Time Synchronization

Readings:
Fan, Lynch. Gradient clock synchronization
Attiya, Hay, Welch. Optimal Clock Synchronization Under Energy Constraints

1 Introduction

In this lecture, we describe a new type of clock synchronization problem, called *gradient clock synchronization (GCS)*, which is particularly relevant to radio networks. We then prove there is a lower bound to how well the problem can be solved, which implies lower bounds to how efficiently certain algorithms which use synchronized clocks can be executed. We demonstrate GCS by an example. Suppose two sensor nodes are being used to measure the velocity of a target, with a desired accuracy of 1%. To do this, each node records its local time when it observes the target. Then, the nodes exchange their time readings, compute the difference as t, and divide t into d, the known distance between the sensors. Assuming d to be accurate, the error in the calculation comes from the clock sync error between the sensors; call this ϵ. If d is small, then ϵ must be small, if we want $\frac{d}{t}$ to be accurate to 1% of the true velocity. But, if d is large, then ϵ can be proportionally larger and $\frac{d}{t}$ will still be accurate to 1%.

Thus, the property we want from the clock sync algorithm is: if nodes are close, their clocks are well synchronized; if nodes are faraway, their clocks are allowed to be more loosely synchronized. In other words, the acceptable clock skew[1] between nodes forms a *gradient* in their distance.

2 Problem Definition

2.1 Network Model

Let's now define the problem more rigorously. We first define the network in which the clock sync algorithm operates. Let N be a network consisting of nodes $p_1, \ldots, p_n$. We connect two nodes by an edge if they can directly communicate with each other; assume N is connected. Consider a particular edge $e = (p_i, p_j)$, and suppose each message sent from p_i to p_j (and from p_j to p_i) takes at least 0 time, and at most $d_{i,j}$ time to reach p_j. That is, $d_{i,j}$ is the *message delay uncertainty* between p_i and p_j. The delay on each message is *adversarially controlled*. That is, each message between p_i and p_j can have any delay between 0 and $d_{i,j}$. We saw from the previous lecture that this uncertainty is the main obstacle to accurate clock synchronization between p_i and p_j. We call $d_{i,j}$ the *distance* between p_i and p_j. The distance between different pairs of nodes can vary. Let D be the maximum distance between any pair of nodes, in the shortest path metric induced by the distance on edges.

[1] Note that in this lecture, we use the term *skew* differently from the GCS lecture. By clock skew, we mean here the difference in the clock values of two nodes. The RBS paper called this *phase offset*.

2.2 Gradient Clock Synchronization

We now describe what the clock sync algorithm is supposed to accomplish. We assume the algorithm is *deterministic*. Intuitively, the algorithm is supposed to produce, at each node and at all times, a single number which represents that node's view of *global time*. Notice that this is somewhat different from the goal of an algorithm like RBS, which only requires each node to be able to compute another node's local time, without adopting either node's local time as the global time. Having a global time is necessary for some applications. Global time is also arguably more natural, simpler, and easier to use. More precisely, we assume that each node p_i has a *hardware clock*, which is a device which at any realtime t outputs a number $H_i(t)$. The hardware clock is characterized by its *drift rate* ρ, where $0 < \rho < 1$: we have $1 - \rho \leq \frac{\mathrm{d}H_i(t)}{\mathrm{d}t} \leq 1 + \rho$. That is, in one second of realtime, H_i increases between $1 - \rho$ to $1 + \rho$ seconds. The drift in H_i is adversarially controlled. That is, $\frac{\mathrm{d}H_i(t)}{\mathrm{d}t}$ can take any value between $1 - \rho$ and $1 + \rho$, at any realtime t. Only p_i can look at H_i. In addition, p_i can send messages to other nodes. The messages are assumed to reliably get to the target node, subject to the message delay uncertainty described above. Using H_i, and the messages it sends and receives, p_i must, at any realtime t, compute a *logical clock* value $L_i(t)$. The goal of the clock sync algorithm is to make the logical clock values of different nodes as close as possible.

