Furthermore, if p is in N, then \((q,r,m,q',Y,Z)\) is a step of p, where q is p's state in F, \(r = Ph_p(t)\), and \(q'\) is p's state in F'.

The first condition merely ensures that only a finite number of occurrences take place by any finite time. The second condition states that the configurations match up correctly. The final condition causes the configurations to change according to the process' transition function, if it is nonfaulty. Since a faulty process need not obey its transition function, it can send any messages and set any timers.

Given \(f\), an initial configuration F, and a schedule s, a history can be constructed inductively by starting with F and applying the transition functions as specified by the events in s to determine the next configuration. We will denote the history so derived by \(hist(s,F,f)\).

Define, for each process p and history h, \(\text{first-step}(h,p) = \min\{t: h(t) \text{ contains an event for } p\}\). This is the earliest time at which a step is taken by p in h. If p never takes a step, then \(\text{first-step}(h,p)\) is \(\infty\). Let \(\text{first-step}(h) = \min_{p \in F}\{\text{first-step}(h,p)\}\). This is the earliest time at which any process takes a step in h. Similarly, define, for each history h and nonfaulty process p, \(\text{last-step}(h,p) = \min\{t: h(t) \text{ contains a configuration in which p is in a final state}\}\). This is the earliest time at which p is a final state. Define \(\text{last-step}(h) = \max_{p \in F}\{\text{last-step}(h,p)\}\). This is the earliest time in h after which all nonfaulty processes are in final states. If some p in N never enters a final state in h, then \(\text{last-step}(h,p)\) and \(\text{last-step}(h)\) are \(\infty\).

### 2.6 Chronicles

In order to isolate the steps of an individual process in a history from the real times at which they occur, we define a chronicle.

The chronicle of nonfaulty process p in history h is the sequence of tuples of the form \((q_i,r_i,m_i,q'_i,Y_i,Z_i)\) which is derived as follows: if the i-th action for p occurs in h(t), then \(m_i\) is the message received in that action, \(q_i\) is the state of p in the preceding configuration of the action, \(r_i\) is p's physical clock reading at real time t, \(q'_i\) is the state of p in the succeeding configuration, \(Y_i\) is the collection of messages to be sent to the message buffer, and \(Z_i\) is the collection of timers to be set. We know that each tuple is a step of p.

Two histories, h for \(f = (P,N,S,PH)\) and h' for \(f' = (P,N,S,PH')\), are equivalent if, for each process p in N, the chronicle of p in h is the same as the chronicle of p in h'.
2.7 Shifting

Given a schedule $s$, nonfaulty process $p$, and real number $\xi$, define a new schedule $s' = shift(s,p,\xi)$ to be the same as $s$ except that an event for $p$ appears in $s'(t)$ if and only if the same event appears in $s(t + \xi)$, and the order of events for $p$ is preserved. The result $s'$ can easily be seen to be a schedule also. All events involving $p$ are shifted earlier by $\xi$ if $\xi$ is positive, and shifted later by $-\xi$ if $\xi$ is negative.

A set of clocks $PH = \{Ph_q\}_{q \in P}$ can also be shifted. Let $PH' = shift(PH,p,\xi)$ for $p \in N$ be the set of clocks defined by $PH' = \{Ph'_q\}_{q \in P}$ where $Ph'_q(t) = Ph_q(t)$ if $q \neq p$, and $Ph'_p(t) = Ph_p(t) + \xi$. Process $p$'s clock has been shifted forward by $\xi$, but no other clocks are altered.

Lemma 2.1 states that if a schedule and a set of clocks are shifted by the same amount relative to the same process, then the histories derived from those schedules and sets of clocks starting from the same initial configuration are equivalent.

Lemma 2.1: Let $f = (P,N,S,PH)$ and $f' = (P,N,S,PH')$, where $PH' = shift(PH,p,\xi)$ for some process $p$ and real number $\xi$. Let $s$ be a schedule for $P$ and $s' = shift(s,p,\xi)$. Let $F$ be an initial configuration for $f$ and $f'$. Then the history $hist(s,F,f) = h$ is equivalent to the history $hist(s',F,f') = h'$.

