Clock Synchronization for Wireless Networks

Rui Fan⋆   Indraneel Chakraborty†   Nancy Lynch⋆

Computer Science and Artificial Intelligence Laboratory

Massachusetts Institute of Technology

⋆{rfan, lynch}@theory.csail.mit.edu, †indranil@lcs.mit.edu

1   Abstract

Time synchronization is a fundamental service in many wireless applications. While the synchronization problem is well-studied in traditional wired networks, physical constraints of the wireless medium impose a unique set of challenges. We present a novel time synchronization algorithm which is highly energy efficient and failure/recovery-tolerant. Our algorithm allows nodes to synchronize to sources of real time such as GPS when such signals are available, but continues to synchronize nodes to each other, even in the absence of GPS. In addition, the algorithm satisfies a relaxed gradient property, in which the degree of synchronization between nodes varies as a linear function of their distance. Thus, nearby nodes are highly synchronized, which is desirable in many wireless applications.

2   Introduction

Wireless networks are an increasingly important medium for distributed computation. As wireless applications grow more diverse and sophisticated, time synchronization among wireless nodes has emerged as a common requirement of many applications. For example, MAC layer protocols such as TDMA [5] require time synchronization to schedule collision-free broadcast schedules. Time synchronization is essential in sensor networks, which collect data from a physical environment, then tag the data with the time of its occurrence. Time synchronization is also needed in high-level applications to timestamp and order events and signals, and for security purposes. While time synchronization in wired networks is a well-studied problem, the wireless medium presents a unique set of challenges. The primary concern of all wireless applications is energy conservation. A clock synchronization algorithm (*CSA* for short) must carefully regiment its frequency of resynchronization, and avoid flooding. In addition, the algorithm cannot typically rely on a power-hungry source of real time such as GPS. Another characteristic of wireless networks is unexpected and possibly frequent changes in network topology. Thus, a CSA in a wireless medium must continue to function in the face of node failures and recoveries. Lastly, many applications in wireless settings are local in nature. That is, only nearby nodes in the network need to participate in some activity. Thus, a desirable property for a CSA is that it closely synchronizes nodes which are nearby, while possibly allowing faraway nodes to be more loosely synchronized.

In this work, we present a time synchronization algorithm addressing the requirements of a wireless network. Our algorithm is energy-efficient: nodes perform at most one local (1-hop) broadcast per synchronization round. It tolerates dynamic network behavior: the algorithm continues to function when there are arbitrary node failures and joins. A novel feature of our algorithm is that it performs both *internal synchronization*, i.e. synchronizing nodes with each other, and *external synchronization*, i.e. synchronizing nodes with real time. The algorithm allows incorporating a GPS source of real time when such signals are available, but continues to synchronize nodes with each other even in the absence of GPS. Thus, for example, a sensor node can use our algorithm both for timestamping data with real time, and to schedule TDMA broadcasts, which only requires relative time among the nodes. Our algorithm satisfies a relaxed *gradient* property. In particular, the algorithm ensures that the clock skew of two nodes which are distance d apart in the network is bounded by a linear function of d, at almost all times. Finally, our algorithm is practical and easy to implement. It requires little memory and computation, and is suited for resource-bounded wireless nodes.

The remainder of this paper is organized as follows. In Section 3, we describe some related work on clock synchronization. In Sections 4 and 5, we describe our formal model and problem. We present our synchronization algorithm in Section 6, and show that it satisfies the desired properties in Section 7. Finally, we conclude and discuss some future work in Section 8.

3   Related Work

NTP [6] is a widely deployed time synchronization service. NTP relies on a hierarchy of time servers, and assumes that root servers have access to a correct source of real time. In contrast, our algorithm works in a network with no infrastructure support. Access to real time via GPS may exist, but is intermittent. Furthermore, unlike NTP, we tolerate a highly dynamic network, and node failures and joins are accommodated without disturbing the synchronization in the rest of the network. Srikanth and Toueg [7] present an optimal clock synchronization algorithm. However, their algorithm relies on broadcast, and is not suitable for a wireless network. Their algorithm performs only internal synchronization, while we integrate external and internal synchronization. Also, [7] is complicated by the need to tolerate Byzantine failures. Our algorithm only tolerates stopping failures, but is much simpler.

RBS [1] is an efficient CSA designed for wireless networks. RBS performs *post-hoc* synchronization, in which nodes determine the time of an event some time after it has occurred. By contrast, our algorithm performs on-the-fly synchronization, so that we can timestamp an event at the moment it occurs. In addition, RBS performs only internal synchronization.

CesiumSpray [8] is a CSA performing both internal and external synchronization. CesiumSpray achieves high accuracy using the simultaneity of message reception by all nodes in a satellite/wireless network. However, we cannot guarantee simultaneity in the wireless networks we consider because the networks are multihop. In addition, CesiumSpray has lower fault tolerance than our algorithm, and does not achieve the gradient property. Fetzer and Cristian [3] also integrate internal and external synchronization. However, their algorithm is more complex than ours because it deals with faulty GPS information. In practice, we think such failures are unlikely to occur.

