TR-86-48

ETempo: A Clock Synchronization Algorithm for Hierarchical LANs - Implementation and Measurements

by

Satish K. Tripathi and Shu-jen H. Chang

ETempo: A Clock Synchronization Algorithm for Hierarchical LANs - Implementation and Measurements

by

Satish K. Tripathi and Shu-jen H. Chang

This research was supported in part by the National Science Foundation under grant 01R-85-00108.

ETempo: A Clock Synchronization Algorithm for Hierarchical LANs - Implementation and Measurements

Satish K. Tripathi and Shu-jen H. Chang
System Research Center,
UMIACS, and
Department of Computer Science,
University of Maryland
College Park, MD 20742

Abstract

The Tempo clock synchronization algorithm is extended for hierarchical LANs. An implementation of the new algorithm, ETempo, is presented. Because of the unavailability of a two-level hierarchical LAN, it is emulated using a single LAN. The simulation procedures and other design issues are discussed. Behavior of ETempo is measured for a variety of parametric values.

1. Introduction

Synchronization of events is an important research area in a distributed system [Lamp82]. The problem is hard in a geographically distributed system because the communication delays are usually large and unpredictable. In a Local Area Network (LAN), however, such delays are usually small and bounded. In this paper we address the issue of clock synchronization in hierarchical local area networks.

Consider a large automated factory environment. Multiple computers may be needed to coordinate the activities of various instruments and operations. An unfinished part may move from one shop to another for work and this involves coordination and

synchronization among the shops. Time synchronization seems to be one of the essential requirements in the *factories of the future.*

In an automated factory with multiple shops (Figure 1.1), each shop usually has a number of workstations (robots, etc.) and these workstations are connected to each other via a local area network. In addition to the workstations, the LAN also connects local databases and provides connection to the *outside world.* A part under manufacturing process may move from shop to shop before getting completed. Within a shop, the activities at various workstations have to be synchronized to a fine level, so that when a part arrives at a station the station is ready to work on it. When parts move from one shop to another shop we still need workstations of the two shops to be synchronized. Because a larger mechanical movement may be needed for the part to move from one shop to another, the synchronization between two shops may not have to be at the same level as within a shop. We assume that the shops are connected via a *higher level* LAN.

In this paper, we assume that a high level LAN is connected to nodes, which are also connected to LANs at lower levels. We present an extension to the clock synchronization algorithm Tempo [Guse83], *ETempo*, to deal with two level hierarchical LANs. ETempo is implemented on the local Ethernet, which emulates a hierarchical LAN, and behavior of ETempo is measured for a variety of parametric values.

The Tempo algorithm uses a central controller to direct the synchronization activities. This is different from the other existing algorithms that adopt a distributed approach for fault-tolerance. The algorithms proposed by Lamport and Melliar-Smith [Lamp82] require a large number of messages for each synchronization round, which somewhat limits the algorithms' practical use. A more efficient algorithm is proposed by Halpern, Simons and Strong [Halp83], which does not require a majority of nonfaulty

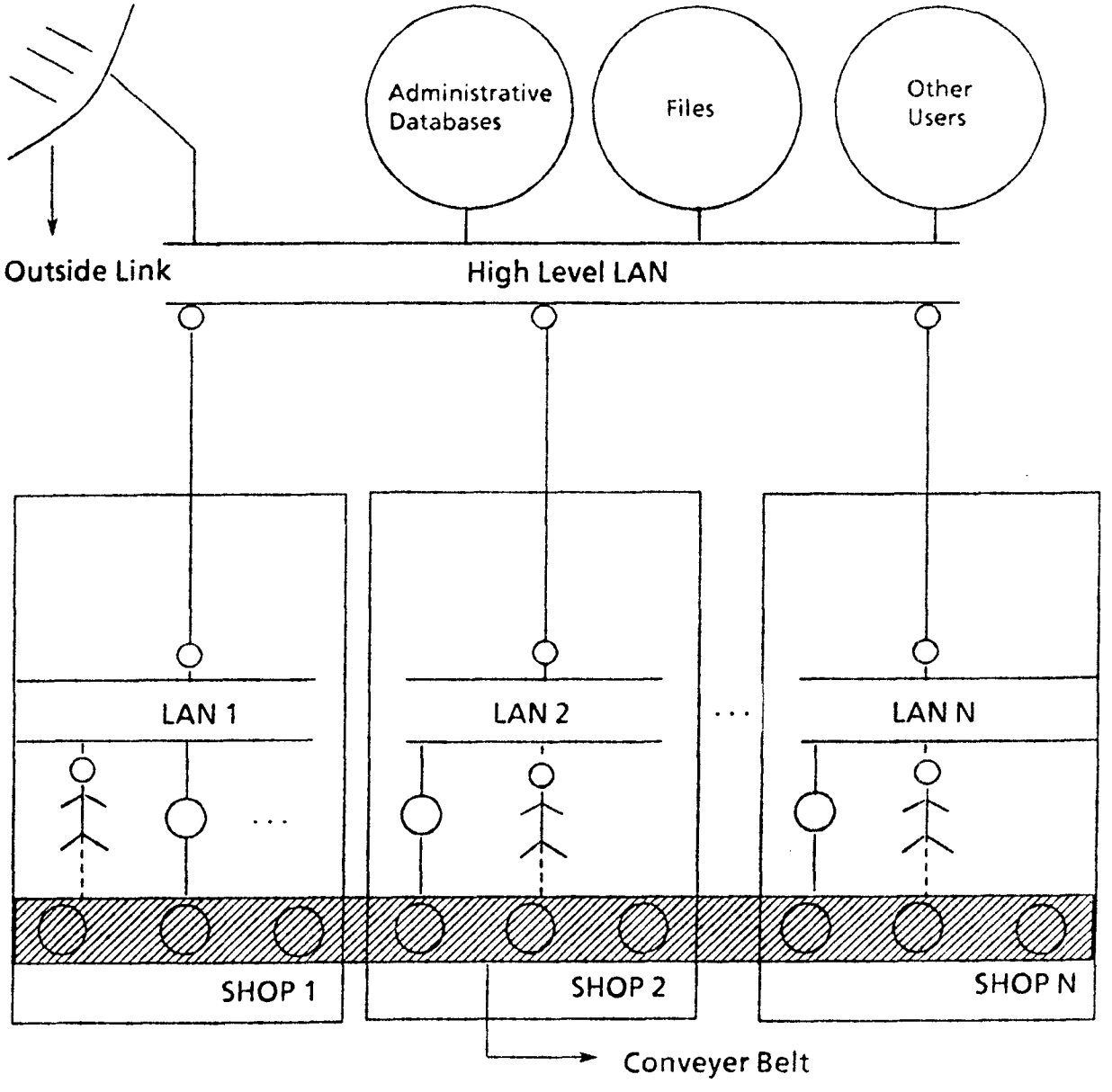

Figure 1.1. Factory of the future.