Notice there is a trivial way to make the node's logical clocks be close: every node p_i sets $L_i(t) = 0$ for all t. But, such a "clock" is useless. To ensure nodes compute useful logical clock values, we require the following *validity condition*: for every node p_i, at any realtime t, we have $\frac{\mathrm{d}L_i(t)}{\mathrm{d}t} \geq \frac{1}{2}$. That is, in one second, a node's logical clock must increase by at least 0.5 seconds. Note that the number 0.5 was chosen for simplicity. We could have used any other constant. We can now formally define the gradient clock synchronization problem.

Definition 2.1 Let f be a function into $\mathbb{R}$, and let A be a clock synchronization algorithm. We say A satisfies the *f-gradient property* if in any execution of A, we have

$$\forall i, j \ \forall t : |L_i(t) - L_j(t)| \leq f(d_{i,j}, N)$$

That is, a clock sync algorithm A satisfies the f-gradient property if it guarantees that at all times, and for all pairs of nodes, the logical clock skew between the nodes is bounded by f of the distance between the nodes, and the network N[2]. Thus, this definition allows the amount of clock skew between nodes to depend on how far apart they are (in terms of message delay uncertainty). Since we want nodes to be well synchronized, we want f as small as possible. However, it turns out there is a lower bound to how small f can be.

3 A Lower Bound on GCS

The rest of this lecture is devoted to showing the following lower bound:

Theorem 3.1 Let A be any clock synchronization algorithm, and suppose A satisfies the f-gradient property. Then, we have

$$f(d, N) = \Omega(d + \frac{\log D}{\log \log D})$$

[2]The reason we have the second argument N is because the skew between two nodes can depend not only on their distance, but also on the network they are in, as we shall see in the next section.

where D is the diameter of network N.

In fact, we will prove a simplified version of this lower bound. In this version, we fix N to be the *line network* on n nodes. That is, N contains nodes $p_1, \ldots, p_n$. There is an edge (p_i, p_{i+1}) for all $i = 1, \ldots, n-1$. The distance of each edge is 1.

[[[PICTURE]]]

For this N, we will show that $f(1) = \Omega(\frac{\log n}{\log \log n})$. That is, no matter what clock sync algorithm A we use, there is some execution of A, in which a pair of neighboring nodes, that is, nodes distance 1 apart, have logical clock skew $\Omega(\frac{\log n}{\log \log n})$. We pause to consider this statement. The lower bound says that the amount of clock skew between two neighboring nodes depends not only on their distance, but also on the size of the network they are in. This may seem odd. Let p_i and p_{i+1} be two neighboring nodes. If our goal is simply to synchronize p_i and p_{i+1} as closely as possible, then we can easily ensure p_i and p_{i+1} have skew at most 1: just have the nodes send each other their logical clock values, and each node sets its clock to be the larger of its received and its own clock value. We can do this for any particular pair of neighboring nodes. What the lower bound says is that we can't do it for *all pairs* of neighboring nodes *at the same time*. That is, if we want to make p_i and p_{i+1}'s clocks to be close, we must sometimes allow p_j and p_{j+1}'s clocks be far apart, for some i and j.

This lower bound has practical implications. We'll just describe one. Consider the line network above, and suppose nodes are using TDMA as their MAC protocol. If each pair of neighboring nodes have a message delay uncertainty of 10 μsec, say using reference broadcasts, and each message takes 10 μsecs to send, one might think the nodes can use a repeating TDMA schedule of two 20 μsec slots, one for each neighbor. What this lower bound says is that the slots need to be much longer. In fact, if we want to guarantee that messages never collide, the slots should be at least $10 + \Omega(\frac{\log n}{\log \log n})$ μsec in width. That is, as the network size grows, the TDMA slots need to be arbitrarily long, even though each node only has two neighbors!

4 Proof Overview

Before proving $f(1) = \Omega(\frac{\log n}{\log \log n})$, we first show something simpler: $f(1) \geq \frac{1}{8}$. This proof introduces the key *indistinguishability principle* and *scaling argument*, which are also the basis for the $\Omega(\frac{\log n}{\log \log n})$ proof. We first describe the indistinguishability principle, which is used in different forms in almost all, if not all, lower bounds in distributed computing.