Fan and Lynch [2] introduced the gradient property for clock synchronization. They showed that for every CSA, there exist executions in which two nodes distance d apart have clock skew $\Omega(d + \frac{\log D}{\log \log D})$, where D is the diameter of the network. However, their result requires a lower bound on the rate of increase of every node's logical clock. Our algorithm permits logical clocks to be constant for some period of time. Thus, their lower bound does not directly apply.

4   System Model

Our system model consists of three parts: a dynamic set of *nodes*, a *communication network* over which nodes send messages, and a *GPS service* which occasionally informs nodes of the real time. Below, we discuss each part separately.

4.1   Nodes

We wish to model a dynamic system in which nodes can fail or join the system at arbitrary times. To do this, we define N to be the set of *potential nodes*. Each node in N can be in either a *sleeping* or *awake* state, and the state of the node can change at any time. The set of nodes which participate in the clock synchronization algorithm at some time consists of the awake nodes at that time. Failures of nodes are modeled by a node changing from the awake state to the sleeping state. Joins of nodes are modeled by the opposite transition.

Each node is equipped with a hardware clock, which we think of as a variable whose value changes as a differentiable function of time. Denote the value of node i's hardware clock at time t by $H_i(t)$. We assume that the hardware clock of every node has *bounded drift*. More precisely, we assume that there exists $\rho < 1$[1], such that for all nodes i,

$$\forall t : 1 - \rho \leq \frac{dH_i(t)}{dt} \leq 1 + \rho$$

Each node uses its hardware clock and messages it receives from other nodes to compute a logical clock value. Denote the value of node i's logical clock at time t by $L_i(t)$. The clock synchronization algorithm tries to ensure that the logical clock values of the nodes are close to each other, and close to real time.

4.2   Communication Network

Nodes communicate with each other over a message passing network. In some wireless networks, e.g., ad hoc networks, the network changes depending on the set of awake nodes, because the awake nodes are responsible for routing messages among themselves. Other problems may arise if the network becomes disconnected by too many node failures. However,

---

[1]In practice, ρ is very small, on the order of 10^{-5} or 10^{-6}.

such network problems are separate from the problem of clock synchronization. Thus, in this paper, we make a simplifying assumption that there exists a virtual communication link between every pair of nodes $i, j \in N$. We assume that each link is reliable, FIFO, and has bounded delay. Regarding the last assumption, we assume that for every i, j, there exists a constant $d_{i,j} < \infty$, called the *distance* between i and j, which upper bounds the amount of time it takes for a message sent by i to be received by j. For simplicity, we assume $d_{i,j} = d_{j,i}$[2]. Finally, we let the *diameter* of the network be $D = \max_{i,j} d_{i,j}$.

4.3   GPS Service

We imagine that all nodes are equipped with GPS receivers[3], and that occasionally, a GPS service transmits a message informing nodes of the correct real time. The times when these transmissions occur are not under the control of the nodes. We model the receipt of a GPS message at a node i by an input action $gps(t)_i$. This message is intended to inform i that the current real time is t. However, the message may not be accurate, in the sense that i may receive the message *after* real time t. This is because it takes some time for the GPS message to propagate to the entire network. However, we bound the inaccuracy of every GPS message, by assuming that all nodes which are awake during the time interval $[t, t + D]$ receive $gps(t)$ no later than time $t + D$, where D is the diameter of the network.

5   The Clock Synchronization Problem

In this section, we define the internal, external and gradient synchronization properties that the clock synchronization algorithm must satisfy. It is not possible to satisfy these properties at all times and at all nodes, when the nodes are allowed to fail and join/recover. Thus, at any time t, the synchronization properties are only required to hold for nodes which are stable at time t. We say a node is *stable* at time t if it has received at least one GPS input before t, and has not failed since receiving that GPS. Let $S(t) \subseteq N$ denote the set of nodes which are stable at time t. We say an execution is *failure free* if $S(t) = N$ for all t. Otherwise, we say the execution is *failure prone*.

The precision requirement deals with internal synchronization of the nodes, i.e., bounding the difference in the logical clock values of any two nodes. Let ϵ be a parameter. Formally, we require the algorithm satisfy

Requirement 1 (ϵ-Precision) $\forall t \forall i, j \in S(t) : |L_i(t) - L_j(t)| \leq \epsilon$

The accuracy requirement deals with external synchronization of the nodes, i.e., bounding the difference between the logical clock value of any node and real time. Let ϵ be a parameter. We require the algorithm satisfy

Requirement 2 (ϵ-Accuracy) $\forall t \forall i \in S(t) : |L_i(t) - t| \leq \epsilon$

The gradient property was introduced in [2]. It requires that at all times, the difference in the logical clock values of any two nodes which are distance d apart in the communication network (i.e, nodes i, j, such that $d_{i,j} = d$) is bounded by $f(d)$, where f is a nondecreasing function of d. The gradient property is desirable in applications where the clocks of nearby nodes must be well synchronized, whereas the clocks of faraway nodes can be more loosely synchronized. An example of such an application is TDMA. In TDMA, only nearby nodes can collide when transmitting, and thus only such nodes need well synchronized clocks for scheduling their transmissions. Please see [2] for additional motivations and discussion of the gradient property. Our synchronization algorithm satisfies a weakened form of the gradient property, where the gradient property holds only *some* of the time. More precisely, let $T \subseteq \mathbb{R}^{\geq 0}$ consisting of the union of nonzero-length intervals. Then we require that the algorithm satisfy the gradient property for all times in T. Of course, our goal is to maximize the size of T, i.e., maximize $\frac{m(T)}{m(\mathbb{R}^{\geq 0})}$, where $m(\cdot)$ denotes the Lebesgue measure on $\mathbb{R}$. Let α, β be parameters. Formally, we require

Requirement 3 ((T, α, β)-Gradient Precision) $\forall t \in T \forall i, j \in S(t) : |L_i(t) - L_j(t)| \leq \alpha d_{i,j} + \beta$

6   Algorithm

In this section, we describe our clock synchronization algorithm. The pseudo-code of the algorithm is written in the TIOA language [4], and is presented in Figure 1. Below, we give an overview of how the algorithm operates.

