PulseSync: An Efficient and Scalable Clock Synchronization Protocol

Christoph Lenzen, Philipp Sommer, and Roger Wattenhofer

***Abstract*—Clock synchronization is an enabling service for a wide range of applications and protocols in both wired and wireless networks. We study the implications of clock drift and communication latency on the accuracy of clock synchronization when scaling the network diameter. Starting with a theoretical analysis of synchronization protocols, we prove tight bounds on the synchronization error in a model that assumes independently and randomly distributed communication delays and slowly changing drifts. While this model is more optimistic than traditional worst-case analysis, it much better captures the nature of real-world systems such as wireless networks.**

The bound on the synchronization accuracy, which is roughly the square-root of the network diameter, is achieved by the novel *PulseSync* protocol. Extensive experiments demonstrate that PulseSync is able to meet the predictions from theory and tightly synchronizes large networks. This contrasts against an *exponential* growth of the skew incurred by the state-of-the-art protocol for wireless sensor networks. Moreover, *PulseSync* adapts much faster to network dynamics and changing clock drifts than this protocol.

***Index Terms*—wireless networks, probabilistic analysis**

I. Introduction

Clock synchronization is a fundamental service to establish a common notion of time across multiple nodes forming a communication network. Accurate synchronization enables a wide range of application and services. For instance, precise synchronization is mandatory for systems that measure the occurence of events such as acoustic or seismic signals, where synchronization errors considerably affect the measurement accuracy. Furthermore, precise timing information is mandatory for coordination amongst different nodes. Wireless networks can greatly benefit from synchronized clocks when scheduling medium access and significantly reduce periods of idle listening.

Many approaches rely on establishing a hierarchical structure amongst multiple nodes. Here, nodes with the highest clock accuracy serve as reference and time information is passed on, possibly traversing multiple intermediate hops. Typical examples of such structured networks include the Global Positioning System (GPS) or Internet time servers using the Network Time Protocol (NTP) [1]. GPS satellites continuously broadcast time information based on their atomic clocks, which are synchronized from ground-based stations. GPS receivers employ data packets from the GPS satellites to provide an accurate clock signal. NTP uses a GPS clock, atomic clock or radio clock as its timesource, which is attached to a time server. NTP clients synchronize to a time server by measuring the round-trip delay of UDP packets containing the current timestamp. In order to reduce the round-trip delay, clients typically only synchronize to a nearby time server, which itself is synchronized to other time servers higher up in the hierarchy.

This approach is not well-suited to, for instance, wireless sensor networks, which consist of resource-constrained nodes deployed for in-situ monitoring of physical phenomena. Access to a GPS clock is often not feasible due to constraints in terms of energy, costs, and satellite coverage (e.g. indoors). During the last decade, several synchronization protocols have been proposed tailored to the constrained nature of wireless embedded systems (see related work in Section II). In contrast to NTP, which exchanges UDP packets with a time server possibly multiple hops away, in this setting the protocol has control over the behaviour of the intermediate nodes. In particular, the synchronization protocol runs also on these nodes and can be utilized to communicate considerably more precise timing information.

Challenges. A crucial part of clock synchronization lies in the ability to compare time information between different spatially separated entities. Ideally, a client node is able estimate its clock offset and drift relative to a reference node based on the comparison of its local clock and the reference clock. The accuracy of this estimation depends highly on the delay required to transfer a clock reading from one place to another and the extent to which non-deterministic contributions to the overall

C. Lenzen is with the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. P. Sommer is with the Autonomous Systems Lab, CSIRO Computational Informatics, Brisbane, Australia. R. Wattenhofer is with the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

delay can be eliminated. Clearly, the synchronization error between adjacent nodes will also influence the synchronization accuracy of nodes further away from the reference node. Furthermore, the network size and the chosen strategy for forwarding time information through the network effects the latency required to synchronize the network and to adapt to dynamics such as changes in the network topology or ambient temperature.