Proof: Let $q$ be an arbitrary process in $N$. It suffices to show that the chronicle of $q$ in $h$ is the same as the chronicle of $q$ in $h'$.

Case 1: $q \neq p$. We proceed by induction on the elements of the chronicles. Let $q$'s chronicle in $h$ be $(m_1,qc_1,Ph_1(t_1),qn_1,Y_1,Z_1)$ and in $h'$ be $(m'_1,qc'_1,Ph'_1(t'_1),qn'_1,Y'_1,Z'_1)$. (qc stands for current state, $qn$ for next state.)

Basis: $i = 1$. Then $t_1 = first-step(h,q)$ and $t'_1 = first-step(h',q)$. By construction of $h'$, these real times are the same. Therefore, $m_1 = m'_1$. Since $F$ is the initial configuration in both $h$ and $h'$, $qc_1 = qc'_1$. $Ph_1(t_1) = Ph'_1(t'_1)$ since $Ph_q = Ph'_q$ by construction. Finally, $qn_1 = qn'_1$, $Y_1 = Y'_1$, and $Z_1 = Z'_1$ since $\tau_q$ is deterministic and the inputs are the same.

Induction: Assume the elements are the same up to $i - 1$, and show that the $i$-th elements are the same. Again, $m_i = m'_i$ by construction of $h'$; $qc_i = qc'_i$ by the induction hypothesis since $qc = qc'$; $Ph_1(t_i) = Ph'_1(t'_i)$ as before; finally $qn_i = qn'_i$, $Y_i = Y'_i$, and $Z_i = Z'_i$ because $\tau_q$ is deterministic.

Case 2: $q = p$. Again we proceed by induction on the elements of the chronicles. Let $p$'s chronicle in $h$ be $(m_1,qc_1,Ph_p(t_1),qn_1,Y_1,Z_1)$ and in $h'$ be $(m'_1,qc'_1,Ph'_p(t'_1),qn'_1,Y'_1,Z'_1)$.

First we note that by construction, $t_1 = t'_1 + \xi$ for all $i$.

Basis: $i = 1$. By construction, $m_1 = m'_1$. Since $F$ is the initial configuration in both $h$ and $h'$, $qc_1 = qc'_1$. $Ph_1(t_1) = Ph'_1(t'_1)$ since $Ph_p(t_1) = Ph'_p(t'_1 + \xi)$ and $Ph'_p(t'_1 + \xi) = Ph'_p(t'_1 + \xi)$. Finally, $qn_1 = qn'_1$, $Y_1 = Y'_1$, and $Z_1 = Z'_1$ since $\tau_p$ is deterministic and the inputs are
the same.

**Induction:** assume the elements are the same up to i - 1, and show that the i-th elements are the same. \( m_i = m'_i \) by construction of \( h' \); \( q_i = q'_i \) by the induction hypothesis; \( Ph_i(t) = Ph'_i(t') \) by the same argument as in the basis case; and again \( q_i = q'_i \), \( y_i = y'_i \), and \( z_i = z'_i \) since \( \tau_p \) is deterministic.

The next lemma quantifies the changes to the message delays in a history when its schedule and set of clocks are shifted by the same amount relative to the same process.

**Lemma 2.2:** Let \( \mathcal{H} = (P, N, S, PH) \) and \( \mathcal{H}' = (P, N, S, PH') \), where \( PH' = shift(PH, p, n, r) \) for some \( p \) in \( P \) and real number \( r \). Let \( s \) be a schedule for \( P \) and \( s' = shift(s, p, r) \). Let \( F \) be an initial configuration for \( \mathcal{H} \) and \( \mathcal{H}' \). Then there is a one-to-one correspondence between the tuples in the message buffer in \( h = hist(s, F, h) \) and \( h' = hist(s', F, h') \), and the message delays for corresponding elements will be the same in the two histories (if defined) except for two cases:

1. if the delay for any tuple of the form \((p, x, q)\) is \( \mu \) in \( h \) for any process \( q \neq p \) and message value \( x \), then the delay for the corresponding element in \( h' \) will be \( \mu + r \); and

2. if the delay for any tuple of the form \((q, x, p)\) is \( \mu \) in \( h \) for any process \( q \neq p \) and message value \( x \), then the delay for the corresponding element in \( h' \) will be \( \mu - r \).