---

[2]If $d_{i,j} \neq d_{j,i}$, we can simply redefine $d'_{i,j} = d'_{j,i} = \max(d_{i,j}, d_{j,i})$, then use $d'_{i,j}$ as the distance between i and j.
[3]Actually, it is enough for one node to have a GPS receiver, and for this node to propagate GPS messages to the rest of the network.

Each node in the algorithm maintains two clocks, a *local* clock and a *global* clock. The local clock of node i represents i's best estimate of the current real time. i's global clock represents i's estimate of the largest local clock of any other node. Roughly speaking, i's logical clock is defined to be the maximum of i's local and global clocks[4]. i's local clock is updated by the occasional GPS inputs which i receives. i's global clock is updated by the periodic internal synchronizations which the nodes perform. i's local clock increases at the same rate as i's hardware clock. i's global clock increases at a rate $\frac{1-\rho}{1+\rho}$ times i's hardware clock rate. The reason for the rate of increase of i's global clock is so that i does not overestimate the local clocks of other nodes.

When i receives a GPS signal, it updates its local clock to the value of the signal. However, to avoid setting i's logical clock backwards[5], i stores its current local clock value, and allocates a new local clock initialized to the time in the GPS signal. i's *virtual local clock*, *mlocal*, is set to be the larger of i's local clock, and i's stored local clock value. Moreover, i's logical clock value is defined as the larger of i's *mlocal*, and i's *mglobal*, which will be defined shortly.

The way that the transfer of local clock values after receiving a GPS is actually implemented in our algorithm is slightly different from what is stated above, though it amounts to the same idea. In our algorithm, i stores an array *local* of local clock values, and there is an index *current* keeping track of i's currently active local clock. i increases *local*[*current*] at the same rate as its hardware clock, but keeps *local*[*k*] constant, for all $k \neq current$. When i receives a GPS input, i increases *current*. This has the effect of storing i's previous local clock and starting a new one. The new local clock is initialized to the value of the GPS input. *mlocal* is defined as the maximum value in *local*[·]. In addition, i stores an array *global* of global clock values, and updates the array in a similar way to how i updates its *local* clock values after it receives a GPS. *mglobal* is defined as the maximum value in *global*[·].

To maintain internal synchronization, each node executes its *sync* action approximately once every τ time, where τ is a constant. More precisely, each node i stores an index *next_sync*, and when i finds *local*[*current*] $= \tau \cdot next\_sync$, i performs the *sync*$_i$ action. Then, i increments *next_sync*. The *sync* action sends out a message of the form (*local*[*current*], *max_gps*), where *max_gps* is the largest GPS value that i has received. *max_gps* acts as a "certificate" of how accurate i's *local*[*current*] is. That is, the higher i's value of *max_gps*, the more recently that i has received a GPS input, and thus the more accurate is i's value *local*[*current*].

Now, consider when i receives a synchronization message (t, s), where t is the *local*[*current*] of some other node, say j, and s is j's value of *max_gps*. i checks that j has received at least as recent a GPS value as i, and also that j's *local*[*current*] is greater than i's *global*[*current*], which is i's current estimate of the largest *local*[*current*] of any other node. If both conditions are true, then i stores t in *global*[*current*], and propagates the message (t, s) to i's neighbors in the network. Lastly, if $t \geq \tau \cdot next\_sync$, then there is no need for i to do *sync* when i finds its own *local*[*current*] $= \tau \cdot next\_sync$, since j has already done a *sync* with the same timestamp. In this case, i sets *next_sync* to $\lfloor \frac{t}{\tau} \rfloor + 1$.

Lastly, we describe how the algorithm deals with node failures and joins. If a node fails, it does not interfere with synchronization among the remaining nodes. Thus, nothing is needed to deal with node failures. If a node joins, then it initializes its state to some default values, and waits to receive its first GPS input. The GPS input initializes the new node's state to some correct values, after which the node can participate normally in the algorithm.

7 Analysis

In this section, we show that the algorithm described in Section 6 satisfies the requirements described in Section 5. We first describe the notation used in the proofs.

7.1 Notation

Let i be a node, let *var* be a state variable of i, and let t be a time. Then we let $i.var(t)$ be the value of *var* at i at time t, *before* any discrete actions have occurred at time t. We let $i.var(t^+)$ be the value of *var* at i at time t, *after all* discrete actions have occurred at time t. Thus, for example, if $i.current = 1$ at time 5, and i receives a GPS at time 5 which causes i to increment *current*, then we have $i.current(5) = 1$, and $i.current(5^+) = 2$.

