Clock Synchronization with Deterministic Accuracy Guarantee

Ryo Sugihara and Rajesh K. Gupta

Computer Science and Engineering Department,
University of California, San Diego, California, USA
{ryo,rgupta}@ucsd.edu

Abstract. Accuracy is one of the most important performance metrics in clock synchronization. While state-of-the-art synchronization protocols achieve μsec-order average accuracy, they usually do not focus on the worst case accuracy and do not have any deterministic guarantees. This lack of accuracy guarantee makes it hard for sensor networks to be incorporated into larger systems that require more reliability than e.g., typical environmental monitoring applications do. In this paper, we present a clock synchronization algorithm with deterministic accuracy guarantee. A key observation is that the variability of oscillation frequency is much smaller in a single crystal than between different crystals. Our algorithm leverages this to achieve much tighter accuracy guarantee compared to the interval-based synchronization methods mostly proposed in the literature of distributed systems. We designed an algorithm to solve a geometric problem involving tangents to convex polygons, and implemented that in TinyOS. Experimental results show the deterministic error bound less than 9.2 clock ticks (280 μsec) on average at the first hop, which is close to the simulation results. Further, by a combination with previously proposed synchronization algorithms, it achieves the estimation error of 1.54 ticks at 10 hop distance, which is more than 40% better than FTSP, while giving deterministic error bounds.

1 Introduction

Clock synchronization is a fundamental service that is required in many applications in sensor networks as well as in distributed systems in general. It is of more importance when sensor networks extend beyond research-oriented systems and get incorporated into industrial systems often referred to as cyber-physical systems (CPS). These systems are often mission-critical, and thus providing a performance guarantee is as important as yielding a good average performance. In the context of clock synchronization, performance guarantee corresponds to guarantees on accuracy.

Accuracy guarantee in clock synchronization, especially deterministic one, is important in sensor networks in several situations including sensor fusion, coordinated actions, and hard-realtime applications, as discussed in [7]. We add security applications to the list. For example in secure localization (e.g., [8]), the

P.J. Marrón and K. Whitehouse (Eds.): EWSN 2011, LNCS 6567, pp. 130–146, 2011.

accuracy of clock synchronization directly affects the reliability of the location estimation and thus the security level that the application can provide.

Recent research on clock synchronization in wireless sensor networks has been mostly focused on improving the average case accuracy rather than the worst case. Although state-of-the-art techniques enable impressive average performance of few microseconds error [14,15,21], they either do not have any guarantees or only have probabilistic guarantees on the worst case accuracy.

In this paper, we propose a novel clock synchronization algorithm that gives a deterministic accuracy guarantee characterized by upper and lower limits on the current time. Although the basic idea is analogous to classical interval-based clock synchronization [6,7,9,13,16,18,20], there are several key differences that enable our algorithm to achieve more correct and tighter bounds. One of the differences is the bounded drift fluctuation, which is given in the data sheet of crystal oscillators and we also experimentally verified by ourselves. The algorithm does not require or construct a fixed network topology and works completely in a distributed manner. It is also efficient by packing the information for multiple receivers into a single packet thus leveraging the broadcasting nature of wireless communication.

Our contribution is a novel clock synchronization algorithm that

- Gives deterministic accuracy guarantee without simplifying assumptions,
- Is fully distributed and does not require any a priori knowledge on the network topology or the topology being stationary, and
- Achieves good clock estimation when combined with other algorithms.

We implement the algorithm in TinyOS for testbed experiments and compare the results with simulation and FTSP.

The rest of the paper is organized as follows. In Section 2, we present the system model as well as the experimental results on temperature vs. clock frequency. We present main ideas of the synchronization algorithm in Section 3 and the details in Section 4. In Section 5, we discuss some of the issues that arise in implementing the algorithm in TinyOS. Section 6 presents the evaluation results from both simulation and testbed experiments. We overview related work in Section 7 and Section 8 concludes the paper.

2 System Model

We first define the clock model and describe the assumptions. Then we validate these assumptions through preliminary experiments on the frequency vs. temperature characteristics of crystals.

2.1 Clocks

We refer to the clock reading at node v as *localtime* at v and denote as s^v. There are one or more nodes that have access to accurate time e.g., through GPS. These nodes are called *roots* and their time is called *globaltime* t, which is

also called "wall-clock time" or "physical time" in the literature. All other nodes are just referred to as *nodes*. Node v's clock has a *clock drift* $h_v(s^v)$, which is defined as the amount of increase in globaltime when the localtime is increased by unit amount. For each node, we define *clock function* f_v to give corresponding globaltime for each localtime as follows:

$$f_v(s^v) = \int_0^{s^v} h_v(\tau)d\tau + \delta_v,$$

where δ_v is a constant called *clock offset*. As a notational convention, we assume globaltime t_i corresponds to localtime s_i^v; i.e., $t_i = f_v(s_i^v)$.

The drift consists of *drift offset* $\overline{h_v}$ and *drift fluctuation* $\alpha_v(s^v)$, where the former is a constant and the latter is a time-varying function:

$$h_v(s^v) = \overline{h_v} + \alpha_v(s^v).$$

While these are not known in advance, we assume they satisfy the following two properties:

- Bounded drift offset: $1 - \eta \leq \overline{h_v} \leq 1 + \eta$,
- Bounded drift fluctuation: $|\alpha_v(s^v)| \leq \xi$,

where η and ξ are given constants. The values of η and ξ depend on the type of crystal oscillator and usually specified in or can be calculated from the data sheet. In normal temperature range, ξ is much smaller than η. Note that we do not assume any statistical properties for $\alpha_v(s^v)$ except that it is bounded by ξ. Also note that η and ξ are different from "drift bound" (usually denoted by ρ) that often appears in the literature. Since $1 - \rho \leq h_v(s^v) \leq 1 + \rho$ by definition, we can consider that $\rho = \eta + \xi$. This is one of the key differences between the proposed algorithm and so-called interval-based methods, which we will discuss later in Section 3.4.