4.1 Indistinguishability Principle

To model a clock synchronization algorithm A, we stated in section 2.2 that each node has access to a private hardware clock, subject to drift, and could send and receive messages from its neighbors, subject to message delay uncertainty. Thus, a node's hardware clock, and the messages it receives, are its only view of the world. Since the algorithm is deterministic, then the actions a node performs are *completely determined* by the content of the messages it gets, and the values of its hardware clock when it receives the messages. More formally, suppose in an execution α of A, a node p_i received messages $m_1, m_2, \ldots, m_k$. Let $H_i(m_j)$ be p_i's hardware clock value at the moment it received message m_j, and let $T(m_j)$ be the *realtime* at which p_j received m_j. Suppose there is another execution β of A, in which p_i receives messages $n_1, \ldots, n_k$. Now, assume that $m_j = n_j$,

and also $H_i(m_j) = H_i(n_j)$, for all $j = 1, \ldots, k$. Then p_i will perform the *exact same actions* in α and β. This is true even if for some j, $T(m_j) \neq T(n_j)$; that is, even if the realtime at which p_i received some message is different in α and β. This is because p_i has no way to observe realtime. Given all the things p_i can observe, the message contents and its hardware clock values, α and β are indistinguishable to p_i.

4.2 Scaling Arguments

To prove $f(1) \geq \frac{1}{8}$, we use a technique called a *scaling argument*. For clarity in demonstrating the technique, we assume that there are only two nodes, p_1 and p_2, with distance 1 between them. Basically, we will create two executions α and β which look indistinguishable to p_1 and p_2, but in which the realtime at which events occur in α and β are different. Then, we can argue that p_1 and p_2's clock skew is at least $\frac{1}{8}$, in α or β.

Let α be any execution of A satisfying the following properties. The delay for all messages between p_1 and p_2 is $\frac{1}{2}$. p_1 and p_2's hardware clock rates are 1 at all times. Lastly, the duration of α is exactly $\frac{1+\rho}{2\rho}$. Now, create a second execution β, in which p_1 and p_2 perform the exact same actions as in α. The hardware clock rate of p_2 is 1 throughout β, and the hardware clock rate of p_1 is $1+\rho$. Every action at p_2 occurs at the same realtime in α and β. If an action occurs at realtime t at p_1 in α, then it occurs at realtime $\frac{t}{1+\rho}$ at p_1 in β. Thus, in β, p_2's execution has been scaled down by a factor of $1 + \rho$.

[[[PICTURE]]]

We now state some properties about α and β. First of all, we can easily check that α and β are indistinguishable to p_1 and p_2, and thus p_1 and p_2 behave the same in both executions.

Next, we can check that the delay of all messages sent between p_1 and p_2 in β is between 0 and 1. To see this, first consider a message sent from p_1 to p_2. Suppose that in α, p_1 sent the message at realtime t_1, and p_2 received the message at realtime t_2. Then in β, p_1 sends the message at realtime $t_1' = \frac{t_1}{1+\rho}$, and p_2 receives the message at realtime t_2. Thus, the delay of this message in β is

$$\begin{aligned} t_2 - t_1' &= t_2 - t_1 + t_1 - t_1' \\ &= \frac{1}{2} + \frac{\rho t_1}{1+\rho} \\ &\leq \frac{1}{2} + \frac{1}{2} = 1 \end{aligned}$$

The last inequality follows because $t_1 \leq \frac{1+\rho}{2\rho}$, the duration of α. Thus, every message from p_1 to p_2 has delay at most 1 in β, which is within the legal bounds. It's easy to show it has delay at least 0 also.

Similarly, if in α, p_2 sent a message to p_1 at realtime t_2, and p_1 received the message at realtime t_1, then in β, p_2 sends the same message at t_2, and p_1 receives the messages at $\frac{t_1}{1+\rho}$. Thus, the delay of the message is

$$\begin{aligned} t_1' - t_2 &= t_1' - t_1 + t_1 - t_2 \\ &= \frac{-\rho t_1}{1+\rho} + \frac{1}{2} \\ &\geq \frac{-1}{2} + \frac{1}{2} = 0 \end{aligned}$$

Thus, every message from p_2 to p_1 has delay at least 0 in β, which is within the legal bounds. It's easy to show it has delay at most 1 also. Thus, we conclude that β is a legal execution of A, because all hardware clock rates are between $1-\rho$ and $1+\rho$, and all message delays are between 0 and 1.