**Proof:** By Lemma 2.1, \( h \) and \( h' \) are equivalent. Therefore, the chronicles of all the processes are the same. The same messages are sent and received at the same physical clock times in \( h' \) and \( h \). Also, the message buffers have the same START elements since the initial configuration is the same for both. Therefore, each element of the message buffer in \( h \) has a corresponding one in \( h' \) and vice versa.

START messages are all either received at some finite time or not, thus START elements have the same delays in the two histories. Since only \( p \)'s clock is shifted, the clocks of the other processes will bear the same relationship to real time in \( h' \) as in \( h \), causing the delays for messages between processes other than \( p \) and the delays of timers for processes other than \( p \) to be the same in the two histories. The delays of timers for \( p \) will be the same as well, since they are both set and received \( r \) earlier in \( h' \) than in \( h \).

Choose \( q \neq p \).

1. Suppose \((p, x, q)\) is sent at \( t \) and received at \( t' \) in \( h \). The relationship between \( s \) and \( s' \) implies that \((p, x, q)\) is sent at \( t - r \) and received at \( t' \) in \( h' \). Thus the message delay in \( h' \) is \( t' - (t - r) = \mu + r \).

2. Suppose \((q, x, p)\) is sent at \( t \) and received at \( t' \) in \( h \). The relationship between \( s \) and \( s' \) implies that \((q, x, p)\) is sent at \( t \) and received at \( t' - r \) in \( h' \). Thus the message delay in \( h' \) is \( t' - r - t = \mu - r \).
2.8 Executions

Now we require correct behavior of the message system. Accordingly, we define an execution to be a history with the necessary properties.

We fix for the remainder of the thesis two nonnegative constants $\delta$ and $\epsilon$ with $\delta > \epsilon$.

An execution for $\mathcal{J}$ is a history for $\mathcal{J}$ with four additional properties:

- the initial state of the message buffer consists exactly of a START message for each process in $S \cup (P - N)$, that is, for each self-starting process and each faulty process;
- all START messages for nontaulty processes are received at some finite time;
- the message delay of any non-TIMER and non-START message is between $\delta - \epsilon$ and $\delta + \epsilon$ inclusive; and
- any (TIMER, $T$, $p$) element of the message buffer, for any $T$ and $p$, has finite message delay and is delivered at $P_{p}^{\mathcal{J}}(T)$.

The intent of the first condition is to model the self-starting processes as those processes that begin the algorithm on their own, and to allow the faulty processes to begin their bad behavior at arbitrary times. The second condition states that nonfaulty self-starting processes all receive their START messages. The third condition guarantees that all interprocess messages arrive at their destinations within $\delta$ of being sent, subject to an uncertainty of $\epsilon$. The fourth condition ensures that a timer goes off if and only if it was previously set and that it goes off at the right time.

2.9 Logical Clocks

Each process $p$ has as part of its state a local variable CORR, which provides a correction to its physical clock to yield the local time. During an execution, $p$'s local variable CORR takes on different values. Thus, for a particular execution, it makes sense to define a function $\text{CORR}_p(t)$, giving the value of $p$'s variable CORR at time $t$. For a particular execution, we define the local time for $p$ to be the function $L_p$, which is given by $P_{p}^\mathcal{J} + \text{CORR}_p$.