The objective of a synchronization algorithm is for each node to obtain a mapping from its localtime to the globaltime. We specifically focus on giving a deterministic guarantee on accuracy: for any localtime, each node must be able to tell the interval that the globaltime is contained within.

2.2 Crystal Oscillator

The assumptions of bounded drift offset and bounded drift fluctuation are not common and also very important for our algorithm. For these reasons, we have measured these values in the actual nodes to assess how reasonable these assumptions are. Since temperature is the primary cause that affects the clock frequency [22], we measure the frequency under different temperatures.

Crossbow Telos (Rev. B) nodes use Citizen CMR200T for 32.768kHz crystal oscillator [3]. This is a tuning fork crystal unit [1] and is known to have a quadratic relation between frequency and temperature. The relation is expressed by $f = f_0(1 + \beta(T - T_0)^2)$, where f_0 is the nominal frequency, T_0 is the reference

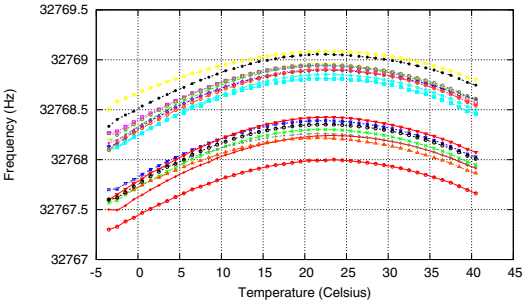


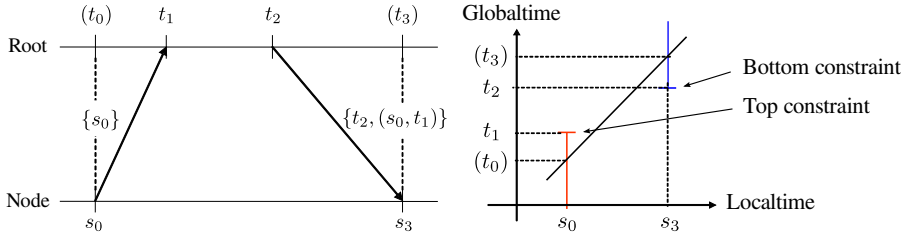
Fig. 1. Temperature vs. frequency of 32.768 kHz crystal oscillator: For 19 nodes

temperature, and β is a constant called "temperature coefficient." In practice the actual frequency f_0' at T_0 is different from f_0 by a small amount and the bound on frequency tolerance $(f_0' - f_0)/f_0$ is usually specified in the data sheet. For CMR200T, $f_0 = 32768$ Hz, $\beta = -0.034 \pm 0.006$ ppm/℃^{2} (ppm: parts-per-million), and the frequency tolerance is ±20 ppm [1]. The characteristic differs by types and also by parameters of the cut.

Figure 1 shows the relation of temperature and frequency measured at 19 Telos B nodes. Clock frequency is measured by counting the number of ticks precisely in 10 seconds, which is obtained from PPS (Pulse-per-Second) output of a GPS module (Garmin 18x LVC). This GPS module guarantees that the PPS signal is aligned to the start of each GPS second within 1 μsec [2]. We gradually change the environment temperature and associate the measured clock frequency with the temperature measured simultaneously by the onboard sensor.

All nodes exhibit curves peaked between 20 to 25 ℃ and their shapes are close to the one specified in the data sheet (not shown). As for the offset, the peaks are above the nominal frequency (32768 Hz) in 18 out of 19 nodes. This is likely to be due to the mismatch between ideal and actual load capacitance. Specifically, CMR200T requires load capacitance of 12.5 pF, whereas the MPU (MSP430F1611) has internal 12 pF fixed capacitance per pin [5] and they are added serially for two pins, resulting in 6 pF capacitance in total. Smaller load capacitance leads to higher oscillation frequency [4] and apparently, it is in accordance with the results. Unfortunately, since there are other sources of parasitic capacitance e.g., from PCB traces, we do not have a guarantee as strong as that for the frequency tolerance in the crystal's datasheet. However, based on the observation, the curve still satisfies the frequency tolerance specification and we assume the frequency is only shifted by unknown small constant amount.

In summary, as we expected, we have observed larger variability among different crystals than in a single crystal at different temperatures. To accommodate the shift of peak frequency, we set the nominal frequency $f_0 = 32768.5$ [Hz]. In the later experiments, we assume the temperature range is 10 to 35 ℃. Then we can guarantee the frequency range for a single crystal is within 10 ppm (0.33 Hz in 32 kHz crystal) from both the specification and the measurement results. Since the peak frequency is within 20 ppm from the nominal frequency, we set



(a) Message exchange: t_0, t_3 are not known. (b) Constraints on clock function

Fig. 2. Basic idea for synchronization between a root and a node

the drift offset bound η to 25 ppm and the fluctuation bound ξ to 5 ppm to cover the whole range. If the temperature range is broader or unknown, these parameters must be chosen accordingly and conservatively to assure the correctness of the accuracy guarantee.