Let L_1 and L_2 be p_1 and p_2's logical clock values at the end of α, resp. Also, let L_1' and L_2' be p_1's and p_2's logical clock value in β at realtime $\frac{1}{2\rho}$, resp. Now, notice that p_1's hardware clock value is the same at realtime $\frac{1}{2\rho}$ in β, as it is at realtime $\frac{1+\rho}{2\rho}$ in α. Therefore, since α and β are indistinguishable to p_1, we have $L_1' = L_1$. What is L_2 in terms of L_2'? We claim that

$$\begin{aligned} L_2 &\geq L_2' + \frac{1}{2}(\frac{1+\rho}{2\rho} - \frac{1}{2\rho}) \\ &= L_2' + \frac{1}{2}\frac{1}{2} = L_2' + \frac{1}{4} \end{aligned}$$

To see the first equality, notice that L_2' is p_2's logical clock value in β at realtime $\frac{1}{2\rho}$, and L_2 is p_2's logical clock value in β at realtime $\frac{1+\rho}{2\rho}$. Then, by the validity property stated in section 2.2, p_2's clock value must increase by at least $\frac{1}{2}(\frac{1+\rho}{2\rho} - \frac{1}{2\rho}) = \frac{1}{4}$ between realtimes $\frac{1}{2\rho}$ and $\frac{1+\rho}{2\rho}$.

[[[PICTURE]]]

Now, to recap, we have shown that $L_1' = L_1$, and $L_2' \leq L_2 - \frac{1}{4}$. Thus, at realtime $\frac{1+\rho}{2\rho}$ in α, the clock skew between p_1 and p_2 is $\Delta = L_1 - L_2$. Also, at realtime $\frac{1}{2\rho}$ in β, the clock skew between p_1 and p_2 is $\Delta' = L_1 - L_2' = L_1 - L_2' \geq L_1 - L_2 + \frac{1}{4} = \Delta + \frac{1}{4}$. Now, it is clear that $\max(|\Delta|, |\Delta'|) \geq \frac{1}{8}$. Thus, we have shown that at the end of either α or β, the clock skew between p_1 and p_2 is at least $\frac{1}{8}$. This lower bound holds for any clock sync algorithm A.

5 Proving $f(1) = \Omega(\frac{\log n}{\log \log n})$

Consider once again the line network on n nodes defined earlier. The scaling argument from section 4.2 can be easily extended to show that for any algorithm A, if there is an execution α of A such that two neighboring nodes have Δ clock skew at the end of α, then there is an execution β of A such that the same two neighbors have $\Delta + \frac{1}{4}$ clock skew at the end of β. What can we say about the skew between non-neighboring nodes?

5.1 Add Skew Lemma (ASL)

Let p_i and p_j be two arbitrary nodes, with $j > i$. Let α be an execution of A satisfying the following properties:

1. The realtime duration of α is at least $C_1(j-i)$, for a sufficiently large constant C_1. Let γ be the suffix of α of realtime duration $C_1(j-i)$.
2. The message delay between all neighbors is $\frac{1}{2}$ during γ.
3. The hardware clock rates of all nodes is 1 throughout γ.
4. At the end of α, we have $L_i^\alpha - L_j^\alpha = \Delta$, for some Δ.

Then, we claim there is an execution β of A such that

1. The message delay between any pair of neighbors in β is between $\frac{1}{4}$ and $\frac{3}{4}$.
2. The hardware clock rates of all nodes is between 1 and $1 + \frac{\rho}{2}$ throughout β.
3. At the end of β, we have $L_i^\beta - L_j^\beta \geq \Delta + C_2(j - i)$, for some constant C_2.

