Massachusetts Institute of Technology
6.895: Advanced Distributed Algorithms
Professor Nancy Lynch

Lecture 6
February 27, 2006

Time Synchronization

Readings:
Elson, Girod, Estrin. Fine-grained network time synchronization using reference broadcasts.
Karp, Elson, Papadimitriou, Shenker. Global synchronization in sensornets.

These notes cover the first of two lectures on the topic of time synchronization. In particular, this lecture describes a system for time synchronization known as Reference Broadcast Synchronization, or *RBS* for short.

Section 1 presents the motivations behind RBS, discusses applications for which time synchronization is essential, and brings up shortcomings of prior approaches. Section 2 describes the major sources of clock synchronization error which RBS is intended to minimize or avoid. Section 3 dives into detail about how the RBS approach works. Finally, section 4 presents techniques and models for improving RBS, based on the paper by Karp *et. al.*

1 Introduction

A notion of time is necessary for the correct and efficient operation of many sensornet applications. Today we discuss an algorithm called Reference Broadcast Synchronization (*RBS*) for giving nodes a common sense of time, which is particularly well suited to the characteristics and constraints of a sensornet.

We first discuss some applications of time sync in a sensornet. Consider the following.

1. *Set up TDMA* In TDMA, nodes are assigned time *slots* when they can broadcast, such that no other nodes are assigned the same slot. But for this to work, all nodes need to know when the slots start, and so their clocks need to be aligned.
2. *Data fusion* Sensors timestamp some readings, then convergecast their readings to a central node, merging the data along the way. For the merging to make sense, the timestamps need to be consistent.
3. *Measure time of flight of sound* This is compared against another signal like RF, in order to do ranging. The sender and receiver of the sound signal can use a clock to measure its travel time.
4. *Compute velocity of target* This is used by sensors to track a target. Each sensor records when it saw the target, then the sensors exchange their time readings, divide this into their distance, and compute the target's velocity.
5. *Distribute a beam forming array* A group of nodes can cooperate to act as one giant sender or receiver antenna. One example is the array of radio telescopes NASA uses. But the nodes need synchronized clocks to match up the signals they receive.

6. *Cryptography, logging, debugging* These are more traditional applications, which also apply in the wired setting.

We observe the following properties of these applications.

1. Some applications, like TDMA, need *very accurate* time sync between nodes, on the order of microseconds.
2. Often, nodes don't need to know the actual time, just their *relative time.* That is, when my clock is x, what is your clock?
3. Nodes don't need to know the correct time all the time, only when certain *interesting events*, like a target passing by, occur.

RBS is a time sync algorithm specifically adapted to meet these needs. RBS can sync nodes to within 1.5-6 microseconds of each other. Instead of having every node compute a global clock value, RBS gives nodes a way to *translate* their clock values to another node's time values. Also, RBS doesn't require the nodes to keep their clocks in sync all the time, which saves energy by letting nodes sleep most of the time. Instead, *after* an interesting event occurs, the nodes run RBS, and then they can *infer* the time of the earlier event.

Not only do most traditional time sync algorithms, e.g. Network Time Protocol, not meet these requirements, we simply can't run many of them in a sensornet. Traditional algorithms might require nodes to send too many messages, draining their battery. They might need an external time source, like GPS, which is too large and energy intensive to put on small sensors. They might also require an infrastructure, such as a spanning broadcast tree, which is difficult to construct or maintain in a radio network due to its ad hoc or dynamic nature.

What makes RBS work so well? It rests on cleverly exploiting the broadcast nature of communication in a radio network. To see this, we first look at what causes error in a clock sync algorithm.