3 Synchronization with Accuracy Guarantee

In this section we describe the overall idea of our synchronization algorithm. For clarity we first assume constant drift (i.e., no drift fluctuation: $\xi = 0$) and explain the algorithm for synchronization between a root and a node. Then we extend it for synchronization between two nodes to enable network-wide synchronization, and also describe how we can take into account the drift fluctuation. Finally we discuss the differences between our algorithm and interval-based methods.

3.1 Main Idea

Our synchronization algorithm is based on a simple causality principle that a message is received only after it is sent. From sender and receiver timestamps, each node obtains a set of constraints that its clock function must satisfy.

Figure 2(a) shows the message exchange between a root and a node. The node sends a message at localtime s_0 and the root receives it at globaltime t_1. Then the root sends back a message at t_2 with the information on previously received message (s_0, t_1) as well as with the timestamp t_2. Upon receiving the message, the node can do the following calculations. For the first message, since the receiving time is later than the sending time, $t_0 \leq t_1$, though we cannot know t_0. Using $f(s_0) = t_0$, where f is the clock function for the node, we have $f(s_0) \leq t_1$. Similarly for the second message, we obtain $f(s_3) \geq t_2$.

The constraints that give upper or lower bounds for a clock function are called *top constraints* and *bottom constraints*, respectively (Fig. 2(b)). A constraint C_i is expressed by $(s_i, l_i, type_i)$, where s_i is localtime, l_i is called the *value* of C_i, and $type_i$ is either "top" or "bottom."

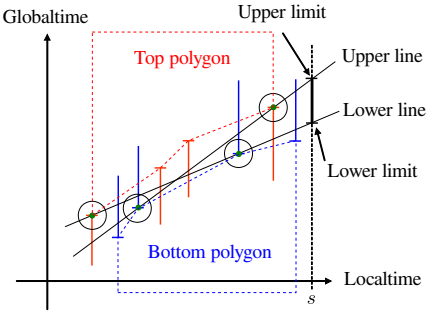


Fig. 3. Upper/lower limit given by upper-/lowermost line satisfying all constraints

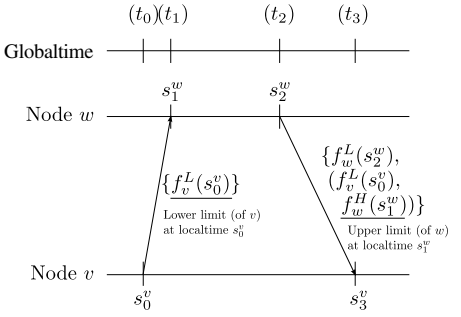


Fig. 4. Node-node message exchange: $t_0, ..., t_3$ are not known

After several messages are exchanged, each node has a set of top constraints and bottom constraints (Fig. 3). When we ignore the drift fluctuation (this is relaxed later in the section), f is a linear function that satisfies all the constraints. Of all such linear functions, we can determine the ones that give the maximum and minimum globaltime at given localtime s. We call these *upper line* and *lower line*, which are collectively called *limiting lines*. The maximum and minimum globaltime are called *upper limit* and *lower limit* at localtime s, respectively. Further, we call a constraint a *support* or a *support constraint* when it determines the upper or lower line.

The problem of finding limiting lines can be viewed in a more geometric way by considering two polygonal objects for each of the sets of top and bottom constraints. The clock function is a long bar and the upper and lower limits correspond to the range of motion when it is inserted between two polygons.

3.2 Network-Wide Synchronization

The algorithm for synchronization between a root and a node can be extended for network-wide synchronization. We describe the case for two nodes where neither of them is a root. We use f_v^H and f_v^L to denote the upper and lower limits at node v. For any localtime s^v, they satisfy $f_v^L(s^v) \leq f_v(s^v) \leq f_v^H(s^v)$.

Figure 4 shows the message exchange for synchronization between two nodes. Suppose node v initiates the message exchange. Different from the root vs. node case, at localtime s_0^v, v sends the lower limit $f_v^L(s_0^v)$ instead of s_0^v itself, which node v remembers for later use. Node w records the localtime s_1^w when it receives the message, but it remembers the upper limit $f_w^H(s_1^w)$ instead. Then in the response message, node w puts the pair $(f_v^L(s_0^v), f_w^H(s_1^w))$ as well as the lower limit $f_w^L(s_2^w)$, just as node v did. This pair of (lower limit at sent time, upper limit at received time) is called *SyncInfo* for the original sender.

After this message exchange, node v obtains the following two constraints:

$$\text{top: } f_v(s_0^v) = t_0 < t_1 = f_w(s_1^w) \leq f_w^H(s_1^w) \quad (1)$$

$$\text{bottom: } f_v(s_3^v) = t_3 > t_2 = f_w(s_2^w) \geq f_w^L(s_2^w) \quad (2)$$

Note that $t_0, ..., t_3, f_w(s_1^w), f_w(s_2^w)$ are all unknown.

As a side-effect, node w also obtains a bottom constraint $f_w(s_1^w) > f_v^L(s_0^v)$ from the first message. This is a preferable property especially for wireless environment where every communication is essentially a broadcast. For network-wide synchronization, multiple receivers within the communication range of a sender can obtain a bottom constraint. We can also embed multiple SyncInfo in one message so that multiple receivers can obtain top constraints simultaneously.