processes. Authentication is a required feature of this algorithm, and $O(n^2)$ signatured messages are exchanged at each round. Lundelius and Lynch [Lund84] also proposed a fault-tolerant, unauthenticated algorithm. The messages exchanged at each round are compatible with that of [Halp83], however, the size of adjustment is independent of the number of faulty processes as is the case of [Halp83]. In the algorithm devised by Srikanth and Toueg [Srik84], not only logical clocks are synchronized, their rates of drift from the real time can also be minimized to the drift rates of the underlying hardware clocks, thus "optimal accuracy" is achieved. The two algorithms proposed by Marzullo and Owicki [Marz83] use a different theoretical framework and error model, which makes comparisons with other work difficult.

Though the distributed algorithms may handle arbitrary faults, extension of any of these algorithms to work in a hierarchical network seems rather complicated, especially if the number of nodes in the hierarchical network is not small. Since Tempo's centralized approach can easily be incorporated in a hierarchical LAN, it is selected for implementation.

In Section 2, we present the Tempo and ETempo algorithms. Section 3 discusses the implementation issues of ETempo. A simulation methodology for measurement is also outlined. In Section 4, we present measurement results for a variety of parametric values. Finally some concluding remarks are given in Section 5.

2. The Clock Synchronization Algorithm

2.1. The Tempo Algorithm

The Tempo algorithm [Guse83] adopts a centralized approach to synchronize clocks. An elected "master" process polls each "slave" process to estimate the clock

difference between the hosts on which the two processes are running. As the clock differences are measured, a network average clock skew is computed using a fault-tolerant averaging function. This function finds the largest set of clocks that did not differ from each other more than a predefined quantity and averages the time offsets of these clocks. This measure prevents clocks with large drift rates from adversely affecting the other clocks. Each clock is then corrected for an amount equal to the difference between the network average skew and the clock skew between the slave's and the master's clocks. In case the master process fails, detected by slave process not receiving any message from the master over a long period of time, the slave processes will elect a new master. The operations assumed by the master and slave processes are outlined in Figure 2.1.

2.2. The ETempo Algorithm

The Tempo algorithm can easily be adapted to a two-level hierarchical network [Chan86]. Clocks in the individual low-level LAN can be synchronized with the Tempo algorithm, and clocks of the "low-level masters" can be synchronized using the ETempo algorithm. Since each low-level LAN has a master process to direct local synchronization, these low-level masters are spokespersons for their networks and may elect a "master of masters" among themselves to be in charge of synchronization of their clocks using ETempo. As the clocks of the low-level masters are synchronized through this "high-level" synchronization, local clocks of the individual low-level LAN may be asked to synchronize with their master's clock.

There is little procedural difference between the high-level and the low-level synchronization except that a low-level master may wish to synchronize local clocks regularly during the idle period between two high-level synchronizations. The operations of

Figure 2.1. Procedures of Tempo executed by the master and slave processes.

Master process:

```
LOOP
   if this is time to start synchronization, do
      BEGIN
         poll each slave to estimate clock skew from master's clock, delta_i ;
         compute network average clock skew, netdelta ;
         compute correction for each clock where corr_i = netdelta - delta_i ;
         inform slaves to correct their clocks, and correct its own, if needed;
         sleep until next synchronization round;
      END
UNTIL time to quit.
```

Slave process:

```
set timer for election;
LOOP
   if receiving message from master before timeout
      BEGIN
         process the incoming message;
         reset election timer;
      END
   else
      start election protocol and exit;
UNTIL ordered by master to quit.
```

ETempo are sketched in Figure 2.2. As in Tempo, a timeout mechanism is used to detect node failure. If a low-level master fails, the slave processes of that low-level LAN will elect a new master, the new master then synchronizes with the other low-level masters. If the high-level master fails, a new high-level master will be elected among the other low-level masters (They are nontheless the slave processes for this high-level synchronization). However, if a low-level network is disconnected, clocks in that network will not be synchronized until that LAN is reintegrated.

Figure 2.2. Procedures of ETempo as executed by the high-level master and slave processes.

Master process:

```
LOOP
   if it is time to synchronize local clocks, do
      BEGIN
         poll local clocks to determine clock skews;
         compute average clock skew for the low-level LAN;
         compute correction for local clocks and inform local clocks
            to make adjustment;
      END;
   if it is time to synchronize with other low-level masters, do
      BEGIN
         poll each low-level master to determine clock skews;
         compute average clock skew for the high-level LAN;
         compute corrections for low-level masters and ask them to adjust;
         make necessary adjustment and inform local clocks to adjust, too;
      END
   sleep until next high/low level synchronization round, whichever comes first;
UNTIL time to quit.
```

Slave process:

```
set election timer;
LOOP
   if election timer expires
      start election protocol and exit;
   if it is time to synchronize local clocks, do
      BEGIN
         poll local clocks to determine clock skews;
         compute average clock skew for the low-level LAN;
         compute correction for local clocks and make clock adjustment;
      END;
   if receiving message from the high-level master, do
      BEGIN
         process the incoming message;
         reset the election timer;
      END;
UNTIL ordered to quit.
```

When a low-level master corrects its clock due to the high-level synchronization, it has a few options as to whether to propagate this adjustment to other clocks in its

network. First, it may ask local clocks to immediately reset to their master's time. Secondly, it may choose not to propagate the adjustment but waits until the next low-level synchronization round to synchronize the local clocks. Thirdly, it may ask the other clocks to adjust by the same amount. The issue is discussed more thoroughly in [Chan86].

3. The Implementation Issues

3.1. Environment of the Implementation

For a single LAN, we implement the Tempo algorithm on the local 10 MB-Ethernet available at the Computer Science Department of the University of Maryland. The implementation is on three hosts running UNIX 4.3 BSD, namely, GYRE, MIMSY, and TOVE. Table 2.1 gives the machine specifications for these hosts.