We identify the following key challenges for synchronization in wireless multi-hop networks:

- *Accuracy:* A synchronization protocol has to provide accurately synchronized clock for all nodes. The synchronization error should be kept small, even when a node is multiple hops away from the reference node.
- *Latency:* Time information has to propagate as quickly as possible from the reference node to every other node in the network.
- *Efficiency:* The number of messages sent by each node and the energy budget dedicated to the synchronization routine should be small.

Contributions. While existing protocols meet these requirements reasonably well for networks of small diameter (i.e., those where all nodes can be reached with a small number of hops), they fail to achieve timely and accurate synchronization over larger distances. To overcome this issue, we introduce an appropriate model (Section III) and analyze the achievable bounds on accuracy, latency, and efficiency in this setting (Section V). We propose the new protocol *PulseSync* (Section VI) and prove that it offers asymptotically optimal trade-offs between these design goals (Section VIII).

Simulations on networks with a diameter of up to $D = 100$ nodes give insights about the synchronization accuracy to expect on larger networks that exceed the dimensions of most sensor networks deployed today. Comparing *PulseSync* to the Flooding Time Synchronization Protocol (FTSP), the de facto state-of-the-art in synchronization of wireless sensor networks, we show an exponential improvement in the dependence of the maximal clock skew on D (Figure 1). Note that the figure shows that *PulseSync* features a clearly sublinear dependence of the skew on D, as predicted by our analysis. This justifies the choice of a more optimistic system model, as it is known for long that the skew must be linear in D in the worst case [2]. These findings are not artifacts of unrealistic simulation parameters or network diameters; careful experiments confirm considerable improvements already for 5-10 hops.

Furthermore, by construction most existing synchronization protocols cannot adapt quickly to changes in

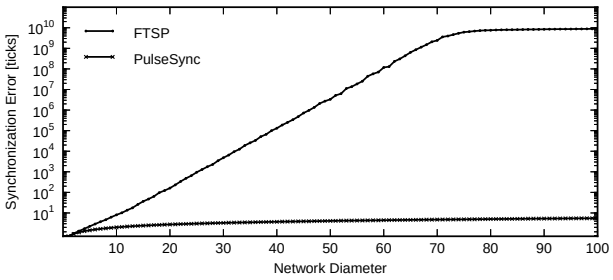

Fig. 1. Simulations of FTSP and *PulseSync* on line topologies. Message delays were sampled from the normal distribution fitted to the measurement data (see Figure 2, 1 tick $= 1\mu s$), clock drifts uniformly at random from [-40,40] ppm and kept constant during each run. The curve for FTSP flattens near diameter 70 due to overflowing variables.

clock drifts (e.g. due to variances in temperature, humidity, or battery voltage) or network connectivity. This is to be attributed to the unaligned sending patterns that are typically employed because they simplify accessing the wireless medium without significant message loss due to interference. A crucial disadvantage of this approach is that the time to spread information throughout the network grows as the *product* of the (average) time between local updates and the network diameter. *PulseSync* avoids this issue by flooding rapid, short "pulses" through the network. Hence it has a much shorter initialization phase and adjusts much faster to changes in topology or drifts, in time in the order of the *sum* of the time between local updates and the network diameter. Our experimental data supports this conclusion.

Note that this mechanism comes with the additional advantage that there is no need for nodes to listen between pulses once they detected any pulse, both on initialization or when joining the network later on. This permits to save energy by turning off radio receivers for a large fraction of the time, without sacrificing robustness of the synchronization technique.

All mentioned experimental results are obtained from a prototype of *PulseSync* on the Opal wireless sensor platform using TinyOS (Section VII) that is published online [3]. The experimental evaluation of the performance of this implementation was carried out on a testbed of diameter $D = 30$ (Section IX). The parameters used in the simulations shown in Figure 1 were derived from these measurements. To allow for a fair comparison, we also performed an experiment with a standard implementation of FTSP on a matching setup.

