}(T)| + \rho R \qquad \text{by A2}$$
$$= |c_p^{(i)}(T+\Delta_p) - c_q^{(i)}(T+\Delta_q)| + \rho R$$

from the algorithm

$$|c_p^{(i)}(T) + \Delta_p - [c_q^{(i)}(T) + \Delta_q]| + \rho R + \frac{\rho}{2}(\Delta_p + \Delta_q)$$

from Lemma 1

$$= \left|\frac{1}{N}\sum_{r=1}^{N}(c_p^{(i)}(T) + \bar{\Delta}_{rp} - [c_q^{(i)}(T) + \bar{\Delta}_{rp}]\right| + \rho(R+\Delta)$$
$$\leq \frac{1}{N}\sum_{r=1}^{N}|(c_p^{(i)}(T) + \bar{\Delta}_{rp} - [c_q^{(i)}(T) + \bar{\Delta}_{rp}]| + \rho(R+\Delta)$$
$$\leq \frac{(N-m)}{N}\left(2(\hat{\epsilon}+2\rho R)+\frac{\rho(\delta+\epsilon+U)}{\left(1-\frac{\rho}{2}\right)}\right) + \frac{m}{N}(\delta+2\Delta)+\rho(R+\Delta).$$ ■

E. Initial Synchronization

There are several ways of ensuring that A1 is satisfied. The simplest scheme is to assume that all nodes are nonfaulty at startup and select one of them as a master. After making sure that all nodes are operational, the master can broadcast its clock value to all other nodes. The CP's can use the scheme proposed in this paper to correct the local clock to the value received from the master.

If it is not possible to ensure that all nodes are nonfaulty at startup, then a more complicated scheme is necessary for initial synchronization. First, use an existing initial synchronization algorithm such as the ones in [7] and [15]. At the end of this phase, the worst case skew will be larger than the difference between the maximum and the minimum message transit delay. To reduce it further, repeatedly use the proposed algorithm until the skew is below the required limit. Before each repetition, calculate the worst case skew at that iteration and set the threshold Δ to that value. This scheme will converge exponentially as long as the total number of nodes $N > 3m+1$, which will usually hold in large distributed systems. The proof that this will converge follows easily from the proof of Theorem 2.

IV. Discussion and Numerical Examples

The expression for δ in Theorem 2 can be simplified by making certain approximations that hold in most distributed systems, namely, $\rho \cdot U \ll \delta$, $\Delta \ll R$, $\epsilon \ll U$, and $\rho \cdot U \ll \epsilon$. These approximations are reasonable, since for most practical applications, ϵ is around 20–30 μs, ρ is around 10^{-6}, U is around 200–300 ms, and R is on the order of seconds. From the exact expression, it is also clear that $\delta \ll U$. Using these approximations, the second hypothesis in Theorem 2 can be restated as

$$\delta > \max\left\{\frac{2(N-m)(\epsilon+2\rho R)+2m\epsilon+\rho RN}{(N-3m)},\ \delta_0+\rho R\right\}. \tag{4.1}$$

One can now characterize δ^*, the minimum worst case skew that can be guaranteed in any given system using our scheme, i.e., δ^* is the minimum value of δ that satisfies (4.1) given N, m, ρ, ϵ, and U. Since N, m, ρ, ϵ, and U are characteristics of the system, the only parameter that can be controlled is R. It is easy to verify that the minimum δ which satisfies the (4.1) increases with R. From A5 we know $R \geq N \cdot U$ and therefore,

$$\delta^* = \max\left\{\frac{2(N-m)(\epsilon+2\rho NU)+2m\epsilon+\rho N^2U}{(N-3m)},\ \delta_0+\rho NU\right\}. \tag{4.2}$$

Clearly, $\delta^* \ll U$ for most practical systems. As a result, the minimum worst case skew that can be guaranteed is substantially less than the minimum worst case skew guaranteed by other software synchronization algorithms [7], [13], [15], [20] where the worst case skews are greater than U. This implies that the skews which can be guaranteed by our scheme are very tight as compared to existing software synchronization algorithms.