A process is created on each of the three hosts for clock synchronization. The implementation is based on the time server, the time "daemon", provided in the UNIX 4.3 BSD, in which the Tempo algorithm is implemented. Since the algorithm uses a centralized scheme, a master process must be elected. Processes communicate with each other through *socket*, which is the basic data structure for communication between hosts. In the implementation, datagram socket is used because the communication protocol supporting the other socket type, the stream socket, would usually incur long com-

Table 2.1 Machine Specifications of the Participating Hosts.

Name	Type of VAX	Physical Memory	Number of Disks	Max. No. of Users
Gyre	750	5M	2	48
Mimsy	780	10M	6	112
Tove	750	5M	2	48

munication delay, especially when messages are lost and retransmissions are necessary.

3.2. The Time Synchronization Protocol

The messages exchanged by the processes are termed the "time synchronization protocol" (TSP) in the time daemon [Guse85a]. With some modifications to accommodate our study, the list of messages are:

TSP_ANY: matches any type of commands,
TSP_ADJTIME: informs recipient to adjust its clock,
TSP_ACK: generic acknowledgement,
TSP_MASTERREQ: asks for master's name,
TSP_MASTERACK: master acknowledges master request,
TSP_MASTERUP: master informs slaves that master is up,
TSP_SLAVEUP: slave indicates to master that slave is up,
TSP_ELECTION: slave announces candidacy for master,
TSP_ACCEPT: non-candidate endorses support for candidate,
TSP_REFUSE: rejects candidate,
TSP_CONFLICT: two or more masters present,
TSP_RESOLVE: resolves conflicts in election,
TSP_QUIT: master informs candidates to quit,
TSP_TIMEREQ: requests for logical clock time,
TSP_TIMEREPLY: reply the request for logical clock time,
TSP_STOP: master announces end of synchronization.

Using a particular sequence of these commands, an election protocol is defined.

3.3. The Election Protocol

There are two instances where the election protocol is used [Guse85b]. First, during system start-up a master process has to be designated to coordinate synchronization activities. Second, a slave process, after a "long" period of wait receives no message from the master, becomes a candidate for the master, and starts the second phase of the election procedure.

The protocol for the first instance, illustrated in Figure 3.1, works as follows. Each process broadcasts a MASTERREQ message at system start-up, if no MASTERACK is received, the process assumes itself to be the master, and checks again if any

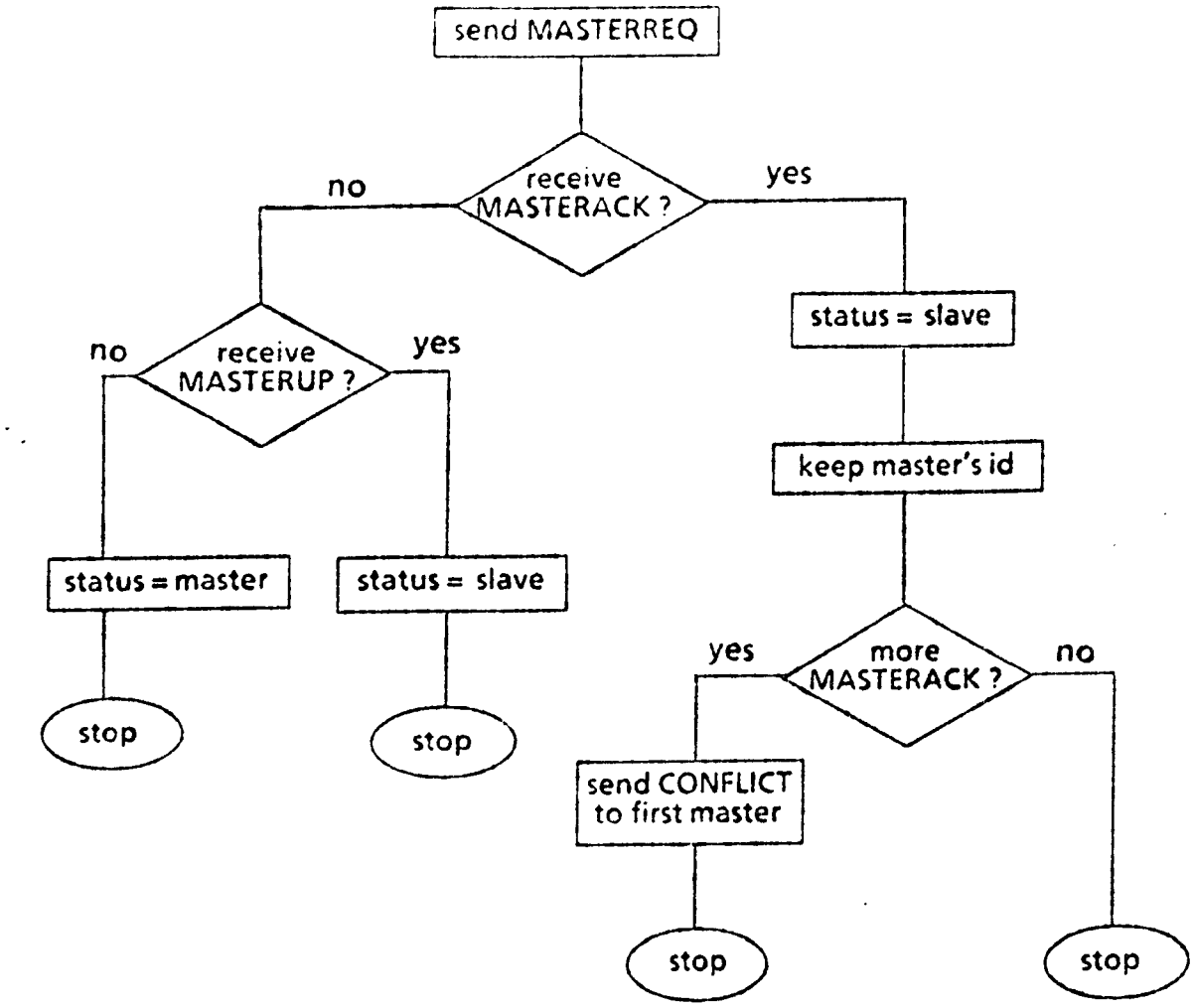

Figure 3.1. Election of the master at system start-up.