2 Sources of Clock Sync Error

Most traditional clock sync algorithms work as follows. Imagine a server and a client node, and the client trying to sync to the server. The server sends messages to the client, who echos them

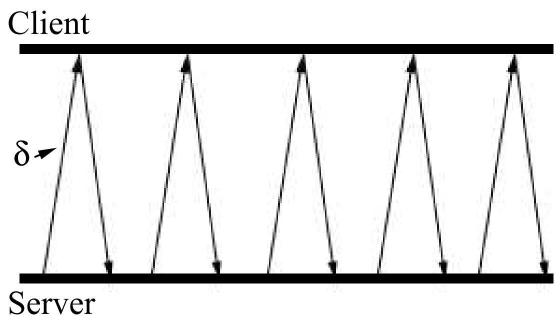

Figure 1: Example of round trip time averaging

as soon as he receives them. This produces a *round trip time* for each message the server sends. By dividing these by 2 and averaging them, the nodes can infer the message delay from server to client, δ. Now, the server sends the client the server's current time t, and the client sets his clock to $t + \delta$. See Figure 1.

This scheme only works well if the message delay from server to client is fairly constant. Otherwise, if we estimate the delay to be $\hat{\delta}$, but the actual delay in the final message from the server to the client is δ which is very different from $\hat{\delta}$, the client will have a poor estimate of the server's time. What can cause variations, i.e. *uncertainty*, in the message delay? There are 4 primary causes.

1. *Send time* The server must form the message, timestamp it, then transfer the message to its network interface. The uncertainty here comes from delays in the operating system, e.g. context switches or processing time.
2. *Access time* The message must be sent over the air. The uncertainty here is waiting to access the medium, e.g. for the channel to become clear, or for the sender's TDMA slot to arrive.
3. *Propagation time* The message travels over the air from the sender to the receiver, at about the speed of light. The uncertainty in this step is on the order of nanoseconds, so negligible compared to the accuracy requirement.
4. *Receive time* The receiver must receive the message and timestamp it. This is also subject to operating system uncertainties.

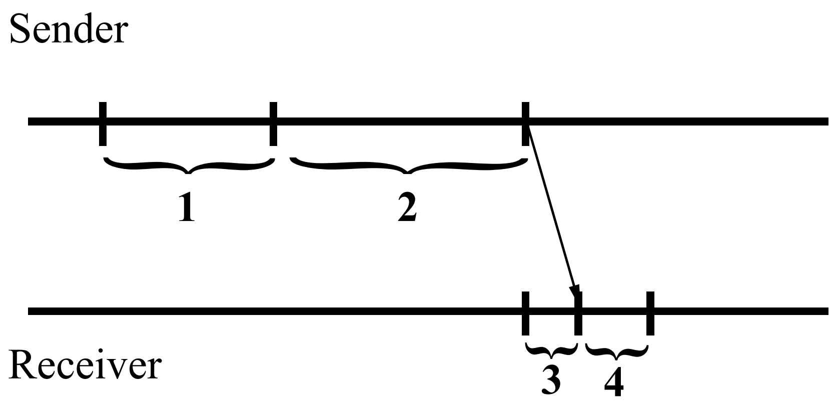

Figure 2: Causes of clock sync error

Of these, the biggest uncertainties come from the first and second steps. For example, access time depends on network traffic load, and can be highly variable. Send time depends on the load at the sender. Propagation time is actually negligible. Also, receive time uncertainty can be reduced substantially, by having the operating system timestamp the message as soon as the interrupt announcing the message reception occurs.

We call the last two sources of uncertainty the *receiver uncertainty.* It was shown by experiments that the difference between different receivers' uncertainty follows a Gaussian distribution. For one implementation on Berkeley Motes, the distribution had mean 0, and standard deviation 11.1 μsec.

3 Reference Broadcast Synchronization (Elson *et. al*)

The goal of RBS is to remove the first two sources of uncertainty, and deal only with receiver uncertainty, which is both smaller, and follows a nice distribution.

To do this, we leverage the broadcast nature of (single hop) radio networks. Consider one *broadcast domain*, i.e. a sender node and a set of nodes which can all hear the sender and hear each other.

Notice that when the sender broadcasts, all the nodes in the broadcast domain hear the message at *nearly the same time*. Thus, the reception of a message from the sender can serve as a *reference point* in time for all the receivers.

RBS uses this reference point to synchronize all the receivers with *each other*. Contrast this with traditional time sync algorithms which try to sync the receiver with the *sender*.