In addition to this significant advantage, another major ad-

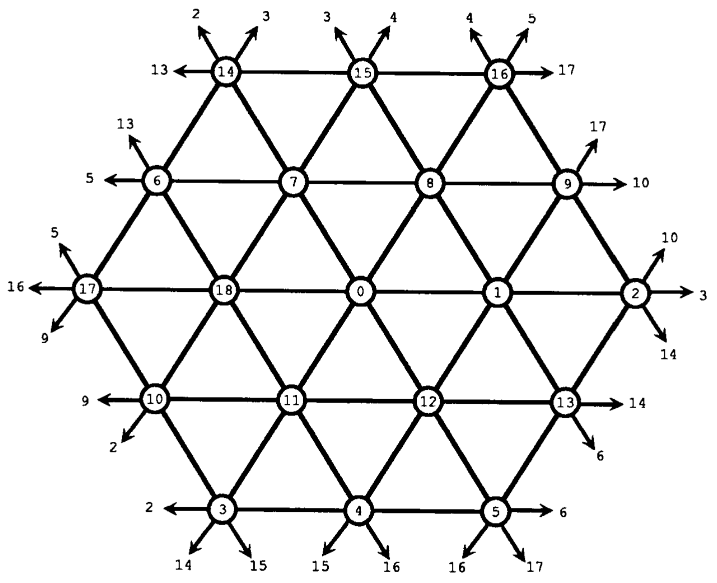


Fig. 5. A C-wrapped hexagonal mesh of dimension 3.

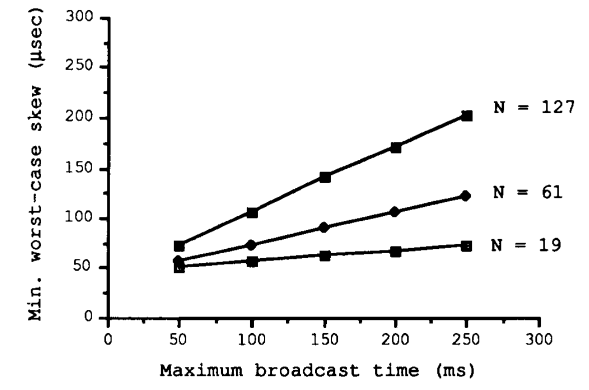
vantage of our scheme is that δ^* increases gradually with U. This should be contrasted to an almost linear increase in the minimum worst case skew with respect to U in existing software synchronization algorithms. Some of these aspects are illustrated with examples for C-wrapped hexagonal mesh and hypercube topologies.

A C-wrapped hexagonal mesh topology [1], [21] is a regular, homogeneous graph in which each node has six neighbors. The graph can be visualized as a simple hexagonal mesh with wrap links added to the nodes on the periphery. A simple hexagonal mesh looks like a set of concentric hexagons with a central node, where each hexagon has one more node on each edge than the one immediately inside of it. It can be defined succinctly as follows.

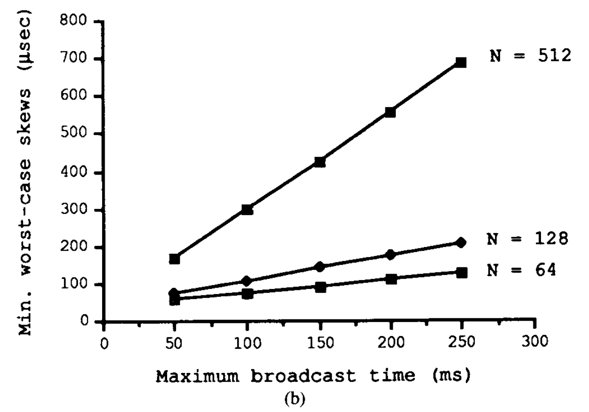
Definition 5: A *C-wrapped hexagonal mesh* of dimension e is comprised of $3e(e-1)+1$ nodes, labeled from 0 to $3e(e-1)$, such that each node s has six neighbors $[s+1]_{3e^2-3e+1}$, $[s+3e-1]_{3e^2-3e+1}$, $[s+3e-2]_{3e^2-3e+1}$, $[s-1]_{3e^2-3e+1}$, $[s-3e+1]_{3e^2-3e+1}$, and $[s-3e+2]_{3e^2-3e+1}$, where $[a]_b$ denotes $a \bmod b$.

Fig. 5 shows a C-wrapped hexagonal mesh of dimension 3. This topology is currently being used for the construction of two experimental distributed systems: the HARTS at the Real-Time Computing Laboratory at The University of Michigan [1], and the Mayfly systems at Hewlett Packard Research Laboratories [3]. An attractive feature of this topology is that it supports a simple reliable broadcast mechanism that is resilient to two Byzantine faults with minimal overhead [8].