That is, β *increases the skew* between p_i and p_j by $C_2(j - i)$, as compared to α. This is called the Add Skew Lemma in the Fan, Lynch paper, and is proved in section 6 of that paper. We won't go through the full proof here; see the paper for details. We simply give the intuition for the proof. The Add Skew Lemma is basically just an enhanced version of the scaling argument we gave in section 4.2. To increase the skew between p_i and p_j, we create an execution β containing the same actions as α. In β, we increase the hardware clock rates of all nodes p_1 through p_i to $1 + \rho$, for some amount of time. We also increase p_{i+1} through p_{j-1}'s hardware clock rates to $1 + \rho$. But for each p_k, $i \leq k \leq j - 2$, we let p_k's hardware clock run fast somewhat *longer* than we let p_{k+1}'s hardware clock run fast. We don't change the hardware clock rates of nodes p_j through p_n. In addition, for $i \leq k \leq j$, we *increase* the message delay from p_k to p_{k+1}, and *decrease* the message delay from p_{k+1} to p_k, and for $1 \leq k \leq i - 1$, we decrease all the message delays between p_k and p_{k+1}. Lastly, we change the realtime at which actions occur in β so that any action at any node occurs at the same hardware clock value of that node in α and β. Note the similarity between this construction and the scaling argument, where we also sped up some nodes, changed the realtimes at which actions occurred, and adjusted message delays asymmetrically. The net effect of these changes is that α and β are indistinguishable, to every node $p_1, \ldots, p_n$. Now, at a certain realtime, called T' in the paper, node p_i has the same logical clock value at T' in β as it has at realtime T in α. But we can show using the validity condition from section 2.2 that p_j's logical clock value at T' in β is at least $C_1(j - i)$ less than its clock value at T in α. Thus, the clock skew between p_i and p_j at realtime T' in β is at least $C_1(j - i)$ larger than their clock skew at realtime T in α. Furthermore, we can show that all message delays between neighbors in β are between $\frac{1}{4}$ and $\frac{3}{4}$. This proves the Add Skew Lemma.

[[[PICTURE]]]

5.2 Bounded Increase Lemma

When we apply the Add Skew Lemma, we can increase the skew between two arbitrary nodes p_i and p_j by an amount $C_1(j - i)$. Therefore, the skew between a pair of neighbors between p_i and p_j must increase by at least C_1. Can we simply apply the Add Skew Lemma $\Omega(\frac{\log n}{\log \log n})$ times, to make the skew between some pair of neighbors $\Omega(\frac{\log n}{\log \log n})$? No! The reason is, in order to apply the ASL to an execution α, to produce β with greater skew, α must satisfy certain preconditions. For example, the message delay between all neighbors must be $\frac{1}{2}$. But execution β doesn't satisfy those preconditions. For example, all we can say about β is that the message delay between all neighbors is between $\frac{1}{4}$ and $\frac{3}{4}$. Therefore, we can't apply the ASL to β, and in general, we can't apply the ASL to the same execution more than once. Yet, we can still use the ASL multiple times to produce superconstant skew between some neighbors. This is described in the next section. First, we describe the Bounded Increase Lemma, which is used in the next section.

Intuitively, the BIL says that no node can increase its logical clock too quickly. The intuition is that if a node p_i increased its clock quickly by a large amount, say $2f(1)$, then its neighbor p_{i+1} won't have time to find out about p_i's increase. Before p_i's increase, its clock skew with p_{i+1} was

at most $f(1)$, by the f-gradient property. That is, $L_i \geq L_{i+1} - f(1)$, where L_i and L_{i+1} are p_i and p_{i+1}'s logical clock values before p_i's increase. After the increase, we have $L'_i > L_i + 2f(1) > L_{i+1} + f(1) = L'_{i+1} + f(1)$, where L'_i and L'_{i+1} are p_i and p_{i+1}'s clock values immediately after p_i's increase. However, that means p_i's clock value is more than $f(1)$ greater than L_{i+1}'s clock value, contradicting the gradient property. Thus, p_i can't increase its clock very quickly.

A bit more formally, let α be any execution of A satisfying the following properties:

1. All nodes have hardware clock rate between 1 and $1 + \frac{\rho}{2}$ throughout α.
2. All neighbors have message delay between $\frac{1}{4}$ and $\frac{3}{4}$.
3. α has duration at least $\frac{1}{\rho}$.

Then for any $t \geq \frac{1}{\rho}$, we have $L_i(t+1) - L_i(t) \leq C_4 f(1)$, for some constant C_4. That is, in one second of realtime, p_i doesn't increase its logical clock by more than $C_4 f(1)$.