For example, consider two receivers p_1 and p_2. The sender broadcasts a message m. p_1 receives m when its clock is t_1, and p_2 receives m when its clock is t_2. We know that t_1 and t_2 occur at nearly the same *real time*. Therefore, the difference between p_1 and p_2's clocks is $t_2 - t_1$. So, when p_1's clock says x, p_2's clock says $x + t_2 - t_1$. Thus, p_1 has a way to *translate* his clock readings into p_2 clock readings. The process can simply be flipped for p_2 translating to p_1.

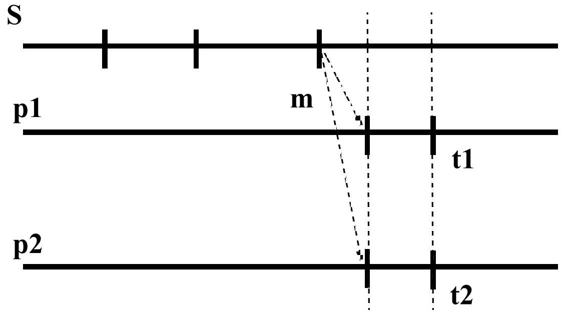

Figure 3: Use of reference message to discover clock offset

3.1 Step 1: Estimating Offsets

The basic RBS algorithm assumes that every node p_i has a clock which at real time t, has the value $t + \beta_i$. That is, the node has some physical timer which runs at the same rate as real time, but the value of the timer has offset β_i from the real time. Later, we handle the case where the clock has an offset from real time, and also doesn't run at the same rate as real time.

Our goal is that given any two nodes p_i and p_j, p_i can translate its clock value to p_j's clock value. That is, p_i can compute $\beta_j - \beta_i$. We call the latter value $\beta_{j,i}$.

Let $p_1, \ldots, p_n$ be part of a broadcast domain. We do the following.

1. The sender broadcasts s reference messages, all uniquely identifiable. Let $t_{i,j}$ be the value of p_i's clock when it receives the j'th message.
2. Each node p_i sends its readings $t_{i,j}, j \in \{1, \ldots, s\}$ to all the other nodes.
3. A node p_i computes its offset from $p_j, j \in \{1, \ldots, n\}$, as $\beta_{i,j} = \frac{1}{s}\sum_{k=1}^{s}(t_{j,k} - t_{i,k})$. To translate its clock value to p_j's, p_i simply adds $\beta_{i,j}$ to its own clock value.

We say the synchronization *accuracy* between p_i and p_j is how close $\beta_{i,j}$ is to the actual offset between p_i and p_j's clocks, $\beta_i - \beta_j$.

The reason for sending s messages instead of just one is that it improves accuracy. Indeed, each $t_{j,k} - t_{i,k}$ has some error, which we saw was the receiver error, with a Gaussian distribution with mean 0. So, by averaging these for many k, the error is decreased. In practice, when s was set to 30, the accuracy between i and j was improved from 11 μsec to 1.6 μsec when there were 2 receivers. When there were 20 receivers, the worst accuracy between any pair of them, called the *group dispersion*, was 5.6 μsec.

3.2 Step 2: Estimating Clock Skew

We assumed above that clocks have no *skew*. That is, their rate of increase is equal to real time. In reality, the crystal oscillators used in the nodes' timers can deviate by 1 part in 10^6 to 10^4. This means that in one second of real time, a node's clock offset can change by 1 to 100 μsecs. Since we require μsecond precision, then we must compensate for the skew.

For simplicity, we assume that the rate of increase of each node p_i's clock is a constant α_i close to 1. So, at real time t, p_i's clock reads $\alpha_i t + \beta_i$. Now, when p_i's clock value reads t_i, then p_j's clock will read $\alpha_{i,j} t_i + \beta_{i,j}$, for some constants $\alpha_{i,j}, \beta_{i,j}$. The goal is for each node to compute these constants.