MASTERUP message is in its received queue. If no MASTERUP message is received, this slave process becomes the master. On the other hand, if a process receives a MASTERACK message, it keeps the sending master's identification and waits for some specified time for other potential masters to respond. If more than one MASTERACK

message is received, this slave process sends a CONFLICT message to the first master whose identification has been kept. The master process that receives a CONFLICT message gains control over the others by issuing a RESOLVE command. Upon arrival of this RESOLVE command, each potential master acknowledges reception of the message. The master that has sent earlier the RESOLVE message then sends a QUIT message to each of the masters that received the RESOLVE message. When a master receives a QUIT message, it relinquishes the mastership.

The protocol for the second instance, depicted in Figure 3.2, is implemented in the function "election". Election is called by a slave when its election timer expires, i.e., when the slave has not received any message from the master over a prescribed period of time. This slave process becomes a candidate for master and broadcasts an ELECTION message. If a master process is alive and well, it sends a QUIT message to the candidate. If a new master comes up, the candidature is withdrawn. Other slave processes have two choices upon receiving the ELECTION message. They can either refuse or accept the candidate depending on the arrival time of this message. A slave process will accept the first candidate and reject other candidates whose ELECTION messages arrive shortly after the first request. If a candidate receives a REFUSE message, the candidature is withdrawn. If another slave process also announces its candidacy, the candidate receiving this ELECTION message sends a REFUSE to that candidate. The possibility of such collision is reduced by changing the value of the election timer using exponential backoff. A candidate becomes a master only when no other processes object to it.

3.4. The Computation of Clock Skews

A good estimate of the clock skew is important to the accuracy of the algorithm. Since the synchronization interval is likely to be in the order of minutes, it is probable

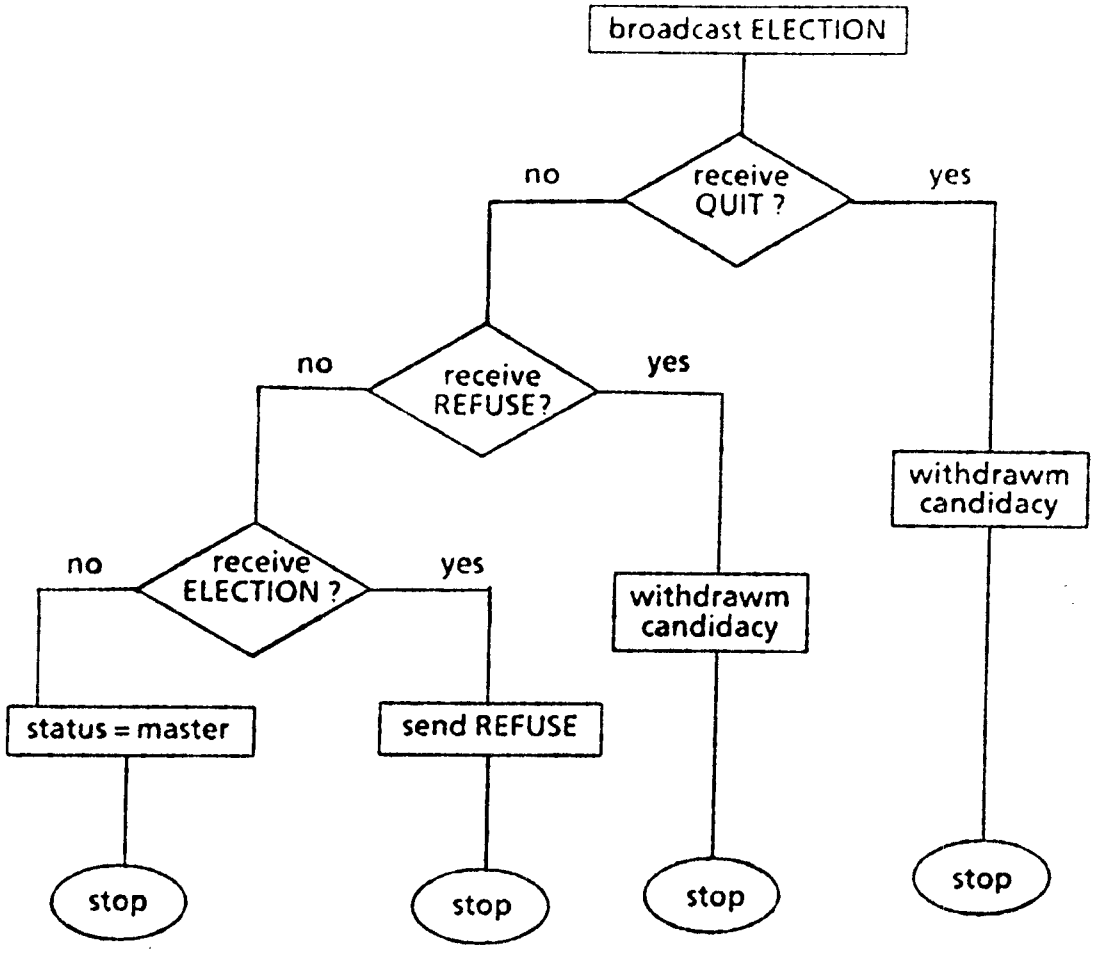

Figure 3.2 . Election started by a slave process at the detection of master node failure.

that both the master and slave processes will be swapped out during the idle period between resynchronizations. For this reason, we do not burden the processes with the time-stamping responsibility, where timestamps are used in measuring clock differences. It may be several seconds before a swapped process can respond to a master's inquiry. Therefore, the timestamping facility of the Internet Control Message Protocol (ICMP) is

used in computing the clock difference.

3.5. The Implementation of ETempo The implementation of ETempo closely resembles with that of Tempo. The messages exchanged and the election protocol described earlier also apply to ETempo. Because a real hierarchical LAN is not available for this study, a two-level LAN is emulated by assuming each of the hosts GYRE, MIMSY, and TOVE to be the master of a single LAN, while the Ethernet connects these three low-level LANs.

3.5.1. The Simulation Procedure

The simulation consists of two steps. In the first step, clocks on the three hosts are synchronized periodically, with clock skews between hosts and adjustments at each round recorded in a "trace" file for use in the second step. These synchronization runs emulate low-level synchronizations that are supposed to interleave with high-level synchronizations. In the second step, master clocks of different LANs are synchronized. The high-level synchronization proceeds in a similar fashion as the low-level synchronization except that physical clocks are not reset in this simulation. Between high-level synchronizations, a low-level master emulates synchronization of local clocks by reading a correction data from the proper trace file and updates its logical clock. Each low-level master keeps track of total adjustments made on its clock, regardless of whether the adjustment is called for from the high-level or the low-level synchronization.