3.3 Compensation for Drift Fluctuation

So far we have assumed that clock drift is constant. However, in practice, clock drift fluctuates over time with external causes, mostly due to temperature changes. Here we extend the algorithm for the case with drift fluctuations.

The idea is to compensate each of the constraints for the effect of drift fluctuation after the constraint is obtained. For constraint $C_i = (s_i, l_i, type_i)$, we define *compensated constraint* $\tilde{C}_i(s) = (s_i, \tilde{l}_i(s), type_i)$ at localtime s, where compensated value $\tilde{l}_i(s)$ is defined as follows:

$$\tilde{l}_i(s) = \begin{cases} l_i + \xi(s - s_i) & \text{if } type_i = \text{“top”} \\ l_i - \xi(s - s_i) & \text{if } type_i = \text{“bottom”} \end{cases}$$

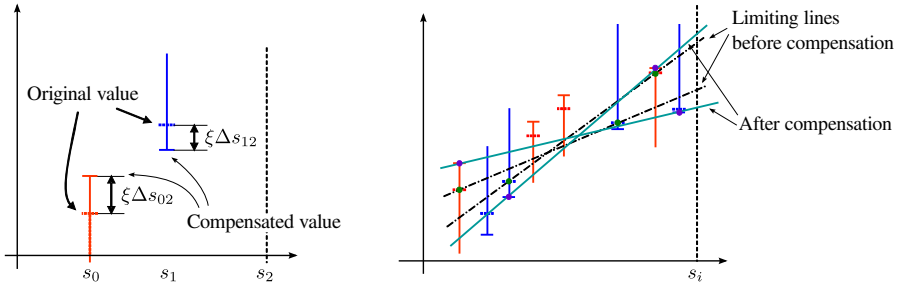
Then we have the following lemma:

Lemma 1. *Given $f(s)$ that satisfies all the constraints and $f(s_1) = t_1$, linear function $g(s) = \overline{h}s + \delta$ with $g(s_1) = t_1$ satisfies all the compensated constraints, where $\overline{h}, \delta$ is the drift offset and clock offset, respectively.*

Proof is omitted from this version. Then we have the following theorem:

Theorem 1. *$\forall s. g^L(s) \leq f^L(s) \leq f^H(s) \leq g^H(s)$, where $g^L(s), g^H(s)$ are the lower and upper limits at localtime s calculated for linear clock function $g(s)$ with the slope in $[1 - \eta, 1 + \eta]$ that satisfies all the compensated constraints.*

Figure 5 explains compensated constraints. As Fig. 5(a) shows, at localtime s_2, the value of the top constraint at s_0 is increased by $\xi \Delta s_{02}$, and that of the bottom constraint at s_1 is decreased by $\xi \Delta s_{12}$. Then, as shown in Fig. 5(b), we change the value for all constraints in the same way and the limiting lines are determined for these compensated constraints. Since the compensated constraints are “looser” than the original ones, according to Theorem 1, the new interval of upper and lower limits contains the one calculated without compensation. In this way, after compensation, we only need to solve the same problem of finding limiting lines.



(a) Compensation at localtime s_2 (b) Limiting lines at localtime s_i

Fig. 5. Compensated constraints

3.4 Comparison with Interval-Based Methods

Our algorithm shares the idea with the clock synchronization algorithms that provide an interval that the globaltime is contained. These algorithms, often called "interval-based algorithms," are proposed mostly in the literature of distributed systems before sensor networks emerged, but there are some papers in the contexts of mobile ad-hoc networks [18] and sensor networks [7] as well. Here we compare the proposed algorithm with these works.

While it is often claimed that an algorithm guarantees the accuracy, the correctness may not hold in the context of sensor networks. For example, some of the interval-based algorithms just ignore transmission delay (e.g., [7,16]). This is not an appropriate assumption when the transmission delay is dominant in the resulting accuracy. More often clock drift is assumed to be constant [6,9,23]. This is not appropriate as well for sensor networks that are often deployed outside and exposed to drastic temperature changes. Our algorithm does not make these assumptions and only uses the bounded drift offset and bounded drift fluctuation assumptions, both of which are guaranteed to be correct and derived from the data sheet of crystal oscillator.

Meanwhile, the common underlying idea for interval-based synchronization methods is to bound the actual length of any time interval by using most pessimistic value of clock drift. For example, when clock drift bound is ρ, we can guarantee that the actual length of localtime interval Δs is bounded by $[(1-\rho)\Delta s, (1+\rho)\Delta s]$. In our settings, we can emulate this by setting drift offset bound $\eta = 0$ and drift fluctuation bound $\xi = \rho$. This is because their assumption is precisely equivalent to assuming that each crystal can fluctuate in the whole range of $[1-\rho, 1+\rho]$. In the experiments, we will use this to compare our algorithm with the interval-based methods.

4 Algorithm Details

In this section we describe the algorithm in more detail. RECEIVE and SEND are the procedures called upon message reception and transmission, respectively.

Upon receiving a new message, constraints are added using AddConstraints and Convexify, and limiting lines are calculated using UpdateUpperLine/UpdateLowerLine. As we see later, all these procedures can be done in the time logarithmic to the number of constraints stored in a node.

4.1 Communications

A root broadcasts a message periodically. The period is to be determined based on application requirements and energy availability. In the experiments we use uniformly random period in $18 - 22$ seconds.

Upon receiving a new message, a node updates upper and lower lines. This is necessary because, due to the compensation of drift fluctuation, the limiting lines will change even without any changes in the sets of constraints.