To do this, imagine two nodes p_1 and p_2 which receive a series of s reference messages. Let $t_{1,k}$ and $t_{2,k}$ be the value of p_1 and p_2's clock when they receives the k'th message, and let $\beta_{1,2}^k$ be the k'th offset value p_1 computes. That is, $\beta_{1,2}^k = t_{2,k} - t_{1,k}$. Now, create a set of s data points, $(t_{1,k}, \beta_{1,2}^k), k = 1, \ldots, s$. Let $\hat{f}_{1,2} = \alpha_{1,2}x + \beta_{1,2}$ be the *least squares linear regression* of these data points. That is $\hat{f}$ is the linear function which best fits the data points with respect to root mean square error. Then, node p_1 will use function $f_{1,2}$ to translate from its clock readings to p_2's.

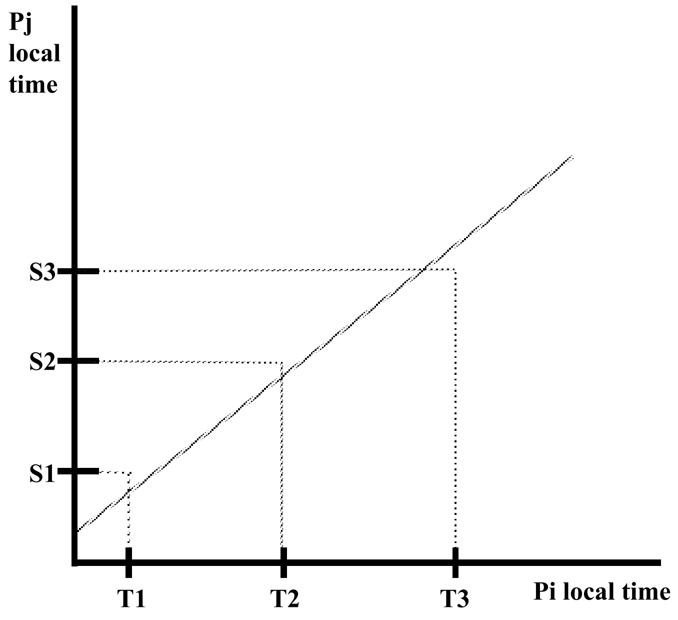

Figure 4: Correcting for skew

Now, we can modify the above RBS algorithm to take care of skew and offset, simply by having each pair of nodes p_i and p_j in a broadcast domain compute $\alpha_{i,j}$ and $\beta_{i,j}$.

3.3 Post-Facto Synchronization

As we mentioned, for many sensornet applications, nodes don't need timing information all the time. We would like to turn off the nodes until they wake up to record some event, then afterwards, *post-facto* the event, synchronize their clocks and timestamp the event.

RBS can be used to retroactively timestamps events. We simply have nodes record the events they observe with their local clock value. Then, after all the observations are done, the nodes synchronize their clocks using the RBS algorithm, to compute the translation coefficients $\alpha_{i,j}$ and $\beta_{i,j}$. Now, any node p_i can use these values to convert its timestamped values to p_j's timestamped values.

Note that post-facto synchronization can't be done with algorithms like NTP, because NTP requires that nodes synchronize their clocks all the time, and therefore be constantly awake.

3.4 Performance Results

RBS performs extremely well in practice, and meets the needs of μsec precision required by ranging and TDMA protocols. For example, an implementation of RBS in user mode on an Ipaq using 802.11 MAC achieves mean clock error of 6.29 μsec under light traffic load, compared to 51.18 μsec for NTP on the same platform. Under heavy traffic, RBS had error 8.44 μsec, while NTP's error exploded to 1542.27 μsec. Note that this tolerance of heavy traffic load is a particularly desirable characteristic of RBS. In sensornet applications, data may be highly bursty. Thus, using time synchronization methods that succumb to high traffic loads (like NTP) introduces inaccuracy precisely when accuracy is most needed!

Furthermore, when the Ipaq was tweaked so that RBS ran in kernel mode, and RBS packets were timestamped by the network interface, instead of passing them up to be timestamped by the operating system, RBS achieved a error of $1.85 \pm 1.28\mu$sec, which is nearly the precision of the Ipaq's hardware clock.