As stated in Section 6, nodes perform *sync* actions approximately every τ time. We also assume that the GPS service updates the nodes every T time, for some constant T. That is, suppose $gps(t)$ occurs at some node at time t_1. Then $gps(t')$ must occur at some node at time t_2, where $t' > t$, and $t_1 \leq t_2 \leq t_1 + T$. Given an action $\xi = gps(t)$, we say t is the *timestamp* of ξ. Given an action $\phi = recv(t, s)$, we say t is the *timestamp* of ϕ.

---

[4]This definition is meant to convey intuition, and is not exactly correct; it is amended in the following paragraphs.

[5]Many applications require logical clocks to be monotonic, in addition to being accurate and precise.

$ClockSync_i, i \in \mathbb{I}$

Constants

$0 \leq \rho < 1$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $0 < \tau$

State

$idle \in$ Boolean, initially $true$
for all $k \in \mathbb{N} : local[k] \in \mathbb{R}$, initially 0
for all $k \in \mathbb{N} : global[k] \in \mathbb{R}$, initially 0
$current \in \mathbb{N}$, initially 0
$next\_sync \in \mathbb{N}$, initially 0

$hardware \in \mathbb{R}$
$max\_gps \in \mathbb{R}$, initially 0
$do\_send \in$ Boolean, initially $false$
$send\_buffer$, a queue of elements of type $\mathbb{R} \times \mathbb{R}$, initially $\quad$ empty

Derived Variables

$mlocal \leftarrow \max_k local[k]$
$mglobal \leftarrow \max_k global[k]$

$logical \leftarrow \max(mlocal, mglobal)$

Transitions

input **wakeup**$_i$
Effect:
$\quad$ if $idle$ then
$\quad\quad$ $idle \leftarrow false$
$\quad\quad$ $current \leftarrow 1$

input **gps**$(t)_i$
Effect:
$\quad$ if $\neg idle$ then
$\quad\quad$ if $t > max\_gps$ then
$\quad\quad\quad$ $max\_gps \leftarrow t$
$\quad\quad\quad$ $current \leftarrow current + 1$
$\quad\quad\quad$ $local[current] \leftarrow t$
$\quad\quad\quad$ $global[current] \leftarrow t$
$\quad\quad\quad$ $next\_sync \leftarrow \lfloor \frac{t}{\tau} \rfloor + 1$

input **recv**$(t, s)_{j,i}$
Effect:
$\quad$ if $\neg idle$ then
$\quad\quad$ if $s \geq max\_gps$ then
$\quad\quad\quad$ if $t > global[current]$ then
$\quad\quad\quad\quad$ $global[current] \leftarrow t$
$\quad\quad\quad$ enqueue (t, s) in $send\_buffer$
$\quad\quad\quad$ $do\_send \leftarrow true$
$\quad\quad\quad$ if $\frac{t}{\tau} \geq next\_sync$ then
$\quad\quad\quad\quad$ $next\_sync \leftarrow \frac{t}{\tau} + 1$

input **crash**$_i$
Effect:
$\quad$ $idle \leftarrow true$
$\quad$ for all $k \in \mathbb{N}$ do
$\quad\quad$ $local[k] \leftarrow 0$
$\quad\quad$ $global[k] \leftarrow 0$
$\quad$ $current \leftarrow 0$
$\quad$ $next\_sync \leftarrow 0$
$\quad$ $max\_gps \leftarrow 0$
$\quad$ $do\_send \leftarrow false$
$\quad$ empty $send\_buffer$

output **sync**$(t, s)_i$
Precondition:
$\quad$ $\neg idle$
$\quad$ $t = local[current]$
$\quad$ $\frac{t}{\tau} = next\_sync$
$\quad$ $s = max\_gps$
Effect:
$\quad$ enqueue (t, s) in $send\_buffer$
$\quad$ $next\_sync \leftarrow next\_sync + 1$
$\quad$ $do\_send \leftarrow true$

output **send**$(t, s)_i$
Precondition:
$\quad$ $\neg idle$
$\quad$ $send\_buffer$ is not empty
$\quad$ $(t, s) =$ head of $send\_buffer$
Effect:
$\quad$ remove head of $send\_buffer$
$\quad$ $do\_send \leftarrow false$

Trajectories

Satisfies
$\quad$ unchanged:
$\quad\quad$ $idle, current, next\_sync, max\_gps, do\_send, send\_buffer$
$\quad$ $1 - \rho \leq d(hardware) \leq 1 + \rho$

$\quad$ $\forall k \in \mathbb{N} :$
$\quad\quad$ if $\neg idle \wedge (k = current)$ then
$\quad\quad\quad$ $d(local[k] - hardware) = 0$
$\quad\quad\quad$ $d(global[k] - \frac{1-\rho}{1+\rho} hardware) = 0$
$\quad\quad$ else
$\quad\quad\quad$ $d(local[k]) = 0$
$\quad\quad\quad$ $d(global[k]) = 0$

Stops at
$\quad$ $(\frac{local[current]}{\tau} = next\_send) \vee (do\_send = true)$

Figure 1: $ClockSync_i$ state and transitions.