As mentioned in section 2.2, clock adjustments to a low-level master due to high-level synchronization may be passed along to local clocks. In this simulation, local clocks are requested to make the same amount of correction as their master after a high-level synchronization. Since each clock in a low-level LAN is adjusted by the same amount

when its master is asked to reset, the clock skew between a slave and its master remains unchanged, thus, the clock difference and correction data in the trace file is valid and usable.

3.5.2. The Computation of Clock Skews in a Simulated LAN

The computation of clock skews between two simulated low-level masters is slightly more complicated than that for a single LAN, because the clock skews now are differences between the logical clocks.

After a high-level master is elected, it polls the low-level masters in sequence to find the clock skews between them. If $P_m(t)$ denotes the reading of the master's physical clock at real time t, $Corr_m(t)$ denotes the total corrections made on the master's clock up to real time t, likewise, $P_s(t)$ denotes the slave's physical clock reading at real time t, and $Corr_s(t)$ denotes the slave's corrections up to real time t, then the difference of the two logical clocks at real time t, $delta(t)$, is

$$\begin{aligned} delta(t) &= (P_s(t)+Corr_s(t))-(P_m(t)+Corr_m(t)) \\ &= (P_s(t)-P_m(t))+(Corr_s(t)-Corr_m(t)). \end{aligned} \tag{3.1}$$

The difference in the physical clock readings, i.e. $P_s(t)-P_m(t)$, can be computed through exchanged timestamps as used in the low-level synchronization [Chan86]. However, the slaves need to send the cumulative totals of corrections, $Corr_s(t)$, to the master at polling times so that the master can compute the logical clock difference. Messages TSP_TIMEREQ and TSP_TIMEREPLY are designed specifically for this purpose.

4. Measurements and Analysis

4.1. Synchronization Experiments for a Single LAN

The implementation of the Tempo algorithm on our local network allows us to make measurements over several parameters. The results presented here confirm the measurements in [Guse83]. The parameters experimented are:

1. the interval between resynchronizations,
2. the number of times a master polls its slaves in estimating clock skews,
3. the maximum round-trip communication delay allowed,
4. the initial clock difference,
5. the system load during the experiment period.

We study the clock skews measured at each synchronization round to determine the effectiveness of the algorithm in keeping clocks synchronized. In our experiments, clock adjustment is always rounded up/down to the nearest multiple of five milliseconds.

Among these parameters, only the interval between resynchronizations shows a significant influence on the distribution of clock skews. For example, in three experiments where synchronization takes place every two, four, and eight minutes, respectively, the distributions of clock skews between the hosts Tove and Gyre are shown in Figure 4.1. In these three experiments, the process on Tove is the master, and each experiment makes 20 synchronization attempts. The positive clock skews (Figure 4.1) indicate that Gyre's clock runs faster than Tove's. It is interesting to note that with two-minute interval, the most frequently occurring clock difference is five milliseconds, whereas it is 10 milliseconds for the four-minute interval, and 15 milliseconds for the eight-minute interval. When the experiments are repeated with the process on Gyre as the master, the resulting distributions of clock skews (Figure 4.2) are almost a mirror image of Figure 4.1. This suggests that the synchronization algorithm is independent of the placement of the master process.

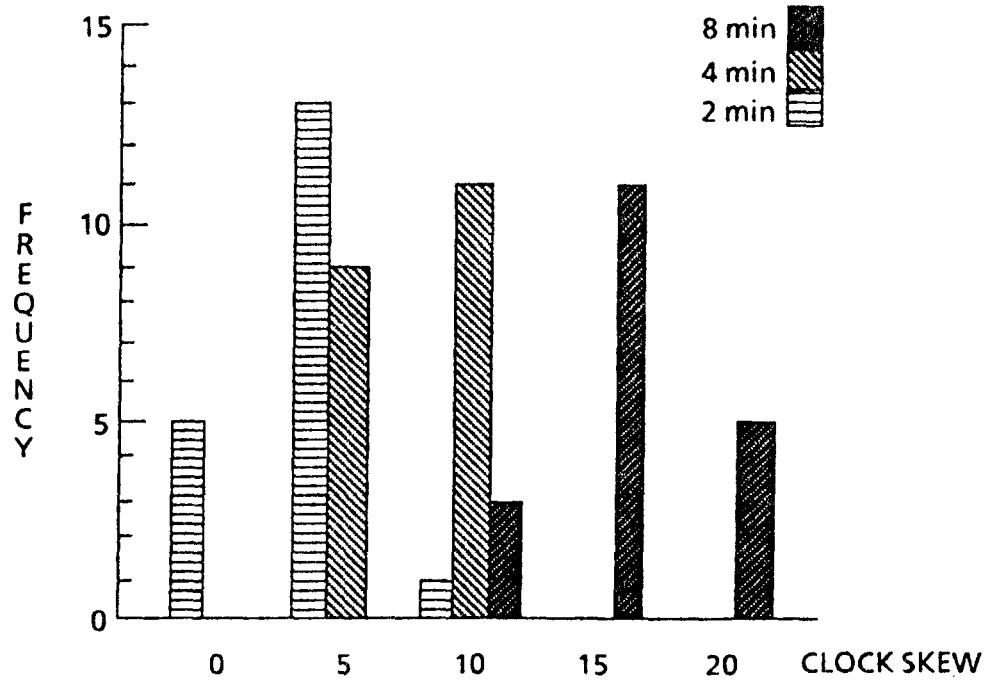

Figure 4.1. Histograms of clock skews (in ms.) between hosts Tove (Master) and Gyre at synchronization interval of two, four, and eight minutes. The abscissa is the frequency of occurrence.