3.5 Multihop RBS

Up till now we've only considered RBS in one broadcast domain, where a set of receivers can all hear one sender. But RBS also works when we have a multihop network where some receivers can't hear some senders. For simplicity, assume once again that the nodes' clocks have no skew. That is, all the coefficients $\alpha_i = 1$. It's not hard to handle clocks with skew, but we won't discuss that today.

To run RBS in a multihop network, we make use of *gateway* nodes, such as node p_2 in Figure 5. Suppose we want to translate between p_1 and p_3's clocks. Since there's no sender which p_1 and p_3 can both hear, the single hop RBS algorithm described above won't work. However, receiver p_2 can hear both senders p_a and p_b. If p_2 informs p_1 of p_2's offset from p_3, p_1 has all the tools it needs to translate into p_3's time.

Let $\beta_{1,2}$ be the offset from node p_1 to p_2, and $\beta_{2,3}$ be the offset from p_2 to p_3. Then, p_1 can compute p_3's clock as $L_3 = L_1 + \beta_{1,2} + \beta_{2,3}$.

In general, for any two nodes p_i and p_j, which are connected by a path $p_i = p_{i_1}, p_{i_2}, \ldots, p_{i_u} = p_j$ in the communication graph, we can translate between their clocks by first computing $\beta_{p_{i_k}, p_{i_{k+1}}}$, for all $k = 1, \ldots, n-1$, using the one hop RBS protocol, then write $L_j = L_i + \sum_{k=1}^{u-1} \beta_{p_{i_k}, p_{i_{k+1}}}$.

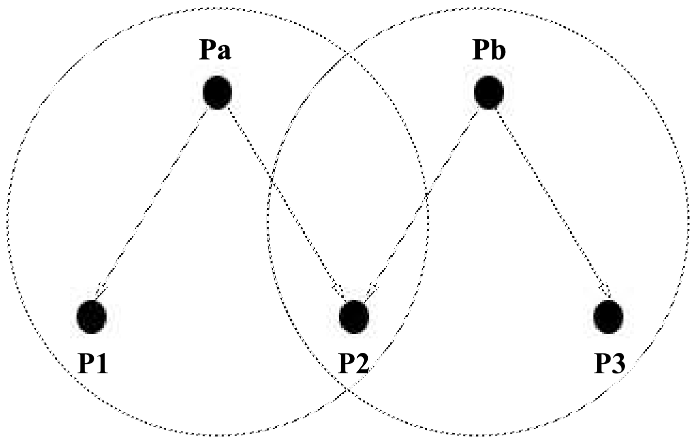

Figure 5: Multihop RBS

Multihop RBS works well in practice. As we increase the number of hops, the synchronization accuracy between faraway nodes degrades. If the mean error for two single hop nodes is ϵ, then the mean error for two n hop nodes is in expectation $\sqrt{n}\epsilon$, due to the cancellation of positive and negative error on the path.

To do multihop translation, we can in fact use several paths between p_i and p_j. This might lead to smaller and lower variance error because the errors along different paths could cancel out even more. This is the subject of the next paper.

4 Improving Multihop RBS (Karp *et. al*)

Consider a multihop network. Let p_1 and p_2 be two nodes, and P_1 and P_2 be two different paths between p_1 and p_2. RBS allows us to translate p_1's clock value to p_2's using either path. However, there is no guarantee of *consistency.* That is, there is no guarantee that we get the same translation from path P_1 and P_2. A second problem is the following. There is random error in the translation along every hop of the path we pick, and each such error is characterized by a probability distribution with some variance. However, the two different paths may have different variances, and the current RBS scheme doesn't guarantee picking the path with *minimum variance.* We want to address these two problems. We first present a theoretical model for how RBS operates.

4.1 A Theoretical Model for RBS

In the following discussion, we assume that there is no clock skew, so that the translation between two nodes' clocks is simply an offset. Karp also gives algorithms for dealing with clock skew, but we won't discuss those today.