A logical clock of $p$ is $P_{p}^\mathcal{J}$ plus the value of $\text{CORR}_p$ at some time. Let $C^0_p$ denote the initial logical clock of $p$, given by $P_{p}^\mathcal{J}$ plus the value of $\text{CORR}_p$ in $p$'s initial state. Each time $p$ adjusts its CORR variable, it is, in effect, changing to a new logical clock $C^i_p$ for some $i$. The local time can be thought of as a piecewise continuous function, each of whose pieces is part of a logical clock.
Chapter Three

Lower Bound

3.1 Introduction

In this chapter, we show a lower bound on how closely clocks can be synchronized, even if the clocks don't drift and no processes are faulty. Since these are strong assumptions, this lower bound also holds for the more realistic case in which clocks do drift and arbitrary faults occur. Just to show that the bound is tight, we present a simple algorithm that synchronizes the clocks as closely as the lower bound.

3.2 Problem Statement

For this chapter alone we make the following assumptions:

1. clocks don't drift, i.e. \( \frac{dC_p(t)}{dt} = 1 \) for all \( p \) and \( t \);
2. all processes are nonfaulty, i.e. \( N = P \). Therefore, we will omit "N" from the notation.

Since the processes have physical clocks which are progressing at the same rate as real time, the only part of the clock synchronization problem which is of interest is the problem of bringing the clocks into synchronization -- once this has been done, synchronization is maintained automatically.

A clock synchronization algorithm \((P,S)\) is \( \gamma, \alpha \)-correct if every execution \( h \) for \((P,S,PH)\), for any set of clocks \( PH \), satisfies the following three conditions:

1. Termination: All processes eventually enter final states. Thus, last-step\((h)\) is defined.

2. Agreement: \( |L_p(t) - L_q(t)| \leq \gamma \) for any processes \( p \) and \( q \) and time \( t \geq \) last-step\((h)\). We say \( h \) synchronizes to within \( \gamma \).

3. Validity: For any process \( p \) there exist processes \( q \) and \( r \) such that \( C_q^0(t) - \alpha \leq L_p(t) \leq C_r^0(t) + \alpha \) for all times \( t \geq \) last-step\((h)\). This ensures that \( p \)'s new logical clock isn't too much greater (or smaller) than the largest (or smallest) old logical clock would have been at this time. We say \( h \) bounds the adjustment within \( \alpha \).

We will show that no algorithm can be \( \gamma, \alpha \)-correct for \( \gamma < 2\epsilon(1 - 1/n) \) and any \( \alpha \), where \( \epsilon \) is the
uncertainty in the message delivery time and \( n \) is the number of processes. Then we exhibit a simple algorithm that is \( 2\epsilon(1 - 1/n) \), \( \epsilon \)-correct.

### 3.3 Lower Bound

In this section we show that no algorithm can synchronize \( n \) processes' clocks any closer than \( 2\epsilon(1 - 1/n) \).

**Theorem 3.1:** No clock synchronization algorithm can synchronize a system of \( n \) processes to within \( \gamma \), for any \( \gamma < 2\epsilon(1 - 1/n) \).

**Proof:** Fix a system of processes \((P,S)\) that synchronizes to within \( \gamma \). We will show that \( \gamma \geq 2\epsilon(1 - 1/n) \).

Let \( P \) consist of processes \( p_1 \) through \( p_n \). Consider the system \( \mathcal{Y}_1 = (P,S,PH_1) \). Consider an execution \( h_1 = \text{hist}(s_1,F,\mathcal{Y}_1) \), for some schedule \( s_1 \) and initial configuration \( F \), of any clock synchronization algorithm in which all messages from \( p_j \) to \( p_k \) have delay \( \delta - \epsilon \) if \( k > i \), have delay \( \delta + \epsilon \) if \( k < i \), and have delay \( \delta \) if \( k = i \).

Consider \( n - 1 \) additional histories, \( h_2 \) through \( h_n \), for \( \mathcal{Y}_2 \) through \( \mathcal{Y}_n \). The systems are constructed inductively by letting \( PH_j = \text{shift}(PH_{j-1},p_i,2\epsilon) \) and \( \mathcal{Y}_j = (P,S,PH_j) \). The histories are constructed inductively by letting \( s_j = \text{shift}(s_{j-1},p_i,2\epsilon) \) and \( h_j = \text{hist}(s_j,F,\mathcal{Y}_j) \). Stated informally, the \( i \)-th history is obtained from the \((i-1)\)-st history by shifting the schedule and set of clocks by \( 2\epsilon \) relative to the \((i-1)\)-st process. Let \( PH_p \) be \( p \)'s physical clock in \( PH_i \).