After that, the node adds a bottom constraint based on the received time and the lower limit that the message has. The same information is saved as SyncInfo, which will be sent back to the sender. Then the node scans through all SyncInfo that the message carries. If there is one for this node, it adds a top constraint based on that.

Shortly after receiving, a node sends a message when one of the new constraints has become a support. We use this "pulse-like" propagation pattern based on the analysis in [14]. Upon sending a message, the node first updates the lower line to calculate the current lower limit.

4.2 Computations

Add Constraint. After adding a new constraint, we call Convexify to eliminate any non-convex constraints, since they will never become supports, as seen from Fig. 3. Then, if the new constraint violates the current limiting line, we update the limiting line.

Convexify. Convexify is the procedure for making the set of constraints form a convex polygon by eliminating non-convex constraints. Finding non-convex constraints is easily done by calculating the turning direction of two vectors (typically by using the external product) made by three points. This is similar to what Graham scan [10] does for calculating the convex hull of points. However, since in our case the points are already sorted and we know the set is convex without the new point, we can use binary search to find a convex point instead of linearly scanning the points. This makes the running time $O(\log n)$, when n is the number of top/bottom constraints stored in the node.

One thing to note is that, in case of top constraints, a newly added constraint may not be the latest constraint in terms of localtime. Therefore, we may need to run the above binary search for both directions from the inserted point, though it does not affect the order of the computation time.

```
procedure Receive(f_w^L, {S})   ▷ at localtime s_recv^v; {S}: Set of SyncInfo in received message
    UpdateUpperLine
    UpdateLowerLine
    AddConstraint(s_recv^v, f_w^L, bottom)
    f_v^H ← CalcUpperLimit
    if f_v^H ≠ ∞ then SaveSyncInfo(src = w, f_w^L, f_v^H)
    for S_i ∈ {S} do
        if S_i is for this node then
            s_send^v ← RetrieveSendTime(S_i)
            AddConstraint(s_send^v, value(S_i), top)
    if new constraint became a support then Send

procedure Send                                          ▷ at localtime s_send^v
    UpdateLowerLine
    f_v^L ← CalcLowerLimit(s_send^v)
    StoreSendTime(s_send^v, f_v^L)
    SendMessage(f_v^L, {S})                             ▷ {S}: Set of SyncInfo

procedure AddConstraint(C_new)
    if C_new is a top constraint then
        C_TOP ← C_TOP ∪ C_new          ▷ C_TOP: set of top constraints, sorted by localtime
        Convexify(C_new)
        if C_new ∈ C_TOP ∧ C_new violates upper line then UpdateUpperLine
    else                                                ▷ C_new is Bottom
        C_BOT ← C_BOT ∪ C_new          ▷ C_BOT: set of bottom constraints, sorted by localtime
        Convexify(C_new)
        if C_new ∈ C_BOT ∧ C_new violates lower line then UpdateLowerLine

procedure Convexify(C_new)
    Find leftmost C_i s.t. {C_i, ..., C_new} is non-convex
    Remove non-convex constraints between C_i, C_new
    Find rightmost C_j s.t. {C_new, ..., C_j} is non-convex
    Remove non-convex constraints between C_new, C_j

procedure UpdateUpperLine
    l ← FindTangent
    if Slope(l) > 1 + η then l ← FindTopSupport(1 + η)

procedure UpdateLowerLine
    l ← FindTangent
    if Slope(l) < 1 − η then l ← FindBottomSupport(1 − η)
```

Update Limiting Lines. The core of the procedures for updating upper and lower lines is FindTangent. In FindTangent we find a tangent of two convex polygons. Note that polygons are convex whenever FindTangent is called. A naive method mentioned in [6] is to try all possible pairs of vertices and check if the line intersects with the polygons. This takes $O(n^2)$ time in the worst case, but can be improved to $O(\log^2 n)$ by using nested binary search. Further, by using more sophisticated algorithms, it is improved to $O(\log n)$ time [11,17].

When the slope of the calculated limiting line is out of range of $[1-\eta, 1+\eta]$, we can replace the line with the slope $(1+\eta)$ (for upper line) or $(1-\eta)$ (for lower line). In this case the line is supported by one point instead of two. Searching the support is done efficiently by calculating the slope of each edge of the polygon, and since it is convex, the slope is monotonically increasing (for top polygon) or decreasing (for bottom polygon). By using binary search, it takes $O(\log n)$ time.

5 Implementation

We have implemented the proposed synchronization algorithm in TinyOS 2.1.1. Besides dealing with limitations in TinyOS programming, we needed to address several fundamental issues specific to our synchronization algorithm.

5.1 Delay Compensation

Reducing the transmission delay between sender and receiver is the key to achieving precise synchronization. This also applies to our algorithm, as the minimum possible gap between upper and lower limits is at least twice as big as the minimum transmission delay (proof omitted).

To minimize the transmission delay, we use MAC-layer timestamping, which is to put a timestamp when the packet is actually transmitted over radio. In case of CC2420 radio chip, the timestamp for a packet is obtained when SFD (start frame delimiter) is captured. Since this is same for both sender and receiver, two timestamps are expected to be obtained at very close time points.

A problem in MAC-layer timestamping is the delay after a node calculates the lower limit until the packet is transmitted. A receiver can get a tighter bottom constraint if the lower limit is calculated at the time when the packet is transmitted. However, it is not feasible to recalculate it at MAC layer.