Intuitively, RBS works by sending *signals* which are heard by a set of nodes at approximately the same time. To model this, let $P = \{p_1, \ldots, p_n\}$ represent a set of n nodes (we also call these *receivers*), and $S = \{s_1, \ldots, s_m\}$ a set of signals. Form a *bipartite graph* with the p_i's as the left vertex set, and the s_k's as the right vertex set. Draw an edge between p_i and s_k if p_i heard the signal s_k, as shown in Figure 6. We call the set of edges E.

For $k = 1, \ldots, m$, let U_k denote the real time when signal s_k was sent. Let T_i represent node p_i's clock offset from real time. That is, at real time t, p_i's clock reads $t + T_i$. Let e_{ik} be a random

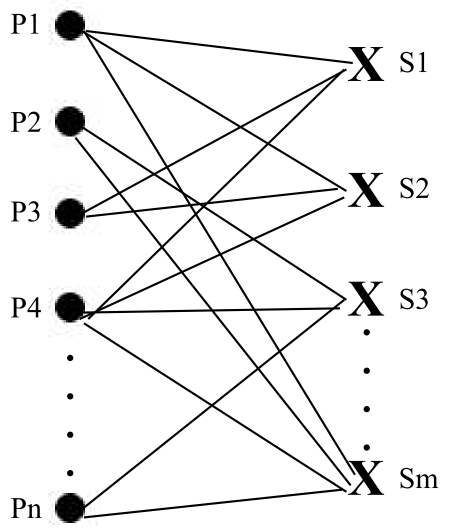

Figure 6: Modeling RBS as a bipartite graph

variable with mean 0 and variance V_{ik}, which is the error in p_i's reception of s_k. Lastly, let y_{ik} represent p_i's clock value when it received s_k. Therefore, $y_{ik} = U_k + T_i + e_{ik}$.

We first present an algorithm for computing a minimum-variance estimate of $T_i - T_j$, for every p_i, p_j. It turns out that this method also ensures that the estimates are consistent. That is, if we compute $\beta_{i,j} = T_i - T_j$, then $\beta_{i,j} + \beta_{j,k} = \beta_{i,k}$, for any i, j, k.

Later, we present an algorithm for computing a *maximum-likelihood* joint estimate for the T_i's. That is, given a set of observations $\{y_{ik}\}$, we compute the set of values $\{T_i\}$ which are most likely to have given rise to the observations. These T_i's automatically define a set of consistent $\beta_{i,j}$ values.

Notice that, as per discussion in class, since this algorithm is a gloabl optimization problem, it makes the most sense to imagine this being run as a centralized algorithm. After collecting data on the y_{ik}'s from all nodes, a central node can efficiently run this algorithm to find the minimum-variance estimates. Efficient decentralized algorithms may exist, but they are not mentioned in the paper.

4.2 Minimum Variance Pairwise Synchronization

Let $G = (P \cup S, E)$ be the bipartite graph described in the previous section, and let p_1 and p_2 be any two nodes. We begin by characterizing the set of *all unbiased estimators* of $T_i - T_j$, that is, all random variables X such that $E[X] = T_i - T_j$. Then, out of this set, we pick the estimator with minimum variance.

Consider a path of the form $p_{i_1}, s_{k_1}, p_{i_2}, s_{k_2}, \ldots, s_{k_t}, p_{i_{t+1}}$, where $p_{i_1} = p_1$ and $p_{i_{t+1}} = p_2$, such that each pair (p_{i_j}, s_{k_j}) and $(s_{k_j}, p_{i_{j+1}})$ are adjacent in G. We can use this path to estimate $T_1 - T_2$, by computing $y_{i_1,k_1} - y_{i_2,k_1} + y_{i_2,k_2} - \ldots - y_{i_{t+1},k_t}$. Since $y_{ik} = U_k + T_i + e_{ik}$, this simpifies to $T_1 - T_2 + e_{i_1,k_1} - e_{i_2,k_1} + e_{i_2,k_2} - \ldots - e_{i_{t+1},k_t}$, which is unbiased because all e_{ik} have expectation 0.