By Lemma 2.1, all the \( h_i \) are equivalent.

Next we show by induction on \( i \) that \( h_i \) is an execution for \( \mathcal{Y}_i \), and further, that the delays in \( h_i \) for messages from \( p_j \) to \( p_k \) are \( \delta + \epsilon \) if \( j < i \) and \( k \geq i \), \( \delta - \epsilon \) if \( j \geq i \) and \( k < i \), otherwise as in \( h_1 \).

**Basis:** \( h_1 \) is an execution and the message delays are as required by hypothesis.

**Induction:** Assume \( h_i \) is an execution with the required message delays, and show that \( h_{i+1} \) is also an execution with the required message delays.

- The initial state of the message buffer is the same in \( h_{i+1} \) as in \( h_i \), since both use initial configuration \( F \). Thus the initial state is as required.

- The START messages are all received in \( h_{i+1} \) as they are in \( h_i \).

- By Lemma 2.2, a message in \( h_{i+1} \) from \( p_j \) to \( p_m \), \( m > i \), will have delay \( \delta - \epsilon + 2\epsilon = \delta + \epsilon \); one from \( p_j \) to \( p_m \), \( m < i \), will have delay \( \delta + \epsilon + 2\epsilon = \delta + \epsilon \); one from \( p_m \) to \( p_i \), \( m > i \), will have delay \( \delta + \epsilon - 2\epsilon = \delta - \epsilon \); and one from \( p_m \) to \( p_i \), \( m < i \), will have delay \( \delta + \epsilon - 2\epsilon = \delta - \epsilon \). The others stay the same. Thus the delays are within the correct range.

- Now we need to show that timers are handled properly in \( h_{i+1} \). Lemma 2.2
implies that the message delays are the same in $h_{i+1}$ as in $h_i$, thus they are finite. For all processes except $p_i$, the timers arrive at the same real times and the same clock times in $h_{i+1}$ as in $h_i$, and thus they arrive at the proper times in $h_{i+1}$. Consider a timer set by $p_i$ for $T$ that arrives at $T = \Phi_i(t)$ in $h_i$. In $h_{i+1}$ it arrives at $t + 2\varepsilon$. However, since $\Phi_{i+1}(t + 2\varepsilon) = \Phi_i(t) = T$, the timer arrives at the proper time in $h_{i+1}$.

Therefore, $h_i$ is an execution for $f_i$.

Since $h_i$ was correct, it terminated; therefore, $h_i$ also terminates. Let $t_f = \max_{i=1..n}(\text{last-step}(h_i))$. In execution $h_i$, the algorithm synchronizes all the processes' clocks to values $v_i$ through $v_n$ at time $t_f$, and all the values are within $\gamma$. In particular,

$$v_n \leq v_1 + \gamma.$$ 

Since $h_i$ is equivalent to $h_{i-1}$, the correction variable for any process $p$ will be the same in both executions at time $t_f$. The value of $p_{i-1}'$'s logical clock at $t_f$ will be $v_{i-1} + 2\varepsilon$ and the value of $p_i$'s logical clock at $t_f$ will be $v_i$ by the way $PH_i$ is defined. Since these values are within $\gamma$, we have

$$v_i - v_{i-1} \leq v_i + \gamma - 2\varepsilon.$$ 

Putting together this chain of inequalities, we have

$$v_n \leq v_1 + \gamma \leq ... \leq v_i + (i-1)(\gamma - 2\varepsilon) + \gamma \leq ... \leq v_n + (n-1)(\gamma - 2\varepsilon) + \gamma.$$ 

Therefore, $v_n \leq v_n + (n-1)(\gamma - 2\varepsilon) + \gamma$, and so $0 \leq (n-1)\gamma - (n-1)2\varepsilon + \gamma$. In order for this inequality to hold, it must be the case that $\gamma \geq 2\varepsilon(1 - 1/n)$. 