A solution for this problem is to reconstruct the lower limit at the receiver side. At localtime s_1, the sender calculates the lower limit $f^L(s_1)$, puts it in the packet, and issues the send command. In the timestamp field, which will be rewritten at the MAC-layer, the sender put s_1, the localtime when it calculated the lower limit. At the MAC-layer this field is rewritten with the difference with value in the field and the current timestamp s_2. When the packet is received, the receiver carries $f^L(s_1)$ and $\Delta s_{12} = s_2 - s_1$. Then it calculates the delay-compensated lower bound $\tilde{f}^L(s_2)$ as follows:

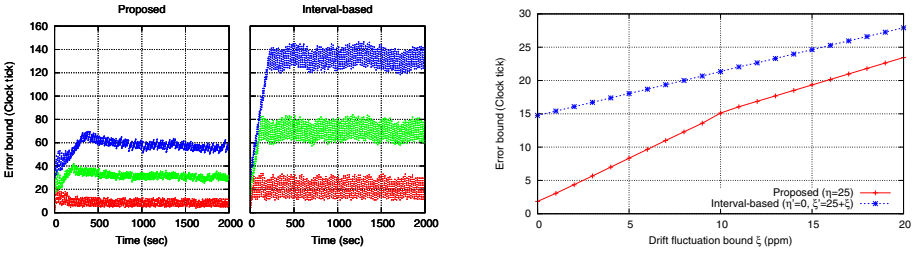
$$\tilde{f}^L(s_2) = f^L(s_1) + (1 - 3\eta - \xi)\Delta s_{12}.$$

This is based on the following theorem. Proof is omitted from this version.

Theorem 2. *$f^L(s_2) \geq f^L(s_1) + (1 - 3\eta - \xi)\Delta s_{12}$, where $f^L(s_2)$ is actual lower limit at localtime s_2.*

5.2 Quantization Error

There are several issues related to quantization error that affect the correctness of the algorithm. One is about the order of timestamping. We strictly require that the timestamp at the sender is taken before that at the receiver. However, when we consider integer timestamps, this is not always satisfied. This order violation will not happen if the receiver timestamp is always taken more than one clock tick after the sender's one, but unfortunately in our case, the difference is much smaller than one tick. To avoid this situation, we add one clock tick to the receiver's timestamp.



(a) Proposed vs. Interval-based method (b) Average error bound at first hop

Fig. 6. Comparison between proposed and interval-based methods

The other is about compensated constraints. Even though the original value of each constraint is an integer, it will become a noninteger when we calculate the compensated value. For not to violate the compensated constraints, we need to be conservative on quantization. Specifically, we round up the value for a top constraint and round down the value for a bottom constraint. Because of this, there are some cases that message transmission loops in two nodes without receiving any new messages. To avoid this, a node does not send a message if it has sent the previous one within a short (~ 1 sec) period.

6 Evaluations

In this section we evaluate the proposed synchronization algorithm by both simulation experiment and testbed experiment. For the performance metric, we use the error bound, which is calculated from the upper and lower limits by $(f^H(s) - f^L(s))/2$. This is the minimum error bound, which is achieved by using the average of upper and lower limits as the estimate.

6.1 Comparison with Interval-Based Method

First we compare the proposed algorithm and interval-based methods by simulation. We use a topology of one root at the end and 10 nodes connected in line. Figure 6(a) compares the error bounds of three nodes (at 1st, 5th, and 10th hop from the root) for both algorithms. For all three nodes, the proposed algorithm achieves much smaller error bounds than the interval-based method. This suggests the information on drift fluctuation bound can help improving the accuracy in clock synchronization.

To see this effect in more quantitative way, we change the drift fluctuation bound ξ and see the error bound. For the proposed method, the drift offset bound η is kept constant at 25ppm, and we emulate an interval-based method by setting $\eta' = 0$ and $\xi' = \eta + \xi$ based on the discussion in Section 3.4. Figure 6(b) shows the results for the first hop. The results show that the proposed method can achieve smaller error bound for all cases and the improvement is bigger for the smaller ξ.

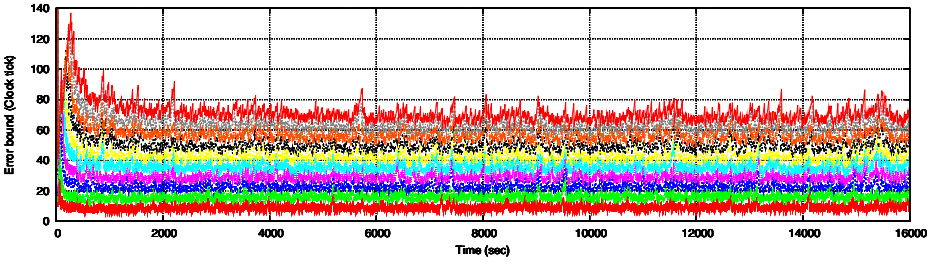
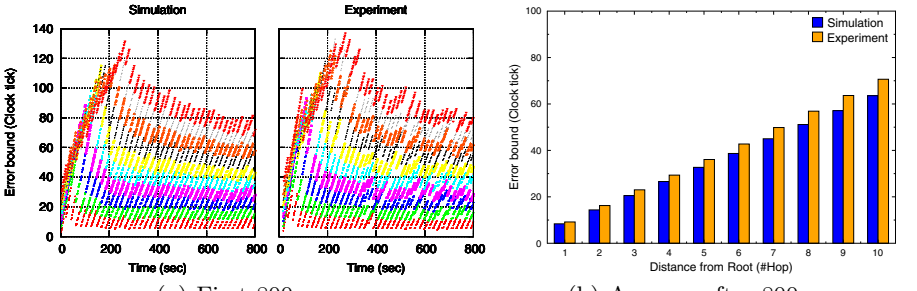