This is shown by creating an execution β, in which p_i's hardware clock rate is increased to $1 + \rho$, and the hardware clock rates of other nodes are not changed. Then, the realtime of actions and the message delays between nodes are adjusted to make α and β indistinguishable to all the nodes. However, by increasing p_i's hardware clock rate, this has the effect of increasing the skew between p_i and its neighbors. In particular, the amount of increased skew is related to the rate of increase of p_i's logical clock. For the right setting of constants, we can show that the skew is increased by $2f(1)$ in β as compared to α. As we noted above, this contradicts the gradient property. See section 7 of [FL] for more details.

5.3 Putting It Together

To prove the lower bound on $f(1)$, we invoke the Add Skew Lemma multiple times, and also use the Bounded Increase Lemma. Suppose by induction that after k applications of the ASL, we produce an execution α_k with the following properties:

1. There is a set of nodes $p_{i_k}, \ldots, p_{j_k}$, for some $1 \leq i_k < j_k \leq n$, such that $L_{i_k} - L_{j_k} \geq C_5 k(j_k - i_k)$, for some constant C_5, where L_{i_k} and L_{j_k} are the logical clock values of p_{i_k} and p_{j_k} at the end of α_k.
2. Let γ_k be the suffix of α_k with realtime duration $\frac{C_5(i_k - j_k)}{2C_4 f(1)}$. The delay of all messages between neighbors during γ_k is $\frac{1}{2}$, and all nodes have hardware clock rates 1 throughout γ_k.
3. The delays of all messages between neighbors in α_k is between $\frac{1}{4}$ and $\frac{3}{4}$, and the hardware clock rates of all nodes is between 1 and $1 + \frac{\rho}{2}$ throughout α_k.

Intuitively, these properties say that at the end of iteration k, we have an execution α_k such that the set of neighboring nodes between p_{i_k} and p_{j_k} have, *on average*, $C_5 k$ amount of clock skew. Furthermore, there is a long suffix γ_k of α_k which is *well-behaved*, in the sense that the message delay between neighbors is $\frac{1}{2}$, and the hardware clock rates of nodes are all 1. In particular, γ_k satisfies the preconditions of the ASL (for an appropriate choice of i and j in the ASL, which we describe later). Also, α_k satisfies the preconditions of the BIL.

How do we use α_k? Let $\alpha_k = \beta_k \circ \gamma_k$. That is, α_k has prefix β_k, and suffix γ_k. Define

$$n_{k+1} = \frac{C_5(j_k - i_k)}{2C_1 C_4 f(1)} \tag{1}$$

Now, since neighbors between p_{i_k} and p_{j_k} have average skew $C_5 k$, then there must be two nodes i_{k+1} and $j_{k+1} = i_{k+1} + n_{k+1}$ with $L_{j_{k+1}} - L_{i_{k+1}} \geq C_5 n_{k+1} k$, where $L_{i_{k+1}}$ and $L_{j_{k+1}}$ are $p_{i_{k+1}}$ and $p_{j_{k+1}}$'s logical clock values at the end of α_k, resp. We can check that we can apply the ASL, with i and j in the lemma set to i_{k+1} and j_{k+1}, resp., to produce an execution $\alpha'_k = \beta_k \circ \gamma'_k$, such that $L'_{j_{k+1}} - L'_{i_{k+1}} \geq C_5 n_{k+1} k + C_2 n_{k+1}$, where $L'_{i_{k+1}}$ and $L'_{j_{k+1}}$ are $p_{i_{k+1}}$ and $p_{j_{k+1}}$'s logical clock values at the end of α'_k. Now, we extend α'_k by a realtime duration of $\frac{C_5(j_{k+1}-i_{k+1})}{2C_4 f(1)}$. In this extension, we set the message delay between all neighbors be $\frac{1}{2}$, and set the hardware clock rates of all nodes to be 1. We call this extended execution α_{k+1}. We can check that α_{k+1} satisfies the preconditions of the Bounded Increase Lemma. Now, let $L''_{j_{k+1}}$ and $L''_{i_{k+1}}$ are $p_{i_{k+1}}$ and $p_{j_{k+1}}$'s logical clock values at the end of α_{k+1}. Then we can check that

