# Improving Clock Synchronization in Large Network

Arieh Bibliowicz
vainolo@vipe.technion.ac.il

March 16, 2005

**Abstract**

Most clock synchronization algorithms rely on direct communication between processors to deduce the difference between their local clocks. In large network this method may not yield the best results, because of internetwork connections and different message routing. Here is shown a system to transmit clock information between processors, that can be used in clock synchronization algorithms to get better readings of other processor's clock's, and also to improve network performance by using the received information from one query to synchronize with more than one processor, instead of having to query al processors in the network.

## 1 Introduction

In distributed systems, the availability of a clock that shown approximately the same time in all processors has many uses, from authentication to distributed task scheduling. This is specially true for large networks where communication between processors tends to be slow. If processors can agree to global time, many synchronization tasks that need network communication may run faster because they won't create idle processors that are waiting for messages.

Large network are normally constructed from smaller local networks that are interconnected by one or more links between them. These links tend to become a bottleneck in the system, and messages sent in the local network tend to have less delay than inter-networked messages. This paper addresses

this problem in the scope of clock synchronization.

The basic idea is to pass clock information between processors, so that a processor may retrieve a better reading of another processor's clock this way than with direct communication. After a processors gets many clock reading from the network it selects the best reading that can be achieved and it applies it in the clock synchronization algorithm.

This use of information gossip is also helpful in cases where network faults may appear, and the network becomes partitioned. If at least one processor of each partition has complete information about the other partition then the clock synchronization achieved will be valid in both networks, although the achieved accuracy may be less than in the connected case.

## 1.1 Relations to Prior Work

There is much work on clock synchronization. This work is related to most of them because it focuses on the reading algorithm instead of the clock synchronizing algorithm. Following is a short list of the most relevant works.

Works in which the algorithm already has a set of views (or clock readings), like the works by Attiya et. al. [1] and Patt-Shamir and Rajsbaum [2] are not related to this work because they don't meddle in how the network achieves these clocks, although this work can be applied to get the clock readings for these algorithms.

Models on which the algorithm is based on remote clock readings, like the Network Time Protocol [3] are closely related to this work because of the use that they can make of this algorithm to improve reading accuracies. This is specially true for network that are strongly interconnected, where there are many paths from one processor to the ohter.

A closely related work to this one the one done by Barak et. al. [4]. In this work, every run of the clock synchronization algorithm requires the processor to retrieve clock readings from all other processors in the system. By passing information, the run time of this algorithm can be reduced by a long range (not that then the algorithm becomes less secure, but a random selection of clock readings reduces the risk. But this is out of the scope of the current work.).

# 2 Algorithm

This section presents the algorithm used to achieve the desired distribution of information in the system. This algorithm is an addition to the clock synchronization algorithm that is in charge of the synchronization.

Each processor in the system has to know he number of processors that constitute the complete network, at each run. Note that this number may vary with time, the only requirement is it's knowledge. For each processor $P_i$ in the system, the following information is saved in the local processor:

$d_i$ - Distance of $P_i$'s clock from the local clock.
$a_i$ - Accuracy of the reading of $P_i$'s clock

This information is saved in a local array denoted *Readings*. This array is initialized with $d_i = 0$ and $a_i = \text{inf}$ for all $i$. We denote $Readings_i$ to be the *Readings* vector of processor $i$, $Readings_i.d_j$ the distance to processor $j$ from processor $i$ in the Readings array, and $Readings_i.a_j$ the accuracy in the reading of processor $j$ for processor $i$.

The clock reading algorithm *Read(j)* which reads the clock of processor $j$ is as follows:

1. Processor $P_i$ sends time request to processor $P_j$; sets S = Local Time.
2. Processor $P_j$ returns its *Readings* vector, and it's local clock $C$.
3. Upon receiving the return message with Readings vector $Readings_j$ and clock C, $P_i$ does the following:
   (a) R = Local Time.
   (b) $d_j = C - \frac{R+S}{2}$
   (c) $a_j = \frac{R-S}{2}$
   (d) if $a_j < Readings_i.a_j$ then $Readings_i.a_j = a_j$, $Readings_i.d_j = d_j$
   (e) else $a_j = Readings_i.a_j$
   (f) for all processors $k \neq j$ do
   - if $a_j + Readings_j.a_k < Readings_i.a_k$
     - $Readings_i.a_k = a_j + Readings_j.a_k$

- $Readings_i.d_k = d_j + Readings_j.d_k$

The algorithm works as follows: Processor $P_i$ asks processor $P_j$ for it's clock. $P_j$ sends it's clock and in addition, sends it's array of readings. Then $P_i$ calculates the the distance and the accuracy that can be achieved from this message. The most important thing of the algorithm is accuracy, therefore it never lowers it. Then it goes on and tries to get a better accuracy for all other processors in the system. This is done by calculating the accuracy of the reading that goes through $P_j$, that is, the reading of $P_j$ with it's accuracy, plus our reading of $P_j$'s clock and it's accuracy. If this is better than the local value, it is saved.