Clearly, there may be many paths between the two nodes. We are free to select any possible path. All paths are equally *valid*, but are not equally *good*. However, notice that we can select more than one path (as shown in Figure 7) to compute the estimator. In fact, we can select any *weighted combination* of paths between p_1 and p_2, such that the weights sum to 1. This allows us to get a substantially improved estimator.

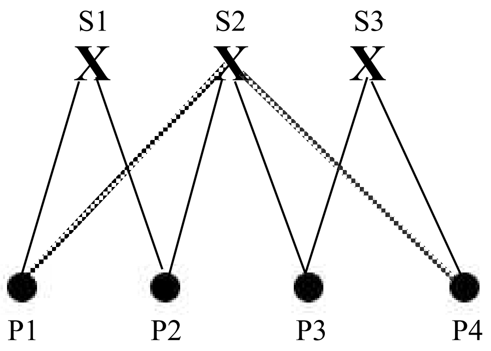

Figure 7: Multiple paths between destination nodes

Each individual path P_i with weight w_i, is a *flow* of weight w_i from p_1 to p_2. Thus, their sum is also a flow from p_1 to p_2, and has weight $\sum_i w_i = 1$. Now, let $\{f_{ik}\}$ represent the flow of weight 1 produced by the weighted combination of the paths. That is, $\{f_{ik}\}$ is an assignment of a real number to each edge $(i, k) \in E$, such that the numbers satisfy the conditions for a flow of weight 1. Then, the estimate of $T_1 - T_2$ produced by the paths is $\sum_{ik} f_{ik} y_{ik}$, where f_{ik} is positive if it goes from p_i to s_k, and negative in the other direction. The variance of estimator $\{f_{ik}\}$ is $\sum_{ik} f_{ik}^2 V_{ik}$. Let $\mathcal{F}$ represent the set of all flows of weight 1 from p_1 to p_2. What we want to do is find a flow $F = \{f_{ik}\} \in \mathcal{F}$ which minimizes the variance.

It is possible to solve this optimization problem directly, using augmenting flows. The paper gives a polynomial time approximation scheme for this. However, it turns out that finding the minimum-variance flow is equivalent to solving a certain problem about electrical networks, for which there are exact algorithms. We now describe this connection.

Consider the following model for an electrical network. Let H be a graph, such that each edge e of the network has resistance R_e. Now, choose two nodes u and v in H, and inject one unit of *current* at u, and extract one unit of current at v. At every other node, the net current is 0. For each edge e, let c_e be the current along edge e. For each node w, let $p(w)$ be the *electical potential* at node w. The currents and potentials must satisfy Kirchoff's first and second circuit laws. Now, the *effective resistance* of H is defined as the potential difference $p(u) - p(v)$.

We define $\mathcal{C}$ to be the set of all current flows, such that 1 unit of current is injected at u, and 1 unit extracted at v, and such that the flow satisfies Kirchoff's first circuit law: the current going into and out of each node is 0. We represent a particular such flow by $\{c_{ik}\}$.

It turns out, by a minimum action principle in physics, that the effective resistance between u and v is equal to

$$\min_{\{c_{ik}\} \in \mathcal{C}} \sum_{(i,k) \in E} c_{ik}^2 R_{ik}$$

Compare this to the formula for the minimum variance of an unbiased estimator for $T_1 - T_2$ which we derived earlier

$$\min_{\{f_{ik}\} \in \mathcal{F}} \sum_{ik} f_{ik}^2 V_{ik}$$

We see that the formulas are identical except for variable names. Thus, we can use a solution to the effective resistance as a solution for the minimum variance estimator. To do this, create an electrical network with node set $P \cup S$ and edge set E, such that the resistance along edge (i, k) is V_{ik}. Then, inject one unit of current at p_1 and remove one unit at p_2. Now, we want to compute the effective resistance between p_1 and p_2. This is done via *nodal analysis*, which is a way of using Kirchoff's two laws to produce a system of linear equations, the solution of which gives the currents and potentials in the network. We will not discuss the method here. We simply note that the currents in the network computed by nodal analysis define a flow F of weight 1 between p_1 and p_2 in G, and we can use F to compute both an unbiased estimator for $T_1 - T_2$ and the variance of the estimator, via the formulas given earlier.