$$\begin{aligned} L''_{j_{k+1}} - L''_{i_{k+1}} &\geq C_5 n_{k+1} k + C_2 n_{k+1} - C_4 f(1) \frac{C_5(j_{k+1} - i_{k+1})}{2C_4 f(1)} \\ &= C_5 n_{k+1} k + C_2 n_{k+1} - \frac{C_5(j_{k+1} - i_{k+1})}{2} \\ &= C_5 n_{k+1}(k+1) \end{aligned}$$

where to obtain the last equality, we set $C_5 = \frac{2C_2}{3}$. The last inequality follows because $p_{i_{k+1}}$'s logical clock value can increase by at most $C_4 f(1) \frac{C_5(j_{k+1}-i_{k+1})}{2C_4 f(1)}$ during the extension, by the BIL. Thus, α_{k+1} satisfies the first of the inductive properties about the α_k's. We can also check that it satisfies the other two properties. Therefore, we have managed to inductively create execution α_{k+1}, in which the neighboring nodes between nodes $p_{i_{k+1}}$ and $p_{j_{k+1}}$ have average skew $C_5 k(j_{k+1} - i_{k+1})$.

[[[PICTURE]]]

We can continue the above inductive construction as long as $j_k - i_k \geq 1$. Now, we can start the construction with $j_1 - i_1 = n$, and by equation 1, we have $j_{k+1} - i_{k+1} = \frac{j_k - i_k}{C_6 f(1)}$, for some constant C_6. Thus, we can continue the construction for $\log_{C_6 f(1)} n$ iterations. However, on the $(\log_{C_6 f(1)} n)$'th iteration, we have two neighboring nodes with logical clock skew equal to $C_5 \log_{C_6 f(1)} n$, by the first condition in the inductive hypothesis. By the gradient property, we then have $C_5 \log_{C_6 f(1)} n \leq f(1)$. Rearranging the inequality, this says that $(C_7 f(1))^{f(1)} \geq n$, for some constant C_7. Solving for $f(1)$, we get $f(1) = \Omega(\frac{\log n}{\log \log n})$.

6 Energy Constrained Clock Synchronization

Attiya, Hay, Welch

External synchronization, based on one or more "source clocks".

They consider a tradeoff, broadcasting with smaller or larger power. Larger power means fewer hops, which means more accurate clock synch. They prove a result that yields optimal clock synch, for given energy constraints.

6.1 Introduction

Power needed to broadcast to distance d is γd^β, where $\beta \geq 1$ is the distance-power gradient, $\gamma > 0$ is the transmission quality parameter

Clock skew: Max difference between logical clock reading and real time. Clock skew depends on uncertainty about delay of delivering messages between the nodes. Thus, clock skew increases as number of intermediate hops increases.

Try to minimize power: To save energy usage. To help avoid interference.

This paper relates the energy constraints with the optimal skew that a clock synch algorithm can guarantee. Given individual energy budgets, they get exact bounds on the optimal skew that can be achieved, as a f unction of uncertainty in message delays.

Optimal skew = minimum depth of a particular spanning forest rooted at the time sources, of the topology graph induced by the energy budgets.

Upper bound: They show how to construct a shallow forest, with minimum depth. Use simple BFS algorithms. Then propagate the time down the trees. Skew = sum of the uncertainties along the path.

Lower bound: Based on shifting techniques.

They get a nice closed form. Situation is better than in Halpern et al. results for "internal" clock synch (with no external source)—they didn't get a closed form solution.

Survey by Elson and Romer may be interesting. RBS broke new ground, by exploiting nature of wireless bcasts. Other papers study tradeoffs between accuracy and energy, but measure energy expenditure in terms of number of messages sent.

6.2 The system model

n nodes in plane Stationary. Bcast has distance d. Recipients are the nodes within distance d of the sender. Power is γd^{β}. Proportional to the energy consumed when bcasting to distance d.

Events Hardware clock HC_i Function from real time to hardware clock time, write as $t + o_i$, where o is the offset. Assumes the hardware clocks have no drift, so the offsets are constant.

Deterministic state machine. But with (nondeterministic) set of possible initial states. Also has final states (but why?). Transition function takes current state, HC, and current event (input or internal) and produces a new state and possibly a message to be bcast to a certain distance.

Hmm...so it's interrupt-driven only. But how do the internal events get generated? Timers get set somehow by the transition function? I suppose that the state must have deadline variables as usual...