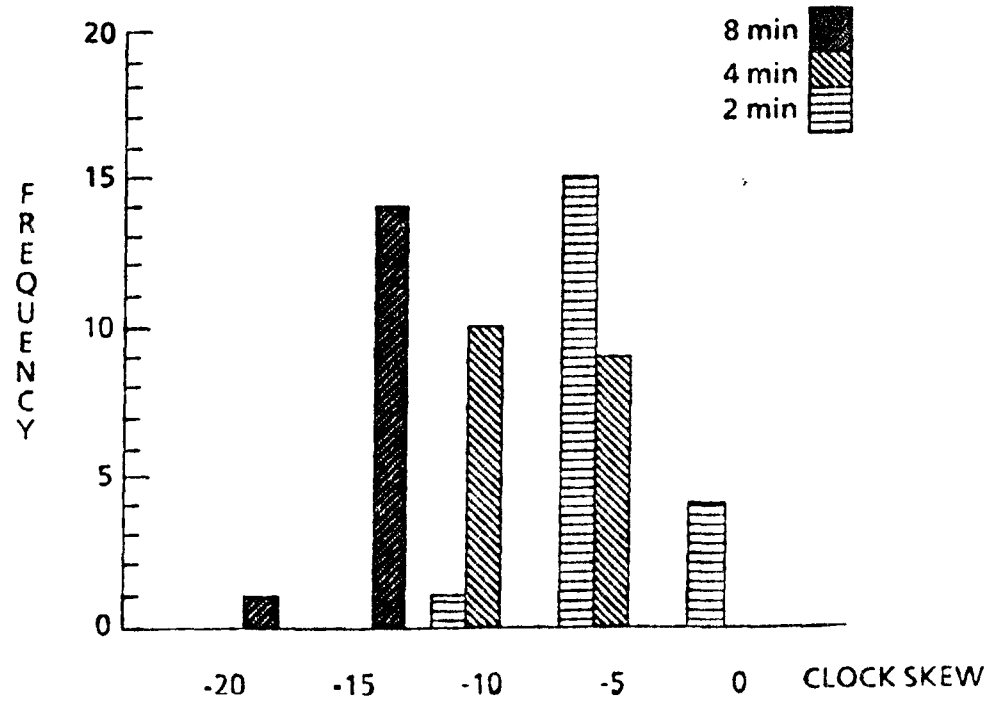

Figure 4.2. Same as Figure 4.1 except for hosts Gyre (Master) and Tove.

To measure clock skews accurately, there should be little variation in the communication delays. Though the variation in the message delays is difficult to measure, we suspect it is not high in our system, because most round-trip delays are shown to be under 20 milliseconds (Table 4.1) even if the maximum round-trip delay is set to 40 milliseconds.

Table 4.1 Round-trip communication delay and its frequency count.

Hosts	Limit	Round-trip Delays (in ms.)			
		10	20	30	40
Mimsy/Tove	20	7	68		
Mimsy/Tove	40	19	48	6	2
Mimsy/Gyre	20	16	59		
Mimsy/Gyre	40	14	55	4	2

On our local network, clock adjustment is performed gradually; during every tick interval of 10 milliseconds, a correction of one millisecond is made, if necessary. Therefore, the time it takes to establish initial synchronization depends on how clock correction is actually implemented. It may seem surprising that the measured clock skews are not sensitive to different system loads. However, because high-priority Internet ICMP time-stamping facility is used by processes to read the clocks of the other hosts directly, delays in awaking idle slave processes do not interfere with the timestamps transmitted, thus, clock skews computed from these timestamps do not seem to be affected by different system loads (Figure 4.3).

4.2. Experiments for a Hierarchical LAN

For the hierarchical network, we organize the experiments into two groups. In one group, we fix the high-level synchronization interval and experiment with different combinations of low-level synchronization intervals. In the second group, the low-level syn-

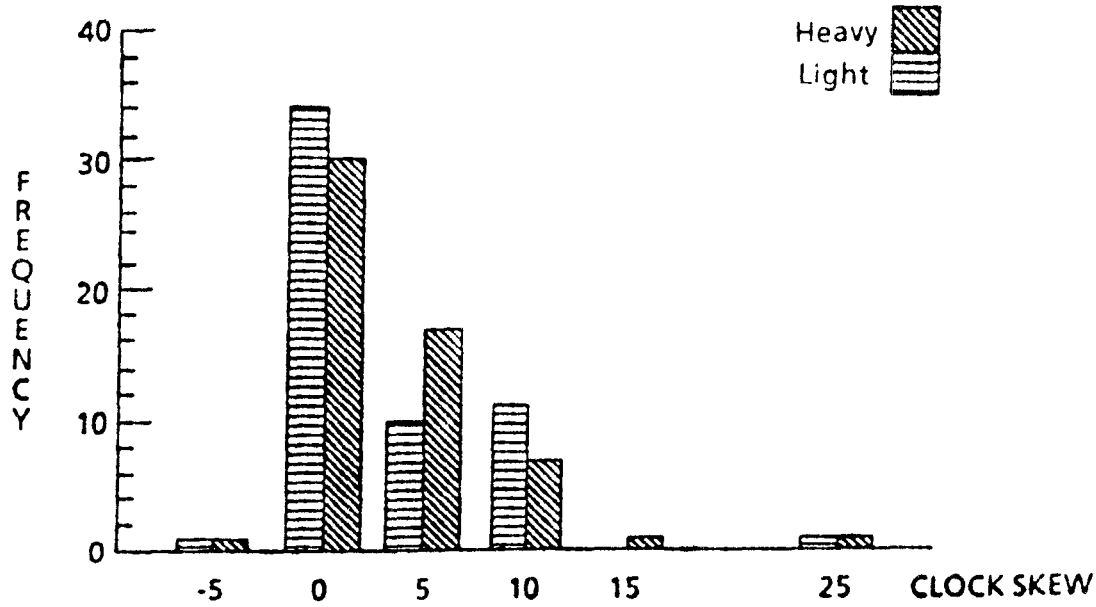

Figure 4.3. Histograms of clock skews (in ms.) between hosts Mimsy (Master) and Gyre from runs executed under different system loads.

chronization interval is fixed and the high-level synchronization interval is varied. Table 4.2 summarizes the set-ups of these experiments.

Table 4.2. Experiment Set-ups for High-level Synchronizations.

Experiment Number	High-level Interval (in min.)	Low-level Interval (min.)			Max. Sync. Count	Length (in hrs.)
		Gyre	Mimsy	Tove		
H1	10	2	2	2	12	2
H2	10	4	4	4	12	2
H3	10	8	8	8	12	2
H4	10	2	8	4	12	2
H5	10	2	2	4	12	2
H6	10	2	2	8	12	2
H7	40	4	4	4	20	13
H8	60	4	4	4	20	20
H9	80	4	4	4	20	26

4.2.1. Measurements on Low-level Synchronization Intervals

In a hierarchical network, each low-level LAN may use different synchronization interval for low-level synchronizations. In Experiments H1-H6 (Table 4.2), the intervals of the high-level synchronizations are kept at ten minutes while the low-level synchronization intervals are allowed to vary from two to eight minutes. For Experiments H1-H3, the distributions of clock skews between low-level masters Gyre and Tove are shown in Figure 4.4. The distributions are difficult to compare because there does not exist a characteristic pattern that can be identified with each of the three experiments.