If a node i receives a $gps(t)_i$ input, we say the GPS is *useful* if $t > max\_gps$, so that it causes i to change its state. Similarly, if i receives a $recv(t, s)_{j,i}$ input, we say the *recv* is *useful* if $s \geq max\_gps$ and $t > global[current]$, so that it causes i to change its state.

7.2 Proof of Correctness

We first prove a lemma which states that in a failure free execution, the *mglobal* of any node is never much more than the maximum *mlocal* of all the nodes. This lemma is used to show that the algorithm satisfies ϵ-accuracy, even in failure prone executions.

Lemma 7.1 *Consider a failure free execution α. Then $\forall t \forall i \in N : \max_i i.mglobal(t) \leq \max_j j.mlocal(t) + (1 - \rho)D$.*

Proof. We begin by proving $\forall t : \max_i i.global[current](t) \leq \max_j j.mlocal(t) + (1 - \rho)D$, then show that this implies the lemma. To prove the former statement, fix an i and a t, and consider the *last* useful message ϕ which i received before time t. Suppose ϕ was received at time t_2. There are 2 cases. Either ϕ is a GPS, or it is a *recv*.

In the first case, we have $i.global[current](t_2^+) \leq i.local[current](t_2^+)$. Also, since i receives no other useful messages during $(t_2, t]$, we have

$$i.local[current](t) \geq i.local[current](t_2^+) + (1 - \rho)(t - t_2)$$
$$i.global[current](t) \leq i.global[current](t_2^+) + \frac{1 - \rho}{1 + \rho}(1 + \rho)(t - t_2)$$
$$\leq i.local[current](t)$$

The second inequality follows because i increases $i.global[current]$ at a rate of $\frac{1-\rho}{1+\rho}$ times its hardware clock rate, which is at most $1 + \rho$.

In the second case, where ϕ is a *recv*, let j be the node which sent ϕ, and suppose ϕ was sent at time t_1. Then we have

$$i.global[current](t_2^+) \leq j.local[current](t_1) \tag{1}$$

Consider the *first* useful GPS ξ that j receives after time t_1, and suppose j received ξ at time t_3. Let the timestamp of ξ be s_1.

Claim 7.2 $t - t_3 \leq D$

Proof. Suppose for contradiction that $t - t_3 > D$. Then since ξ takes at most D time to reach i, i must have received ξ by time t, say at time t_4. Consider two cases. Either $t_4 < t_2$, or $t_4 \geq t_2$.

In the first case, let $s_2 = j.max\_gps(t_1)$. Then, since j found ξ useful at time t_3, we have $s_1 > s_2$. Since i receives ξ, which has timestamp s_1, at time $t_4 < t_2$, then $i.max\_gps(t_2) \geq s_1$. But i receives ϕ with timestamp s_2 at time t_2, and i found ϕ useful, and so $s_2 \geq i.max\_gps(t_2) \geq s_1$, which is a contradiction.

In the second case, we also get a contradiction, because when i receives ξ at time t_4, i must either find ξ useful, or i found some other message it received in the time interval $[t_2, t_4]$ useful. In either case, this contradicts the assumption that ϕ was the last useful message i received before time t. Thus, we have $t - t_3 \leq D$. $\square$

Since the first GPS which j received after time t_1 occurs at $t_3 \geq t - D$, then j did not change $j.current$ until at least time t_3, and $j.local[current]$ increased at a rate which is at least $1 - \rho$ in the time interval $[t_1, t_3]$. Thus, we have

$$j.local[current](t) \geq j.local[current](t_1) + (1 - \rho)(t_3 - t_1)$$

Also, since i did not receive any useful messages during time interval $(t_2, t]$, we have

$$i.global[current](t) \leq i.global[current](t_2^+) + \frac{1 - \rho}{1 + \rho}(1 + \rho)(t - t_2)$$
$$\leq j.local[current](t_1) + (1 - \rho)(t - t_1)$$
$$\leq j.local[current](t) + (1 - \rho)(t - t_3)$$
$$\leq j.local[current](t) + (1 - \rho)D$$

Where the second inequality follows from Eqn. 1. Thus, we have shown that in all cases, and for all t, we have $\max_i i.global[current](t) \leq \max_j j.mlocal(t) + (1-\rho)D$. Now, now let t_k^* be the k'th time when i incremented $i.current$. Then, we have

$$
\begin{aligned}
i.mglobal(t) &= \max_k i.global[k](t_k^*) \\
&\leq \max_k \max_j j.mlocal(t_k^*) \\
&\leq \max_j \max_k j.mlocal(t_k^*) \\
&\leq \max_j j.mlocal(t)
\end{aligned}
$$

Thus, we have shown that $\forall t : \max_i i.mglobal(t) \leq \max_j j.mlocal(t) + (1-\rho)D$.

The next lemma states that in all executions, including failure prone ones, any node's logical clock value is not much greater than real time.