### 3.4 Upper Bound

In this section we show that the $2\varepsilon(1 - 1/n)$ lower bound is tight, by exhibiting a simple algorithm which synchronizes the clocks to within this amount.

#### 3.4.1 Algorithm

There is an extremely simple algorithm that achieves the closest possible synchronization. As soon as each process $p$ receives a message, it sends its local time in a message to the remaining processes and waits to receive a similar message from every other process. Immediately upon receiving such a message, say from $q$, $p$ estimates $q$'s current local time by adding $\delta$ to the value received. Then $p$ computes the difference between its estimate of $q$'s local time and its own current local time. After receiving local times from all the other processes, $p$ takes the average of the estimated differences (including $0$ for the difference between $p$ and itself) and adds this average to its correction variable. Note that in contrast to many other agreement algorithms, in
this one each process treats itself non-uniformly with the others.

Since it is obviously impractical to write algorithms in terms of transition functions, we have employed a clean, simple notation for describing interrupt-driven algorithms. To translate this notation into the basic model, we first assume that the state of a process consists of values for all the local variables, together with a location counter which indicates the next beginstep statement to be executed. The initial state of a process consists of the indicated initial values for all the local variables, and the location counter positioned at the first beginstep statement of the program.

The transition function takes as inputs a state of the process, a message, and a physical time, and must return a new state and a collection of messages to send and timers to set. This is done as follows. The beginstep statement is extracted from the given state. The local variables are initialized at the values given in the state. The parameter u is set equal to the message. The variable NOW is initialized at the given physical time + CORR. The program is then run from the given beginstep statement, just until it reaches an endstep statement. (If it never reaches an endstep statement, the transition function takes on a default value.) The next beginstep after that endstep, together with the new values for all the local variables resulting from running the program, comprise the new state. The messages sent are all those which are sent during the running of the program, and similarly for the timers.

There is a set-timer statement, which takes an argument U representing a logical time. The corresponding physical time, U - CORR, is the physical time described by the transition function. (This statement is not used in this algorithm but will be used later in the thesis.)

We will use the shorthand NOW to stand for the current logical clock time and ME for the id of the process running the code.

For this algorithm, initial states are those in which the location counter is at the beginning of the code, local variables CORR and V have arbitrary values, and local variables SUM and RESPONSES have value 0. Final states are those in which the location counter is at the end of the code.

The code is in Figure 3.1.

We will show that any execution h of Algorithm 3-1 is $\gamma, \alpha$-correct, where $\gamma = 2\epsilon(1 - 1/n)$ and $\alpha = \epsilon$. Thus, Algorithm 3-1 synchronizes the clocks to within $2\epsilon(1 - 1/n)$, showing that the lower bound is tight. The upper bound isn't as unintuitive as it might look at first glance; it can be
beginstep(u)
send(NOW) to all q ≠ ME

do forever
    if u = (v,q) for some message value v and process q then
        V := v + δ - NOW
        SUM := SUM + V
        RESPONSES := RESPONSES + 1
    endif
    if RESPONSES = n - 1 then exit endif
endstep
beginstep(u)
enddo

CORR := CORR + SUM/n
endstep

Figure 3.1: Algorithm 3.1, Synchronizing to within the Lower Bound

rewritten as \((2\varepsilon + (n - 2)2\varepsilon)/n\), the average of the discrepancies in the estimated differences. The estimated differences of two processes for each other can differ by at most \(\varepsilon\) apiece (giving the \(2\varepsilon\) term), and their estimated differences for the other \(n - 2\) processes can differ by up to \(2\varepsilon\) apiece (giving the \((n - 2)2\varepsilon\) term). Then the estimated differences are averaged, so the sum is divided by \(n\). A more careful analysis is given below.

3.4.2 Preliminary Lemmas

The next two results follow easily from the assumption that clocks don't drift.

Lemma 3.2: For any \(p\) and \(i \geq 0\), \(C^i_p(t') - C^i_p(t) = t' - t\).