History: Alternating sequence of states and pairs (event, HC-value). HC values are strictly increasing, for each node.

Timed history: History plus a real time for each pair. They define the value of a variable at a real time t to be the last value that occurs at that time.

Execution: A set of timed histories for all the nodes. Assume bcasts are reliable.

6.3 Problem statement

$LC_i = HC_i + adj_i$, adjustment variable

Source nodes have perfect time, ST. $LC_i(t) = ST(t) = t$ for all source nodes i.

Every node is supposed to set the adjustment vars to try to minimize the difference between its clock and the source time.

They have the nodes enter final states when they are "done synchronizing". When they halt, they should all be synchronized to external time, to within some epsilon (must be the same epsilon in all executions).

Given an energy bound for each node, $energy_i$, then can compute its neighborhood N_i. This is the set of nodes within Euclidean distance that can be reached by the given power; well, actually, the neighborhood is determined bidirectionally—both nodes must be able to reach each other within their given powers.

So is this the only way they use the power information? To determine the neighborhoods? That gives rise to an undirected graph describing the single-hop neighbor relationship. Assume this graph is connected. (They say strongly connected, but of course this is an undirected graph, so they must mean just "connected".)

Each undirected link gets delay delta and uncertainty (plus or minus) u. They regard the uncertainties as weights on the edges. Then they measure the cost (uncertainty) of a path by the sum of the uncertainties along the edges.

Define: Minimum-uncertainty path between two nodes Minimum-uncertainty path between a node and some source.

They want to minimize $maximum_i$ (skew of LC_i). In fact, their algorithm minimizes the clock skew for each node separately.

Comments on their assumptions: Bidirectional links justification: Message delays and uncertainties mostly due to processing in the nodes and contention in the physical layer. Nodes have similar hardware/software. Contention similar within a neighborhood. Delays between sender and different receivers can be quite different. Different assumption from RBS. Justify by saying the receive times can be different, and may need different numbers of retransmissions.

6.4 Multi-hop broadcast-based clock synch

4.1. A generic synchronization algorithm

Assumes we have the spanning forest already, each source the root of a tree. Each source sends its time along its tree. Other nodes adopt the time, adjusted by the median delay along the path. Trivial algorithm.

The max clock skew for any node is the cost of its path in this forest—the sum of the uncertainties. So, obviously, what we need is to construct the forest to minimize this max cost.

4.2. Precomputing a BFS forest

Centralized or distributed. Distributed: Nodes determine their neighbors by doing bcasts and waiting for acks. Then use a standard distributed BFS algorithm.

4.3. Computing the BFS on-the-fly

This doesn't seem too interesting in the static setting they describe. But it might make sense in a more dynamic, say mobile setting.

A standard relaxation algorithm keeps recalculating the best path. And at the same time, calculates the clock adjustment corresponding to that path.

They redo the standard argument for relaxation algorithms (in Lemma 1).

6.5 Optimality

Claim that the synchronization achieved over a shallow spanning forest is optimal for the topology graph. More specifically, for each node i, they prove a lower bound of u_i^{min} for the guaranteeable skew. Why isn't this obvious? Because the model doesn't rule out "overhearing", which could potentially yield algorithms with lower skew then the spanning forest could. But guaranteeable lower skew? It seems like overhearing is accidental.

They state the standard shifting lemma from LL84. Then they use this to prove their main optimality result, Theorem 3. It focuses on getting the worst skew for a particular node i. Start with an execution with exactly median delays. Now they consider two cases: $LC_i \leq ST$ Shift every node j later by u_j^{min}, the best uncertainty for j. Nice little argument shows this leaves the delays within bounds. A picture would be good to show this. $LC_i > ST$ Similar argument, but shift earlier. In either case, the resulting skew is $\geq u_i^{min}$ (using the shifting lemma).

6.6 Discussion

Main contribution: Method of picking the multi-hop bcasts in a way that minimizes the worst-case accumulated uncertainty, subject to energy constraints. "Shallow energy-constrained spanning forests"

Future work: Use the same forest construction, but consider other ways of estimating clocks and combining info, perhaps optimizing other measures (e.g., expected skew instead of worst-case). Optimizing skew on a per-execution basis (for changing topologies/mobile case?)