Lemma 7.3 $\forall t \forall i \in N : \max_i i.logical(t) - t \leq \rho(T+D) + (1-\rho)D$

Proof. Fix an i and a t. Since $i.logical(t) = \max(i.mlocal(t), i.mglobal(t))$, we first show that $i.mlocal(t) - t \leq \rho(T+D)$. Consider the last useful GPS ξ that i received before time t. Suppose ξ occurred at time t_1, and the timestamp for ξ was s. We have $t - t_1 \leq T + D$, because a GPS occurs somewhere in the network every T time, and the GPS takes at most D time to reach i. Now, $i.local[current](t_1^+) = s \leq t_1$, and because i received no other GPS in the time interval $(t_1, t]$, we have $i.local[current](t) \leq i.local[current](t_1^+) + (1+\rho)(t - t_1)$. Thus, we have

$$
\begin{aligned}
i.local[current](t) - t &\leq t_1 + (1+\rho)(t - t_1) - t \\
&= \rho(t - t_1) \\
&\leq \rho(T+D)
\end{aligned}
$$

We have shown that for all i and t, $i.local[current](t) - t \leq \rho(T+D)$. Since $i.mlocal(t) = \max_k i.local[k]$, we have $i.mlocal(t) - t \leq \rho(T+D)$, for all i and t.

By Lemma 7.1, we have that in failure free executions, $i.mglobal(t) \leq \max_j j.mlocal(t) + (1-\rho)D \leq \rho(T+D) + (1-\rho)D$. Now, we observe that if there are failures in an execution, then the failures cannot cause $i.mglobal(t)$ to increase. Thus, in failure prone executions, we also have $i.mglobal(t) \leq \rho(T+D) + (1-\rho)D$. Finally, since $i.logical(t) = \max(i.mlocal(t), i.mglobal(t))$, we have that $i.logical(t) \leq \rho(T+D) + (1-\rho)D$, for all i and t. $\square$

The next lemma states that any stable node's logical clock value is not much less than real time.

Lemma 7.4 $\forall t \forall i \in S(t) : t - \min_i i.logical(t) \leq D + \rho(T+D)$

Proof. Fix a t and an $i \in S(t)$. Since i is stable at time t, i has received a GPS before time t, and has not failed since that GPS. Consider the last GPS ξ that i received before time t, and suppose that ξ occurred at time t_1, and had timestamp s. Then $t_1 - i.local[current](t_1^+) = t_1 - s \leq D$. Also, $i.local[current](t) \geq i.local[current](t_1^+) + (1-\rho)(t - t_1)$. Since $t - t_1 \leq T + D$, we have $t - i.logical(t) \leq t - i.local(t) \leq D + \rho(T+D)$, for any i and t. $\square$

Combining Lemmas 7.3 and 7.4, we get the following.

Theorem 7.5 (Accuracy) *In all executions,* $\forall t \forall i \in S(t) : |i.logical(t) - t| \leq D + \rho(T+D)$.

From Theorem 7.5, we immediately get the following.

Theorem 7.6 (Precision) *In all executions,* $\forall t \forall i, j \in S(t) : |i.logical(t) - j.logical(t)| \leq 2(D + \rho(T+D))$.

To save energy in practice, the GPS service might update the nodes infrequently, so that T can be quite large. Yet even in periods without GPS, the nodes still perform internal synchronization approximately once every τ time. Since τ may be much smaller than T, we would like a sharper bound on precision, stated in terms of τ instead of T. Unfortunately, there is no such bound which holds at all times. The reason for this is that GPS inputs cause "instability" in the network,

as follows. Consider when a node i receives a GPS signal ξ. Since i's logical clock may differ from real time by up to $O(\rho T)$, and since ξ causes i to adjust its logical clock to real time, then $i.logical$ may "jump" by $O(\rho T)$ after i receives ξ. However, since other nodes may not receive ξ at the same time as i, there may be a time period when i's logical clock has jumped forward, but other nodes' logical clocks have not. In this period, the precision is bounded by $O(\rho T)$. On the other hand, we show that if a GPS has *not occurred* within the last D time, then the precision is bounded by $O(\rho \tau)$. To prove this statement, we first show it holds in failure free executions in which *no* GPS inputs occur. Then we show it holds in failure free executions with GPS, and finally, we show it holds in failure prone executions with GPS.

Lemma 7.7 *Consider a failure free execution in which no GPS inputs occur. Then $\forall t \forall i, j \in N : |i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$.*

Proof. Fix an i and j. Let m be the largest integer such that $i.logical(t) \geq \tau m$. Let t_2 be such that $i.logical(t_2) = \tau m$. Let t_1 be the earliest time such that the *mlocal* of any node equals τm. That is, $t_1 = \min_s \exists k : k.mlocal(s) = \tau m$. Let t_3 be the earliest time such that the *mlocal* of any node equals $\tau(m+1)$. In the following analysis, it suffices to assume that $t_1 \leq t_2 \leq t_3 \leq t$. Let $d_1 = t_2 - t_1$, $r = t_3 - t_2$, and $d_2 = t - t_3$. We have that $d_1 \leq D$, because the first node to reach τm sends out a *sync* message, which i receives no more than D time later. After i receives the message, we have $i.logical \geq \tau m$. Similarly, $d_2 \leq D$.

We claim that $r \geq \frac{\tau}{1+\rho} - d_1$. Indeed, since there are no GPS inputs, then the maximum rate of increase of the *mlocal* of any node is at most $1 + \rho$. Since $\max_k k.mlocal(t_1) = \tau m$, $\max_k k.mlocal(t_3) = \tau(m+1)$, and $t_3 - t_1 = r + d_1$, we have $(1+\rho)(r+d_1) \geq \tau$, from which the claim follows.