Lastly, it turns out that currents and potentials in an electrical network satisfy the following *superposition principle.* Let e_1 and e_2 be two current vectors, giving for each node, the amount of current injected at the node. Let $c_{ik}(e)$ and $p_i(e)$ be the current flow along edge (i, k), and the potential at node i, resp., when the current vector is e. Then we have[1]

$$c_{ik}(e_1 + e_2) = c_{ik}(e_1) + c_{ik}(e_2)$$
$$p_i(e_1 + e_2) - p_j(e_1 + e_2) = ((p_i(e_1) - p_j(e_1)) + (p_i(e_2) - p_j(e_2))$$

Let A_{ij} be the minimum variance unbiased estimator of $T_i - T_j$. Then, using the superposition principle, we can show that for any $i, j, k \in P$, we have $A_{ij} + A_{jk} = A_{ik}$. That is, the minimum variance unbiased estimators are also consistent.

4.3 Maximum Likelihood Offset Assignments

Instead of computing $T_i - T_j$ for each pair p_i, p_j, we can also try to compute the T_i's individually. Notice that if we do this, then the set of differences $\{T_i - T_j\}_{i,j}$ are guaranteed to be consistent.

To compute the T_i's, we assume that the e_{ik}'s are all independent Gaussian variables with variance V_{ik} (notice that we didn't need to assume the e_{ik}'s were Guassian in in the minimum variance formulation). Then, since $y_{ik} = U_k + T_i + e_{ik}$, the probability of observing a particular value y_{ik} is $\frac{1}{\sqrt{2\pi V_{ik}}} e^{\frac{(y_{ik} - U_k - T_i)^2}{2V_{ik}}}$, and the probability of observing the set of values $\{y_{ik}\}$ is

$$\prod_{ik} \frac{1}{\sqrt{2\pi V_{ik}}} e^{\frac{(y_{ik} - U_k - T_i)^2}{2V_{ik}}} \tag{1}$$

Now, we let the U_k's and T_i's be the independent variables. Also, set $C_{ik} = \frac{1}{V_{ik}}$. To find the values of the U_k's and T_i's which minimizes the above expression, we differentiate equation 1 with respect to the U_k's and T_i's, to obtain, for each k

$$\sum_i C_{ik}(U_k + T_i) = \sum_i C_{ik} y_{ik} \tag{2}$$

and for each i

$$\sum_k C_{ik}(U_k + T_i) = \sum_k C_{ik} y_{ik} \tag{3}$$

[1]Note that for consistency with the flow notation, I changed the notation for currents and potentials from the Karp paper.

To solve this system of linear equations, we can use the following iterative method. In each iteration, for each k, we compute

$$U_k \leftarrow \frac{\sum_i C_{ik}(y_{ik} - T_i)}{\sum_i C_{ik}}$$

Then, for each i, we compute

$$T_i \leftarrow \frac{\sum_k C_{ik}(y_{ik} - U_k)}{\sum_k C_{ik}}$$

Each iteration of the above procedure reduces the value of

$$\sum_i \sum_k (y_{ik} - U_k - T_i)^2$$

so that eventually the method converges to a solution of equations 2 and 3.

4.4 General Contributions

The Karp paper presents many ideas, and makes many interesting claims. Unfortunately, the paper does not really present any proofs for such claims. The general big-picture idea to take away from the Karp paper is that RBS-style clock synchronization can be improved dramatically by using multiple paths to compute an estimator. For example, consider an LxL grid of nodes. In traditional RBS without multipath techniques, the variance along a path will be O(L). However, by using the techniques from this paper, we can improve this to O(log L). Clearly, the improvments can be significant.