The remaining sections cover related work (Section II), some properties of normally distributed random variables and linear regression we use in our formal analysis (Section IV), and conclusions we draw from

our results (Section X). We remark that the theory and practice parts of the article are self-contained, with the obvious reservation that the high-level description of *PulseSync* in Section VI is relevant to both parts.

II. Related Work

Clock synchronization has been studied extensively both in wired and wireless networks. The Network Time Protocol (NTP) [1] is commonly used to synchronize devices over the Internet. It uses UDP packets to periodically poll a time server. Packets are timestamped at both the client and server, which allows the client to calculate its clock offset and the message delay. Different algorithms are used to discard packets that contain outliers. In contrast to NTP, which timestamps packets only at the start and the end of a path, synchronization protocols for wireless networks employ MAC layer timestamping [4], [5]. Hence, these protocols achieve improved accuracy since the effects of non-deterministic delays due to buffering in the radio driver or random backoff are mitigated. Reference Broadcast Synchronization (RBS) [6] exploits the broadcast nature of the physical channel to synchronize a set of receivers. Since differences in the propagation times are small in most wireless networks, packets arrive almost simultaneously at all receivers, yielding a common event serving as a reference point. The Timing-sync Protocol for Sensor Networks (TPSN) [5] builds a spanning tree of the network. Nodes synchronize to their parent by a two-way message exchange to estimate delay and offset.

The *Flooding-Time Synchronization Protocol (FTSP)* [4] uses flooding and MAC layer timestamping to reduce the number of messages to one per node and synchronization interval. A root node is elected which periodically floods timestamps into the network, inducing a tree structure. The root node is dynamically elected by the network based on the smallest node identifier. Each node uses a linear regression table to convert between its local hardware clock and the clock of the reference node. Nodes that are synchronized to the root node start broadcasting their local estimation of the global clock.

While in FTSP every node broadcasts synchronization beacons periodically, the Routing Integrated Time Synchronization protocol (RITS) [7] uses a reactive approach to provide post-facto synchronization. Clocks are synchronized only when an event is detected. While the timestamp of an event is forwarded towards the sink node, it is converted from the local time of the sender to the receiver's local time at each hop. Similar to our approach, *Rapid Time Synchronization (RATS)* [8] employs a network-wide flooding to disseminate the local time of the root node, however, no drift compensation is implemented for the forwarding of synchronization beacons. Recently, Ferrari et al. [9] proposed Glossy, which combines flooding and time synchronization by exploiting constructive interference of radio packets. While Glossy allows for flast flooding of synchronization pulses, it requires specialized radio receivers providing very low jitter in the message delay (e.g. the TI CC2420 radio [10]). Furthermore, the differences in the propagation delay between nodes cannot exceed 0.5 microseconds in order to exploit constructive interference at the receivers, which limits the use of Glossy to deployments with short (<150 m) radio links only.

In contrast, *PulseSync* achieves provably sublinear skews with respect to D in a quite generic setting that merely assumes that the expected communication delay is known. This bound is probabilistic, whereas the worst-case accuracy that can be achieved is no smaller than $D/2$ in a general model where message delays vary arbitrarily in the range $[0, 1]$ and clocks are perfect [2]. A worst-case upper bound of $(1 + \rho)D$ can be achieved by a simple algorithm even if in addition clock speeds maybe vary arbitrarily between $1-\rho$ and $1+\rho$ [11]. This upper bound is essentially tight (not only up to factor $2(1 + \rho)$) if one also requires that the computed clocks remain within an optimal envelope of real time [12].[1]

III. Model

In a wireless network, communication takes place by radio. In theory, in order to send a message, a node powers up its radio, transmits the message, and powers the radio down. In practice, of course, there are a number of issues. Does the receiver listen on the respective channel? Is there interference with other transmissions? Is an acknowledgement to be sent? If so, was it successfully received, etc. We will not delve into these matters, although one has to respect the peculiarities of wireless communication when devising clock synchronization protocols for such systems.