Now, again because there are no GPS inputs, the rate of increase of the logical clock of any node is at most $1 + \rho$, and at least $\frac{1-\rho}{1+\rho}(1-\rho)$. Thus, we have

$$
\begin{aligned}
j.logical(t) &\leq j.logical(t_3) + (1+\rho)(t - t_3) \\
&\leq \tau(m+1) + (1+\rho)d_2 \\
i.logical(t) &\geq i.logical(t_2) + \frac{(1-\rho)^2}{1+\rho}(t - t_2) \\
&\geq \tau m + \frac{(1-\rho)^2}{1+\rho}(r + d_2) \\
&\geq \tau m + \frac{(1-\rho)^2}{1+\rho}\left(\frac{\tau}{1+\rho} + d_2 - d_1\right)
\end{aligned}
$$

Now, since we have $d_1, d_2 \leq D$, then subtracting the two inequalities above, we get

$$
\begin{aligned}
j.logical(t) - i.logical(t) &\leq \frac{4\rho}{(1+\rho)^2}\tau + \left(1 + \rho - \frac{(1-\rho)^2}{1+\rho}\right)d_2 + \frac{(1-\rho)^2}{1+\rho}d_1 \\
&\leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D.
\end{aligned}
$$

$\square$

Lemma 7.8 *Consider a failure free execution, and let t be a time such that no GPS inputs occur in the time period $[t-D, t]$. Then $\forall i, j \in N : |i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$.*

Proof. Fix an i, j and t. Define t_1, t_2, t_3 as in the proof of Lemma 7.7. Now, since no GPS occurs in the time interval $[t-D, t]$, and $t-t_3 \leq D$, then no GPS occurs in the time interval $[t_3, t]$. Consider two cases. Either no GPS occurs in time interval $[t_1, t_3]$, or some GPS occurs. In the first case, we can prove the lemma using similar ideas as in the proof of Lemma 7.7. For the second case, we consider a simplified version, in which only one GPS occurs in $[t_1, t_3]$. The general case with multiple GPS signals is similar. Let ξ be the GPS input which occurs, and suppose ξ has timestamp s, and ξ occurs at j at time t_j. Following the proof of Lemma 7.7, we can show that $|i.logical(t_j) - j.logical(t_j)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$. Also, since $t_3 - t_1 \leq \frac{\tau}{1+\rho}$, we have

$$
\begin{aligned}
j.local[current](t) &\leq s + \frac{\tau}{1+\rho}(1+\rho) \\
i.local[current](t) &\geq s + \left(\frac{\tau}{1+\rho} - D\right)(1-\rho)
\end{aligned}
$$

8

Now, since there is only one GPS in $[t_1, t_3]$, we have $j.logical(t) = \max(j.logical(t_j), j.local[current](t))$, and $i.logical(t) = \max(i.logical(t_j), i.local[current](t))$. Thus

$$
\begin{aligned}
|j.logical(t) - i.logical(t)| &\leq \max(|j.logical(t_j) - i.logical(t_j)|, |j.local[current](t) - i.local[current](t)|) \\
&\leq \max(\frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D, \frac{2\rho}{1+\rho}\tau + (1-\rho)D) \\
&= \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D
\end{aligned}
$$

The last equality follows because $\rho < 1$. Thus, we have proven the lemma in all cases. $\qquad\square$

Theorem 7.9 (Strong Precision) *Let t be a time such that no GPS inputs occurred in the time interval $[t - D, t]$. Then $\forall i, j \in S(t) : |i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)D$.*

Proof. To prove this theorem, notice first that failures have *no effect* on the precision of synchronization. Second, if $i, j \in S(t)$, then they have each received at least one GPS. Using this fact, the lemma follows from very similar ideas as in the proof of Lemma 7.8. We omit the proof for lack of space. $\qquad\square$

Lastly, we consider the gradient property requirement. Consider two nodes i and j which are distance d apart. Just as GPS inputs caused "instability" which made precision $O(\rho T)$ instead of $O(\rho\tau)$, *sync* messages cause instability which makes the logical clock difference between i and j $O(\rho\tau + D)$, instead of $O(\rho\tau + d)$, as required by the gradient property. However, the gradient property does holds at time t, if there are no GPS and no *sync* inputs in the time interval $[t - D, t]$. More precisely, we have the following.

Theorem 7.10 (Gradient Precision) *Consider any execution, and let t be a time such that no GPS inputs and no sync inputs occur in the time interval $[t - D, t]$. Let $i, j \in S(t)$ be two nodes which are distance d apart. Then $|i.logical(t) - j.logical(t)| \leq \frac{4\rho}{(1+\rho)^2}\tau + (1+\rho)$.*

Proof. We prove a simpler version of the theorem, when the execution is failure free, and when there are no GPS inputs, but possibly some *sync* inputs. The full theorem can be proved in a similar way, and by following ideas in the proofs of Lemma 7.8 and Theorem 7.9. We omit the full proof due to lack of space.