Proof: Immediate since the slope of \(C^i_p\) is 1. \(\square\)

Lemma 3.3: For any \(p\) and \(q\), \(i \geq 0\), and times \(t\) and \(t'\), \(C^i_p(t') - C^i_q(t') = C^i_p(t) - C^i_q(t)\).

Proof: \(C^i_p(t') - C^i_q(t') = t' - t = C^i_p(t) - C^i_q(t)\) by two applications of Lemma 3.2. The result follows. \(\square\)

Now we can define the initial difference between two processes’ clocks in execution \(h\). Define \(\Delta_{pq}\) to be \(C^0_p(t) - C^0_q(t)\). That is, \(\Delta_{pq}\) is the difference in local times before either of the processes has changed its correction variable. Since there is no drift in the clock rates, any time will give the same value.

Lemma 3.4: For any execution \(h\), and processes \(p\) and \(q\), \(\Delta_{pq} = -\Delta_{qp}\).

Proof: Immediate from the definition of \(\Delta\). \(\square\)

Lemma 3.5: For any execution \(h\), and processes \(p, q,\) and \(r\), \(\Delta_{pq} = \Delta_{pr} + \Delta_{rq}\).

Proof: Immediate from the definition of \(\Delta\). \(\square\)
3.4.3 Agreement

For q ≠ p, let \( V_{qp} \) be the value of variable V in the code when q's message is being handled by p.

\[ V_{qp} = L_q(t) + \delta - L_p(t'), \]

where local time \( L_q(t) \) was sent by q at real time t and received by p at real time \( t' \). Let \( V_{pp} = 0 \). We will denote \( \text{SUM}/n \), p's addition to its correction variable, by \( A_p \).

First we relate the estimate \( V_{qp} \) to the actual value \( \Delta_{qp} \).

**Lemma 3.6:** \( |V_{qp} - \Delta_{qp}| \leq \epsilon. \)

**Proof:** Suppose at real time t, q sent the value \( L_q(t) \), which was received by p at real time \( t' \). Then

\[
|V_{qp} - \Delta_{qp}| = |L_q(t) + \delta - L_p(t') - \Delta_{qp}| = |C^0_q(t) + \delta - C^0_p(t') - \Delta_{qp}|
\]

\[
= |C^0_p(t) + \Delta_{qp} + \delta - C^0_p(t') - \Delta_{qp}|, \text{ by definition of } \Delta_{qp}
\]

\[
= |C^0_p(t) - C^0_p(t') + \delta|
\]

\[
= |t - t' + \delta|, \text{ by Lemma 3-2}
\]

\[
\leq |\delta - (\delta - \epsilon)|, \text{ since } \delta - \epsilon \text{ is the smallest message delay}
\]

\[
= \epsilon. \quad \blacksquare
\]

Here is the main result.

**Theorem 3.7:** (Agreement) Algorithm 3-1 guarantees clock synchronization to within \( 2\epsilon(1 - 1/n) \).

**Proof:** We must show that for any execution h, any two processes p and q, and all times \( t \) after last-step(h),

\[
|L_p(t) - L_q(t)| \leq 2 \epsilon - 2\epsilon/n.
\]

Without loss of generality, assume \( p = p_1 \) and \( q = p_2 \) so that the remaining processes are \( p_3 \) through \( p_n \). By the way the algorithm works,

\[
|L_p(t) - L_q(t)| = |(C^0_p(t) + A_p) - (C^0_q(t) + A_q)| = |\Delta_{pq} + A_p - A_q|.
\]

We know by definition of \( A_p \) and \( A_q \) that

\[
A_p = (1/n)(V_{pp} + V_{qp} + \sum_{i=3..n} V_{pi}) \quad \text{and}
\]

\[
A_q = (1/n)(V_{pq} + V_{qq} + \sum_{i=3..n} V_{pq}).
\]

Substituting these values and noting that \( V_{pp} = V_{qq} = 0 \), we get

\[
|L_p(t) - L_q(t)| = |\Delta_{pq} + (1/n)(V_{qp} + \sum_{i=3..n} V_{ip} - V_{pq} - \sum_{i=3..n} V_{pq})|
\]