Keeping these issues in mind, we choose a simplistic description of the network as a static graph $G = (V, E)$, where V is the set of nodes and E is the set of bidirectional, reliable communication links. If node $v \in V$ sends a message, all neighbors $w \in \mathcal{N}_v$ listening on the channel can receive this message. We focus on the following aspects of wireless systems:

Communication is expensive. The energy consumption of a node whose radio is powered on is orders of magnitude larger than that of a sleeping node (cf. [13]).

[1] Formally, if all local clocks are initialized to 0, at time t the computed clock values must be within $(1 - \rho)t$ and $(1 + \rho)t$.

In fact, in many cases radio usage determines the lifetime of a sensor node. Therefore, we would like to minimize the amount of communication dedicated to the synchronization routine. Consequently, we require that nodes send and receive (on average) one message per *beacon interval* B only.

Communication is inexact. In a distributed system, it is not possible to learn the exact clock values of communication partners. In the wireless setting, this is mainly due to two causes. Firstly, transmission times vary. This effect can be significantly reduced by MAC layer time-stamping [4], yet a fraction of the transmission time cannot be determined exactly. Secondly, the resolution of the sensor nodes' clocks is limited. Thus, quantization errors are introduced that make it impossible to determine the time of arrival of a message precisely (this can also be improved [10]). As these fluctuations are typically not related between different messages, we model them as independently distributed random variables. For the sake of our analysis, we assume their distributions to be identical and refer to the respective standard deviation as *jitter* $\mathcal{J}$. Our results hold for most "reasonable" distributions. For simplicity, we will however assume normally distributed variables with zero mean[2] in this article, a hypothesis which is supported by empirical study [6] as well as our own measurements (see Figure 2).

Sending times are constrained. Due to signal interference, in wireless networks one cannot simply send a message whenever it is convenient. In order to account for this, we define the time it takes in each beacon interval between a node receiving and sending a message to be predefined and immutable by the algorithm. For the reason that every node will receive and transmit only once during every interval, we need only a single value $\tau_v \in \mathbb{R}^+$ for each node $v \in V$, denoting the time difference between receiving and sending the respective messages. This time span also accounts for the fact that it takes some time to receive, send, and process messages. Note that this is a simplification in that this time span is variable for several reasons. However, the respective fluctuations are small enough to have negligible effects, as in a real system the fact that radios are powered down most of the time necessitates that nodes can predict when the next message arrives in order to activate the receiver and listen on the appropriate channel.

Message size is constrained. The number of bits in radio messages should be small for various reasons. Hence, we restrict the "payload" of a message to a small (constant) number of values. We do not formalize this in our model; in particular, we assume unbounded clocks. In practice, a limited number of bits is used to represent clock values and a wrap-around is implemented.

[2]I.e., the expected value of the variable is known and can be subtracted, yielding a distribution of zero mean.

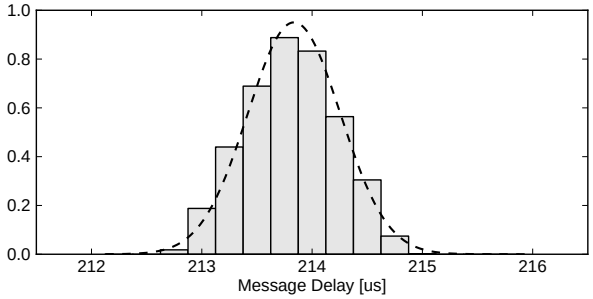

Fig. 2. Measurement of the message delay between two Atmel AT86RF231 radio transceivers. The fitted normal distribution has a mean of 213.8 µs and a variance of 0.18 µs. The probability mass corresponding to each bar is the product of its width and height.