Fix an i, j and t. Let m be the largest integer such that $i.logical(t) \geq \tau m$, and let t_2 be such that $i.logical(t_2) = \tau m$. Let t_1 be such that $j.logical(t_1) = \tau m$, and let t_3 be such that $j.logical(t_3) = \tau(m + 1)$. Let $d_1 = t_2 - t_1$, $r = t_3 - t_2$, and $d_2 = t - t_3$. In the following analysis, it will suffice to consider $t_1 \leq t_2 \leq t_3 \leq t$. By the assumptions of the theorem, j does not receive a *sync* in the time interval $[t - d, t] \subseteq [t - D, t]$.

We claim that j does not receive any useful *sync* inputs during $[t_1, t]$. Indeed, suppose j received a useful *sync* ϕ at time $t' < t - d$. Then the timestamp for ϕ must be at least $\tau(m + 1)$, since otherwise j would not find ϕ useful. Since $t' < t - d$ and $d_{i,j} = d$, then i must receive ϕ before time t. But then we would have $i.logical(t) \geq \tau(m + 1)$, which is a contradiction. Thus, since j does not receive a useful *sync* during $[t_1, t]$, j's logical clock increases at a rate at most $1 + \rho$ during $[t_1, t]$, and so we have

$$
\begin{aligned}
j.logical(t) &\leq j.logical(t_3) + (1 + \rho)(t - t_3) \\
&= \tau(m + 1) + (1 + \rho)d_2
\end{aligned}
$$

Also, i's logical clock increases at a rate at least $\frac{1-\rho}{1+\rho}(1 - \rho)$, so we have

$$
\begin{aligned}
i.logical(t) &\geq i.logical(t_2) + \frac{(1 - \rho)^2}{1 + \rho}(t - t_2) \\
&= \tau m + \frac{(1 - \rho)^2}{1 + \rho}(r + d_2)
\end{aligned}
$$

Now, since j's logical clock increased by τ from time t_1 to t_3, and j's logical clock rate was at most $1 + \rho$ during this time, we have $(1 + \rho)(t_3 - t_1) = (1 + \rho)(r + d_1) \geq \tau$. Thus, $r \geq \frac{\tau}{1+\rho} - d_1$. Also, we have $d_1, d_2 \leq d$, because $i.logical$ must reach τm (resp., $\tau(m + 1)$) within d time that $j.logical$ reaches τm (resp., $\tau(m + 1)$). Thus, subtracting the two inequalities from above, we get

$$
\begin{aligned}
j.logical(t) - i.logical(t) &\leq \frac{4\rho}{(1 + \rho)^2}\tau + \left(1 + \rho - \frac{(1 - \rho)^2}{1 + \rho}\right)d_2 + \frac{(1 - \rho)^2}{1 + \rho}d_1 \\
&\leq \frac{4\rho}{(1 + \rho)^2}\tau + (1 + \rho)d
\end{aligned}
$$

□

Finally, we consider the communication complexity of the algorithm. We show that each node performs roughly one local (i.e. 1-hop) broadcast per τ time. By comparison, many clock synchronization algorithms require each node to broadcast to the entire network during every synchronization period, which is not feasible for energy-conserving wireless nodes.

Theorem 7.11 (Communication Complexity) *Let i be any node. Then i performs at most 1 local broadcast every $\frac{\tau}{1+\rho}$ time.*

Proof. By looking at the *sync* and *recv* actions in Figure 1, we see that each node performs only one local broadcast for each value of *next_sync*. The value of *next_sync* can only increase when the *local*[*current*] of some node increases by τ, and this takes at least $\frac{\tau}{1+\rho}$ time. □

8   Conclusions

We have presented an energy-efficient and fault-tolerant clock synchronization algorithm which integrates internal and external synchronization, and which satisfies a relaxed gradient property. Our algorithm is simple, and easily implementable on resource-bounded wireless nodes.

Our algorithm ensures tight synchronization among nodes when the network is stable, i.e. in periods when a GPS or synchronization operation has not recently occurred. However, when the network is unstable, clock skew may be much larger. Though certain lower bounds exist on the optimal tightness of non-gradient and gradient synchronization, the lower bounds do not immediately apply in our setting, because we do not require a lower bound on the rate of increase of logical clocks. It is an interesting theoretical and practical question whether tight synchronization can be maintained at all times, by allowing logical clocks to remain constant during unstable periods. Another interesting direction of further research is to implement our algorithm on a large scale wireless network, and to compare its average case behavior with the worst case bounds proven in this paper.

References

[1] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[2] Rui Fan and Nancy Lynch. Gradient clock syncrhonization, to appear. In *Proceedings of the Twenty-third Annual ACM PODC*. ACM Press, 2004.

[3] C. Fetzer and F. Cristian. Integrating external and internal clock synchronization. *Journal of Real-Time Systems*, 12(2):123–172, 1997.

[4] Dilsun Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. Timed i/o automata: A mathematical framework for modeling and analyzing real-time systems. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003.

[5] Errol L. Lloyd. *Broadcast scheduling for TDMA in wireless multihop networks.* John Wiley & Sons, Inc., 2002.

[6] D. L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Computers*, 39(10):1482–1493, 1991.

[7] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *J. ACM*, 34(3):626–645, 1987.

[8] P. Verissimo, L. Rodrigues, and A. Casimiro. *Cesiumspray: a precise and accurate global time service for large-scale systems.* Technical Report NAV-TR-97-0001, Universidade de Lisboa, 1997.