Fig. 7. Testbed experiment: Error bounds of 10 nodes in a line topology



(a) First 800 sec (b) Average after 800 sec

Fig. 8. Comparison between simulation and testbed experiment

6.2 Testbed Experiments

We programmed 11 Telos B nodes; one for root and other 10 for nodes. In the software we made a line topology with a root at one end. The UserINT ports of the root and all nodes are connected to a single trigger source that emits a trigger every two seconds. Upon receiving a trigger, the root node sends the current time and all other nodes send the upper and lower limits at that time to the PC. We use serial communication via USB port for the communication with the PC to avoid congesting the radio channel.

Comparison with Simulation Results. Figure 7 shows the error bound of all 10 nodes (except the root). Figure 8 shows the comparison between measured and simulated results. The change of error bounds closely matches the simulation result both in the transient state at first (Fig. 8(a)) and in the steady state after long time (Fig. 8(b)), though the average error bound in the experiment is a little (9-13%) higher than in the simulation. The error bound at the first hop is 9.2 ticks, which corresponds to 280.0 μsecs at 32768.5Hz clock, and it is roughly proportional to the hop distance from the root.

Improved Clock Estimation and Comparison with FTSP. As an extension we have incorporated the ideas from previous work to improve the clock estimation performance. We slightly change the proposed algorithm and add the current estimate of globaltime in each message, in addition to the lower bound of it.

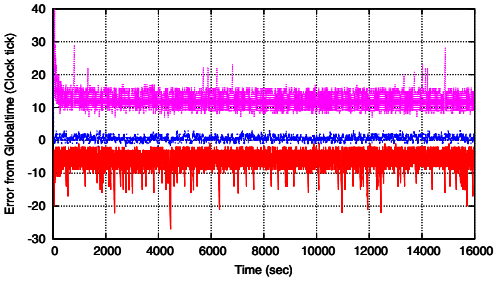


Fig. 9. Upper/lower limits and estimate at the first hop

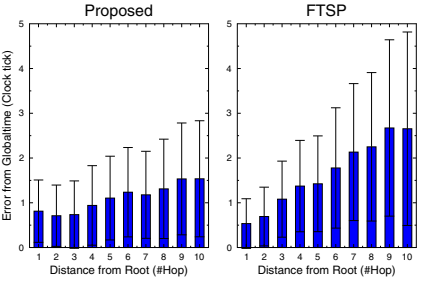


Fig. 10. Comparison with FTSP: Average estimation error

For estimation, we use a common method used in e.g.,[12,14]: Each node considers the time lapse between receiving and sending, makes a table of (localtime, globaltime) pair, and uses linear regression to obtain an estimate.

Figure 9 shows upper/lower limits and estimate at the first hop in the 10-node line topology. The vertical axis is the error from globaltime. Since the upper and lower limits are deterministic bounds, the error for upper limit is always positive and that for lower limit is negative. The estimate distributes around zero. Figure 10 shows the average estimation error for each of 10 hops line topology compared between the proposed algorithm and FTSP. It is observed that the average error for the proposed algorithm is comparable to that for FTSP in the near distance and up to 40% better in the far (At 10th hop, 1.54 vs. 2.65 ticks; 42% better).

7 Related Work

Our synchronization algorithm is most closely related to interval-based methods, as we have discussed in Section 3.4. Marzullo and Owicki [16] studied the synchronization problem where each time server returns an interval that contains the true time and the objective is to minimize the length of interval by exchanging messages among multiple time servers. Blum et al. [7] modified their algorithm for sensor networks and improved the average case performance. Römer proposed another interval-based algorithm tailored for ad hoc networks settings [18]. Schmid and Schossmaier [20] provided bounds under the assumption that several performance parameters such as transmission delay bounds as well as drift bounds are given. In our work, we have made our first attempt to give bounds only from the information readily available in the specifications of the hardware components. Potentially we can improve the results when we have more information about the hardware and the environment.

The use of convex polygons and estimation of clock functions as tangents to them was first proposed by Duda et al. [9] and studied in further detail by Berthaud [6]. In [23], the authors designed optimal and approximate algorithms to keep only the constraints that become supports. All these works assume

constant drift either indefinitely or for a short time. Our algorithm share the same idea for determining the limiting lines, but without constant drift assumptions.

The notion of delay compensation (Section 5.1) has been discussed in [13,20]. We have extended these results under the bounded drift fluctuation assumption and also devised a method suitable for the case of MAC-layer timestamping.

Since MAC-layer timestamping reduces the transmission delay down to few microseconds, clock drift is the main issue for synchronization, since many sensor nodes use normal crystal oscillators with large drift rather than TCXO or OCXO due to energy and cost limitations. FTSP [15] estimates the clock drift and offset with high precision by using linear regression, assuming constant drift for a short period. GTSP [21] focuses on minimizing the local error among the neighboring nodes by adjusting the logical clock rate based on the estimation of clock drift of neighborhood nodes. PulseSync [14] improves the performance at distant hops by introducing a pulse-like propagation pattern. The latter two papers include convergence and optimality results on the performance including accuracy, but they are either asymptotic or probabilistic under the assumption of constant drift and bounded transmission delay. In this paper we focused on the deterministic accuracy guarantee instead, though we have demonstrated that the proposed algorithm can be combined with them to obtain good estimation.