Dynamics. Which nodes can communicate directly may depend on various environmental conditions, in particular interference from inside or outside the network. Thus, in contrast to the previous definition of G, the communication graph is typically not static. Moreover, the speed of the nodes' clocks will vary, primarily due to changes in the nodes' temperatures; we remark that nodes equipped with temperature sensors can significantly reduce this influence [14], [10]. We do not capture these aspects in our model, which assumes a static configuration of the system, both with regard to communication and clock rates. This aspect is addressed by the design of the proposed algorithm, which strives for dependency of computed clock values on a short period of time. Thus, the algorithm will adapt fast to changes in topology or clock speeds.

Let us now formalize the clock synchronization problem in this communication model. Each node $v \in V$ has a local *hardware clock* $H_v : \mathbb{R}_0^+ \to \mathbb{R}_0^+$. It is an affine linear function $H_v(t) = o_v + h_v \cdot t$, where $o_v \in \mathbb{R}_0^+$ is the *offset* and h_v is the *rate* of v's clock. Node v has access to $H_v(t)$ only, i.e., it can read its local clock value, but does neither know o_v nor h_v. The rate h_v determines by how much a local measurement of a difference between two points in time deviates from the correct value. We require that the *relative drift* of H_v is bounded, i.e., $\rho_v := |h_v - 1| \leq \rho < 1$. Here ρ is independent of the number of nodes n, meaning that each clock progresses at most by a constant factor slower or faster than real time. Typical hardware clocks exhibit drifts of at most 50 ppm, i.e., $\rho \leq 5 \cdot 10^{-5}$.

Observe that given an infinite number of messages,

two neighboring nodes could estimate each other's clock values arbitrarily well. Sending clock updates repeatedly and exploiting independence of message jitters, node v can approximate the function $H_w(H_v(t))$, $w \in \mathcal{N}_v$, arbitrarily precisely with probability arbitrarily close to 1. For theory, it is thus mainly interesting to study local clocks with fixed drift in combination with algorithms whose output at time t depends on a bounded number of messages only. In light of our previous statements, the same follows canonically from our goals to (i) minimize the number of messages nodes send in a given time period and (ii) enable the algorithm to deal with dynamics by making it oblivious to information that might be outdated. If we relied on clock values from a large period of time (where the meaning of "large" depends on the speed of changes in environmental conditions), the assumption of clock drifts being constant (up to negligible errors) would become invalid.

A *clock synchronization algorithm* is asked to derive at each node v a *logical clock* $L_v := \mathbb{R}_0^+ \to \mathbb{R}_0^+$ based on local computations, its hardware clock readings, and the messages exchanged. The goal is to minimize the *global skew* $\mathcal{G}(t) := \max_{v,w \in V}\{|L_v(t) - L_w(t)|\}$, using few messages and only recent information.

Note that so far a trivial solution would be to simply set $L_v(t) := 0$ for all times t and nodes v. Naturally, we rather want L_v to behave like a "real" clock. In particular, we expect clock speeds to be close to one and clock values to be closely related to real time. To avoid a cluttered notation, we will adopt the following convention. There is a distinguished root node $r \in V$ that has a perfect clock, i.e., $H_r(t) = t = L_r(t)$ at all times t, and nodes try to synchronize their logical clocks with L_r. This is known as *external synchronization* in the literature, as opposed to the closely related concept of *internal synchronization* where there is no reference to the real time and nodes synchronize their clocks to each other. Note that for the purpose of internal synchronization it is feasible to choose an arbitrary node as reference and assume its clock to be "perfect" without significantly affecting the bounds on the relative drift. Observe that $\mathcal{G}(t) \leq 2\max_{v \in V}\{|L_v(t) - L_r(t)|\}$, i.e., minimizing the global skew is essentially equivalent to synchronizing clocks to $t = L_r(t)$. We do not impose explicit restrictions on the progress speeds of the logical clocks in our model. However, we note that one can ensure smoothly progressing clocks by interpolation techniques, without weakening the synchronization guarantees.