A hypercube topology is also a regular, homogeneous graph. Each node in an n-dimensional hypercube, Q_n, has n neighbors. A Q_n also supports a simple reliable broadcast algorithm that delivers n copies of the message through node-disjoint paths [16]. It has been used as an interconnection topology in both research [18] and commercial systems by Intel, NCUBE, Floating Point Systems, Ametek, Thinking Machine, to name a few. It can be defined formally as follows.



(a)

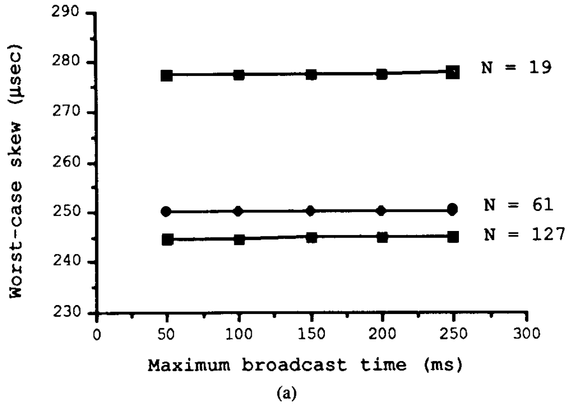


(b)

Fig. 6. δ^* versus U when $m = 2$, $\rho = 10^{-6}$, $\epsilon = 20$ μs. (a) Hexagonal mesh. (b) Hypercube.

Definition 6: A *hypercube* of dimension n is comprised of 2^n nodes, labeled from 0 to $2^n - 1$, such that two nodes are neighbors if and only if the binary representation of their labels differ by one and only one bit.

Fig. 6 shows the variation in δ^* with respect to U for the C-



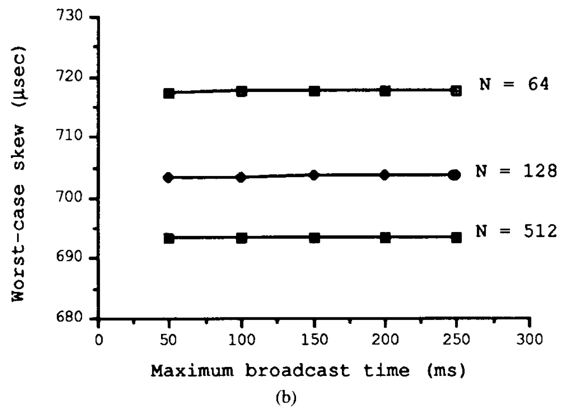


Fig. 7. δ versus U when $R = 40$ s for hexagonal mesh, $R = 130$ s for hypercube. (a) Hexagonal mesh. (b) Hypercube.

wrapped hexagonal mesh and hypercube topologies. Clearly, skews on the order 100 μs can be guaranteed even in large systems using our scheme. When U varies from 50 to 250 ms as in this figure, the minimum worst case skews in [7], [13], [15], and [20] will also vary linearly from 50 to 250 ms, as compared to the 50–700 μs variation in our scheme. Another observation that can be made from the figure is that better skews can be achieved in a C-wrapped hexagonal mesh as compared to a hypercube due to the sparse connectivity of a hypercube in comparison to a hexagonal mesh.[2]

Fig. 7 shows the variation in the worst case skews that can be guaranteed in the above two topologies with respect to the maximum time required for broadcast when the resynchronization interval R is significantly greater than the minimum resynchronization interval necessary for synchronization, i.e., $R > N \cdot U$. This figure illustrates how the worst case skews can be made insensitive to the maximum time required for broadcast at the cost of tightness of synchronization.

V. Conclusion

A hardware-assisted software synchronization scheme was proposed in this paper. The two main advantages of this scheme are 1) skews on the order 100–200 μs can be easily achieved even in large partially-connected distributed systems, 2) the guaranteed worst case skews can be made insensitive to the maximum time required for broadcast. These two aspects are in sharp contrast with the existing software synchronization algorithms. The additional hardware required at each node to obtain these advantages is similar to the support necessary to ensure timely delivery of messages. This kind of support is commonly available in distributed real-time systems.

Another advantage of the proposed scheme is that the broadcast of clock values from different nodes is staggered throughout the resynchronization interval, instead of being lumped at the end of the interval as in most software synchronization algorithms. This avoids a sudden increase in network traffic caused by almost simultaneous broadcast of clock values from all nodes.

A possible drawback of our scheme is that it is only suitable for topologies that support efficient reliable broadcast mechanisms. This is not a serious problem, however, since most fault-tolerant distributed systems require reliable broadcast mechanisms for various other regions. Consequently, the scheme proposed in this paper has high potential for synchronizing large distributed systems.