The value calculated as $P_j$'s accuracy is estimated as follows. $P_i$ estimates that at time $\frac{R+S}{2}$, $P_j$'s clock was $C$. If the network is symmetric and there are both clocks tick at the same time, then the estimation would be accurate. Otherwise, $P_j$ answered in any time between $S$ and $R$, so the offset cannot be more than $\frac{R-S}{2}$.

# 3 Analysis

The idea of the algorithm is to improve the accuracy of the clock readings of other processors. Therefore an analysis of the probability that an improvement takes place must be done. The problem is that the round trip delay of a message (the value from which the accuracy is taken) is not known, and in most systems, it cannot be described easily by mathematical means.

The intuition that started this paper stated that the probability of delay is not directly proportional to network distance (this refers to the number of sub-networks the message travels on until it reaches it's destination), but increases in a larger than linear way with each new sub-network it must pass. An analysis based on this supposition is not made here, because it may prove to complex for the need to prove the algorithm's usefulness.

For the analysis, suppose that messages in the network (this is the complete network that connects all computers in the system) has an upper bound **U** and a lower bound **L** on message delay. For processors $P_i$ and $P_j$, denote the message delay by $d_{i,j}$. Because the round trip delay of the message is two times a one-way delay, we treat the round trip delay the same way as a one-way delay. The message delay is taken to be a random variable that is

distributed uniformly over al possible values (that is, can take any value in $[L, U]$).

Suppose processors $P_i$ and $P_j$ have clocks synchronized with accuracy $d_{i,j}$and processors $P_j$ and $P_k$ are synchronized with accuracy $d_{j,k}$. The algorithm helps only if the sum of these accuracies (which are the network delays) is less than the message delay from $P_i$ to $P_k$. More formally, denote by $P(acc)$ the probability that an increase in the accuracy occurs, then:

$$P(acc) = P(d_{i,j} + dj, k \leq di, k)$$

The probability shown above is independent of the message bounds, therefore from this point on, $L = 0$ and $U = 1$.

$$\begin{aligned} P(acc) &= \int_0^1 P(d_{i,j} + dj, k \leq x \mid d_{i,k} = x)dx \\ &= \int_0^1 P(d_{i,j} + d_{j,k} \leq x) \cdot P(d_{i,k} = x)dx \end{aligned}$$

The inequality is true because message delays are independent. The probability is calculated in two parts. The probabilities are calculated as uniform random variables, as stated above.

$$\begin{aligned} P(d_{i,j} + d_{j,k} \leq x) &= \int_0^x P(d_{i,j} = y) \cdot P(d_{j,k} \leq x - y)dy \\ &= \int_0^x x - y \; dy \\ &= xy - \frac{y^2}{2}\Big|_{y=0}^{y=x} \\ &= \frac{x^2}{2} \end{aligned} \tag{1}$$

To finish things up:

$$\begin{aligned} P(acc) &= \int_0^1 \frac{x^2}{2} \, dx \\ &= \frac{x^3}{6}\Big|_{x=0}^{x=1} \\ &= \frac{1}{6} \end{aligned}$$

As can be seen, the probability of increase in the the accuracy is large, which means that the use of the algorithm is worthwhile. A solution to the probability for the case where the information is passed over more than one processor is outside the possibilities of the document.

# 4 Discussion

The given algorithm achieves better clock synchronization were direct communication to all processors are not equal. It uses better connections between close processors to achieve high accuracy between them and then uses this to distribute the synchronization on the whole system. It remains an open question whether the use of this algorithm really increases the accuracy of the system, or reduces it because of the size of the messages that are passed on the network.

An interesting use for this algorithm is to get the clock readings for the protocol proposed by Barak et. al. [4]. There, all processors are queried each time the synchronization protocol is run. Using the proposed algorithm, the number of queries could be diminished by a large number. The problem here is that the algorithm becomes less secure, because there is no knowledge on which processors are occupied by the *attacker* named in their paper. It may prove interesting to find that this algorithm may improve their protocol without reducing the security of their algorithm too much.

# References


[1] H. Attya, A. Herzberg, S. Rajsbaum. Optimal Clock Synchronization Under Differend Delay Assumptions. October 25, 1994.

[2] B. Patt-Shamir, S. Rajsbaum. A Theory of Clock Synchronization. *26th Symp. on Theory of Computing.* May 1994.

[3] D. L. Mills. Internet Time Synchronization: the Network Time Protocol. October 1991.

[4] B. Barak, S. Halevi, A. Herzberg, D. Naor. Clock Synchronization with Faults and Recoveries. *PODC 200, Portland, Oregon.*