Schmid et al. [19] proposed a method for compensating for the drift change due to temperature change. This is essentially a software implementation of what TCXOs do. Using the onboard temperature sensor, each node makes a table storing the pair of temperature and relative drift. After getting enough entries in the table, a node can estimate the drift with high accuracy and thus can compensate for that without communicating with other nodes. Our algorithm is orthogonal to this. As we have seen in Fig. 6(b), we can achieve tighter error bound for smaller drift fluctuation bound. Therefore, with this or any other techniques that reduce drift fluctuation and provide deterministic guarantee for that, we can improve the accuracy guarantee with the proposed algorithm.

8 Conclusions

We have presented a clock synchronization algorithm that gives deterministic accuracy guarantees. The main idea is to find the upper and lower limits of clock function that satisfies all the constraints obtained using causality relations in communication. While the idea is similar to classical interval-based synchronization methods, we do not make simplifying assumptions such as constant drift or negligible transmission delay to obtain strict guarantees. Still, as demonstrated by the experiments, we achieve tighter guarantees owing to the bounded drift fluctuation assumption, which is confirmed by the hardware specification as well as by the preliminary experiments. We have implemented the algorithm in TinyOS and demonstrated that the accuracy guarantees are close to the simulation results. Furthermore, we extended the algorithm to obtain good estimation and achieved the estimation error up to 40% better than FTSP.

Acknowledgments. This work was supported in part by UCSD/LANL Engineering Institute and by NSF under grant numbers CNS-0932360, CCF-0702792, and SRS-0820034. The first author thanks Eric Yip for his help on the TinyOS implementation. The first author also thanks Muhammad Abdullah Adnan and Thomas Schmid for fruitful discussions.

References

1. CMR200T data sheet, http://www.citizencrystal.com/images/pdf/k-cmr.pdf
2. GPS 18x Technical Specifications, http://www8.garmin.com/manuals/GPS18x_TechnicalSpecifications.pdf
3. TinyOS Hardware Designs: Telos (Rev B) BOM, http://webs.cs.berkeley.edu/tos/hardware/telos/telos-revb-bom-2004-09-21.xls
4. MSP430 32-kHz Crystal Oscillators, Rev. B (2006), http://focus.ti.com/lit/an/slaa322b/slaa322b.pdf
5. MSP430x1xx Family User's Guide, Rev. F (2006), http://focus.ti.com/lit/ug/slau049f/slau049f.pdf
6. Berthaud, J.M.: Time synchronization over networks using convex closures. IEEE/ACM Trans. Networking 8(2), 265–277 (2000)
7. Blum, P., Meier, L., Thiele, L.: Improved interval-based clock synchronization in sensor networks. In: IPSN (2004)
8. Capkun, S., Cagalj, M., Srivastava, M.: Secure localization with hidden and mobile base stations. In: INFOCOM (2006)
9. Duda, A., Harrus, G., Haddad, Y., Bernard, G.: Estimating global time in distributed systems. In: ICDCS (1987), http://scholar.google.com/scholar?hl=en\&btnG=Search\&q=intitle:Estimating+Global+Time+in+Distributed+Systems#0
10. Graham, R.L.: An efficient algorithm for determining the convex hull of a finite planar set. Information Processing Letters 1, 132–133 (1972)
11. Kirkpatrick, D.G., Snoeyink, J.: Computing common tangents without a separating line. In: Sack, J.-R., Akl, S.G., Dehne, F., Santoro, N. (eds.) WADS 1995. LNCS, vol. 955, pp.183–193. Springer, Heidelberg (1995)
12. Kusy, B., Dutta, P., Levis, P., Maroti, M., Ledeczi, A., Culler, D.: Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services. Int. J. Ad Hoc Ubiquitous Comput. 1(4), 239–251 (2006)
13. Lamport, L.: Synchronizing time servers. SRC Research Report 18 (1987)
14. Lenzen, C., Sommer, P., Wattenhofer, R.: Optimal clock synchronization in networks. In: SenSys (2009)
15. Maróti, M., Kusy, B., Simon, G., Lédeczi, A.: The flooding time synchronization protocol. In: SenSys (2004)
16. Marzullo, K., Owicki, S.: Maintaining the time in a distributed system. In: PODC (1983)
17. Overmars, M.H., van Leeuwen, J.: Maintenance of configurations in the plane. J. Comput. Syst. Sci. 23(2), 166–204 (1981)
18. Römer, K.: Time synchronization in ad hoc networks. In: MobiHoc (2001)
19. Schmid, T., Charbiwala, Z., Shea, R., Srivastava, M.B.: Temperature compensated time synchronization. IEEE Embedded Systems Letters 1(2), 37–41 (2009)

20. Schmid, U., Schossmaier, K.: Interval-based clock synchronization. Real-Time Systems 12(2), 173–228 (1997)
21. Sommer, P., Wattenhofer, R.: Gradient clock synchronization in wireless sensor networks. In: IPSN (2009)
22. Vig, J.R.: Introduction to quartz frequency standards. Tech. Rep. SLCET–TR–92–1, Army Research Laboratory (1992), http://www.ieee-uffc.org/frequency_control/teaching.asp?name=vigtoc
23. Yoon, S., Veerarittiphan, C., Sichitiu, M.L.: Tiny-sync: Tight time synchronization for wireless sensor networks. ACM Trans. Sensor Networks 3(2), 8 (2007)