[2] For those dimensions greater than 6, a hypercube will have a denser connectivity than a C-wrapped hexagonal mesh, though.

References

[1] M.-S. Chen, K. G. Shin, and D. D. Kandlur, "Addressing, routing, and broadcasting in hexagonal mesh multiprocessors," *IEEE Trans. Comput.*, vol. 39, no. 1, pp. 10–18, Jan. 1990.
[2] F. Cristian, "Probabilistic clock synchronization," Tech. Rep. RJ 6432 (62550), IBM Almaden Research Center, Sept. 1988.
[3] A. Davis, R. Hodgson, B. Schediwy, and K. Stevens, "Mayfly system hardware," Tech. Rep. HPL-SAL-89-23, Hewlett-Packard Co., Apr. 1989.
[4] D. Dolev, "The Byzantine generals strike again," *J. Algorithms*, vol. 3, pp. 14–30, 1982.
[5] D. Dolev, M. J. Fischer, R. Fowler, N. A. Lynch, and H. R. Strong, "An efficient algorithm for Byzantine agreement without authentication," *Inform. Contr.*, vol. 52, no. 3, pp. 257–274, Mar. 1982.
[6] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, Apr. 1988.
[7] J. Y. Halpern, B. Simons, R. Strong, and D. Dolev, "Fault-tolerant clock synchronization," in *Proc. 3rd Symp. Principles Distributed Comput.*, 1984, pp. 89–102.
[8] D. D. Kandlur, "Networking issues in distributed real-time systems," Tech. Rep., Real-Time Computing Laboratory, Dep. Elec. Eng. Comput. Sci., Univ. Michigan, Ann Arbor, May 1989.
[9] J. L. W. Kessels, "Two designs of a fault-tolerant clocking system," *IEEE Trans. Comput.*, vol. C-33, no. 10, pp. 912–919, Oct. 1984.
[10] C. M. Krishna, K. G. Shin, and R. W. Butler, "Ensuring fault tolerance of phase-locked clocks," *IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 752–756, Aug. 1985.
[11] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems," *ACM Trans. Programming Languages Syst.*, vol. 6, no. 2, pp. 254–280, Apr. 1984.
[12] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," CSL Tech. Rep. 141, SRI International, 1982.
[13] —, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, no. 1, pp. 52–78, Jan. 1985.
[14] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Programming Languages Syst.*, vol. 4, no. 3, pp. 382–401, July 1982.
[15] J. Lundelius-Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Inform. Computat.*, vol. 77, no. 1, pp. 1–36, 1988.
[16] P. Ramanathan and K. G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. Comput.*, vol. 37, no. 12, pp. 1654–1657, Dec. 1988.
[17] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. 37, no. 7, pp. 867–872, July 1988.

[18] C. L. Seitz, "The Cosmic Cube," *IEEE Trans. Comput.*, vol. C-34, no. 1, pp. 22-33, Jan. 1985.

[19] K. G. Shin and P. Ramanathan, "Clock synchronization of a large multiprocessor system in the presence of malicious faults," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 2-12, Jan. 1987.

[20] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, vol. 34, no. 3, pp. 626-645, July 1987.

[21] K. S. Stevens, "The communication framework for a distributed ensemble architecture," AI Tech. Rep. 47, Schlumberger Research Lab., Feb. 1986.

[22] N. Vasanthavada and P. N. Marinos, "Synchronization of fault-tolerant clocks in the presence of malicious failures," *IEEE Trans. Comput.*, vol. 37, no. 4, pp. 440-448, Apr. 1988.

[23] C. L. Yang and G. M. Masson, "A distributed algorithm for fault diagnosis in systems with soft failures," *IEEE Trans. Comput.*, vol. 37, no. 11, pp. 1476-1480, Nov. 1988.

[24] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Trans. Comput.*, vol. C-36, no. 8, pp. 933-940, Aug. 1987.

P. Ramanathan received the B. Tech degree from the Indian Institute of Technology, Bombay, in 1984, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1986 and 1989, respectively.

From 1984 to 1989 he was a Research Assistant in the Department of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. Currently, he is an Assistant Professor of Electrical and Computer Engineering at University of Wisconsin, Madison. His research interests include fault-tolerant computing, distributed real-time computing, VLSI design, and computer architecture.

Dilip D. Kandlur, for a photograph and biography, see the January 1990 issue of this TRANSACTIONS, p. 18.

Kang G. Shin (S'75-M'78-SM'83), for a photograph and biography, see the January 1990 issue of this TRANSACTIONS, p. 18.