These results are not surprising considering that after a high-level synchronization, there are two factors causing clocks to deviate (Because we only measure clock

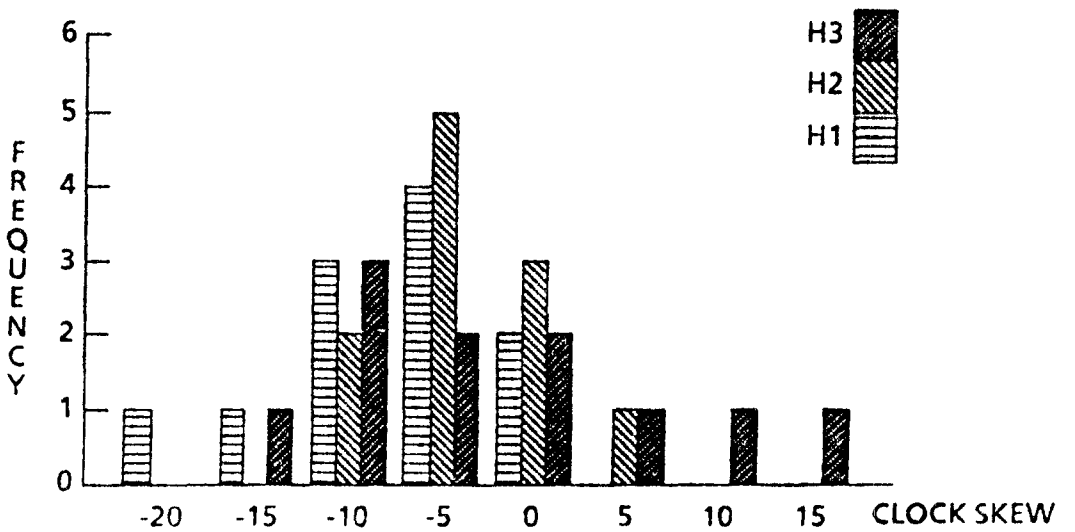

Figure 4.4. Histograms of clock skews between low-level masters Gyre and Tove of Experiments H1, H2, and H3 (See Table 4.2 for parameter set-ups).

differences at each round of high-level synchronizations, clock skews measured at such instances thus become control points for later comparisons). One factor is the drifts between clocks over the ten-minute interval between high-level resynchronizations. The other is the total corrections made on each of the two clocks during the same period from low-level synchronizations within the individual LAN. The time offset between any two clocks in a hierarchical LAN is bounded by (See [Chan86]):

$$
\begin{aligned}
& C_i(t_{HP}) - C_j(t_{HP}) + Corr_{Hi} - Corr_{Hj} - (\delta_i + \delta_j)(t - t_{HP}) + Corr_{Li} - Corr_{Lj} \\
& \leq C_i(t) - C_j(t) \\
& \leq C_i(t_{HP}) - C_j(t_{HP}) + Corr_{Hi} - Corr_{Hj} + (\delta_i + \delta_j)(t - t_{HP}) + Corr_{Li} - Corr_{Lj}, \qquad (4.1)
\end{aligned}
$$

where $C_i(t)$ represents the reading of clock i at real time t, t_{HP} denotes the time when the high-level master starts polling a low-level master for the high-level synchronization, δ_i is the drift rate of clock i, $Corr_{Hi}$ and $Corr_{Li}$ represent the clock adjustments completed so far on clock i since t_{HP} from the high-level and low-level synchronization, respectively. Similar notation is used for clock j.

Unlike low-level synchronizations where clocks are actually reset and the adjustments are performed gradually, corrections in the simulated high-level synchronization are instantaneously "completed" because physical clocks are not reset. Only the adjustment is accumulated. Therefore, the term $C_i(t_{HP}) - C_j(t_{HP}) + Corr_{Hi} - Corr_{Hj}$ in the above equation becomes zero right after each high-level synchronization as corrections on the two clocks will eliminate the skew. That leaves two terms that can influence the clock skew: the term associated with drifts between clocks, $(\delta_i + \delta_j)(t - t_{HP})$, and the low-level correction term $Corr_{Li} - Corr_{Lj}$.

Because physical clocks are not reset, the drifts between two clocks during the high-level synchronization interval can be estimated as the difference of two

consecutively measured physical clock skews (Table 4.3). We also keep track of the total corrections made on each clock from low-level synchronizations during the ten-minute interval. For Experiments H1-H6, the drifts between clocks over the ten-minute interval may be quite consistent, however, the total corrections made on individual low-level master during the same period are independent and the difference of the total corrections is not predictable. Therefore, with the two factors interacting together, it is difficult to trace the difference in experimental set-ups from the resulting distributions of clock skews. This can be demonstrated by the experimental data (Experiment H4) in Table

Table 4.3. Measurements of Clock Skews and Drifts between Gyre and Tove (in ms).

Sync. Round	Interval=5min		Interval=10min		Interval=20min	
	Skew	Drift	Skew	Drift	Skew	Drift
1	-21705		-14805		-48000	
2	-21715	-10	-14825	-20	-48045	-45
3	-21725	-10	-14845	-20	-48085	-40
4	-21735	-10	-14870	-25	-48130	-45
5	-21745	-10	-14890	-20	-48175	-45
6	-21755	-10	-14910	-20	-48215	-40
7	-21795	-40	-14930	-20	-48260	-45
8	-21805	-10	-14955	-25	-48305	-45
9	-21815	-10	-14970	-15	-48350	-45
10	-21825	-10	-14995	-25	-48390	-40
Between Gyre and Mimsy.						
Sync. Round	Interval=5min		Interval=10min		Interval=20min	
	Skew	Drift	Skew	Drift	Skew	Drift
1	-6285		-13565		-163764	
2	-6295	-10	-13580	-15	-163794	-30
3	-6305	-10	-13595	-15	-163824	-30
4	-6315	-10	-13605	-10	-163844	-20
5	-6315	0	-13645	-40	-163879	-35
6	-6325	-10	-13655	-10	-163904	-25
7	-6335	-10	-13670	-15	-163934	-30
8	-6340	-5	-13705	-35	-163958	-34
9	-6345	-5	-13720	-15	-163994	-36
10	-6355	-10	-13735	-15	-164024	-30

4.4. Let $Corr_{Li}$ in Equation (4.1) represents the total adjustment made on Mimsy's clock from low-level synchronizations, and $Corr_{Lj}$ represents the total adjustment on Gyre's clock. In synchronization round 4 of Table 4.4, $Corr_{Li}$ is 30 milliseconds, and $Corr_{Lj}$ is -10 milliseconds, the drift between the two clocks is -10 milliseconds, therefore, the logical clock difference is 30 milliseconds. However, in Round 8, seemingly larger drift and corrections result in a smaller clock skew of 10 milliseconds. This explains the randomness in the distribution of clock skews in the high-level synchronization.

Table 4.4. Components Contributing to Deviations in Logical Clock Times.

Sync. Round	$Corr_{Mimsy}$	$-Corr_{Gyre}$	$(\delta_i + \delta_j) * Intvl$	logical clock difference
2	0	10	-15	-5
3	0	10	-15	-5
4	30	10	-10	30
5	5	10	-40	-25
6	5	0	-10	-5
7	10	10	-15	5
8	30	15	-35	10
9	0	0	-15	-15
10	10	5	-15	0

4.2.2. Measurements on High-level Synchronization Intervals

In Experiments H7-H9, the low-level synchronization interval is kept at four minutes for all low-level LANs, while the high-level synchronization interval varies from 40 to 60 to 80 minutes. The distributions of clock skews between low-level masters Tove and Gyre are shown in Figure 4.5. It is observed that longer interval between resynchronizations does in fact cause wider clock separation. For instance, the average clock skew between low-level masters Tove and Gyre is 8.2 milliseconds in H7, 9.5 milliseconds in H8, and 16.8 milliseconds in H9. Though the clock skew is still governed by Equation (4.1), however, due to the fact that the high-level synchronization interval is significantly

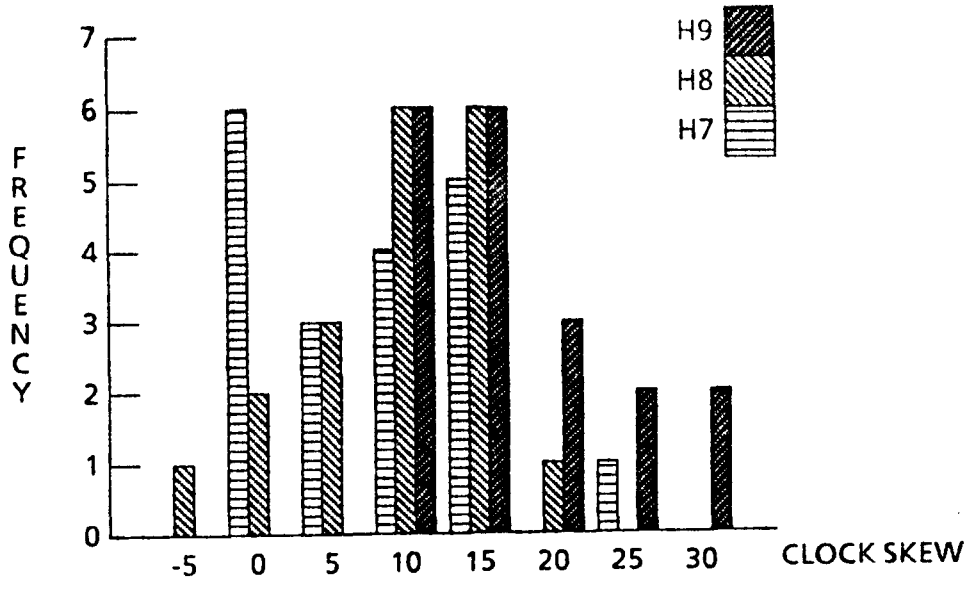

Figure 4.5. Histograms of clock skews between low-level masters Tove and Gyre of Experiments H7, H8, and H9 (See Table 4.2 for parameter set-ups).

longer than the low-level synchronization interval, the drift factor in this case thus plays a more important role, and results in a clearer distribution pattern than earlier measurements.

5. Concluding Remarks

Experiments with the Tempo algorithm on the local Ethernet have shown encouraging results. Consistent with measurement results presented in [Guse83], the clock separation on the three participating hosts is less than 20 milliseconds when these clocks are synchronized every ten minutes. The drifts between clocks over the synchronization interval account for a major part of the clock skew, which implies a close synchrony right after each synchronization. The close synchrony may be attributed to relatively low network load, which makes more accurate measurements of clock skews possible .

For clocks in a two-level hierarchical LAN, the clock skews are not large, either. The master clocks of different LANs are synchronized to within 20 and 30 milliseconds, when the high-level synchronization is spaced 40 and 80 minutes apart. Clock drifts over a longer interval cause larger skews, however, because of local synchronizations in the individual low-level LAN, the resulting clock skews are quite small in this case.

From our experiments, we feel ETempo can be used to synchronize clocks in a two level network, if the overall network load is not heavy and good estimate of clock skews is achievable. Extension of ETempo to a multi-level network needs further study.

Acknowledgment

The original ideas for this research were conceived while Tripathi was visiting University of Erlangen, W. Germany. He would like to thank Alexander Von Humboldt-Stiftung and Professor Herzog for support while in Germany.

References

[Chan86] Shu-jen Chang, Clock synchronization in a hierarchical network, M.S. thesis, University of Maryland, Aug. 1986.

[Guse83] Riccardo Gusella and Stefano Zatti, TEMPO, time services for the Berkeley local network, Report No. UCB/CSD 83/163, University of California, Berkeley, Dec. 1983.

[Guse85a] Riccardo Gusella and Stefano Zatti, The Berkelsy UNIX 4.3BSD time synchronization protocol: protocol specification, Report No. UCB/CSD 85/250, University of California, Berkeley, June 1985.

[Guse85b] Riccardo Gusella and Stefano Zatti, An election algorithm for a distributed clock synchronization program, Report No. UCB/CSD 86/275, University of California, Berkeley, Dec. 1985.

[Halp83] J. Y. Halpern, B. B. Simons, and H. R. Strong, An efficient fault-tolerant algorithm for clock synchronization, IBM RJ4094, 1983.

[Lamp78] L. Lamport, Time, clocks, and the ordering of events in a distributed system,