IV. Preliminaries

This section summarizes some standard tools used throughout the article.

Lemma 1. *Given normally distributed random variables $X_1, \dots, X_N$, their sum $X := \sum_{i=1}^N X_i$ is normally distributed with expectation $\mathbb{E}[X] = \sum_{i=1}^N \mathbb{E}[X_i]$ and variance $\mathrm{Var}[X] = \sum_{i=1}^N \mathrm{Var}[X_i]$.*

Lemma 2. *Any normally distributed random variable X satisfies that $P[|X - \mathbb{E}[X]| > \sqrt{\mathrm{Var}[X]}] \in \Omega(1)$ and $P\left[|X - \mathbb{E}[X]| \leq \delta\sqrt{\mathrm{Var}[X]}\right] \in 1 - e^{-\Omega(\delta^2 \log \delta)}$ for any $\delta \in \mathbb{R}^+$.*

Definition 3 (Simple Linear Regression)**.** *Given data points (x_i, y_i), $i \in \{1, \dots, N\}$, such that not all x_i are the same, their* linear regression *is the line $\hat{f}(x) = \hat{s}x + \hat{t}$, where $\hat{s}, \hat{t} \in \mathbb{R}$ are minimizing the expression $\sum_{i=1}^N (\hat{f}(x_i) - y_i)^2$. Denoting by $\bar{\cdot}$ the average of the respective values $\cdot_i$, $i \in \{1, \dots, N\}$, we have*

$$\hat{s} = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2} \quad \text{and} \quad \hat{t} = \bar{y} - \hat{s}\,\bar{x}.$$

Theorem 4. *Assume that we are given a set of measurements $(x_i, \hat{y}_i)$ of data points (x_i, y_i), $i \in \{1, \dots, N\}$, obeying the relation $f(x_i) = y_i$, where $f(x) = sx + t$. Furthermore, assume that $\hat{y}_i = y_i + X_i$, where the X_i are identically and independently normally distributed random variables with expectation μ and variance σ^2. Denote by $\hat{f}(x) = \hat{s}x + \hat{t}$ the linear regression of the data set $\{(x_i, \hat{y}_i)\}_{i \in \{1,\dots,N\}}$. Then we have that*

(i) *$\hat{s}$ is normally distributed with $\mathbb{E}[\hat{s}] = s$ and $\mathrm{Var}[\hat{s}] = \sigma^2 / \sum_{i=1}^N (x_i - \bar{x})^2$,*
(ii) *$\hat{f}(\bar{x}) - f(\bar{x})$ is normally distributed with mean μ and variance σ^2/N.*

V. Lower Bound

In order to derive our lower bound on the global skew of *any* algorithm in our model, we first examine how well a neighbor of the root can synchronize its clock to L_r.

Lemma 5. *Assume that r sends at most $k \in \mathbb{N}$ messages within $T := (t - kB, t] \subset \mathbb{R}_0^+$. Suppose $v \in \mathcal{N}_r$ computes an estimate $\hat{o}_v(t)$ of $o_v(t) := H_v(t) - t$ without—directly or indirectly—relying on any events preceding T. Then the probability that $\hat{o}_v(t)$ has an error of $\mathcal{J}/\sqrt{k}$ is at least constant, i.e., $P[|\hat{o}_v(t) - o_v(t)| \geq \mathcal{J}/\sqrt{k}] \in \Omega(1)$.*

Proof: We claim that w.l.o.g. (i) no other nodes relay information about r's clock values to v, (ii) r sends all messages at time t and (iii) each message contains only the clock value at the time of sending.

To see this, observe first that even if another node knew the state of r exactly, it could do no better than r as its messages are subject to the same variance in delay as r's. Next, including several values into a single message does not help in estimating $o_v(t)$, as the crucial point is

that the time of delivery of the message in comparison to the expected time of its delivery is unknown to both sender and receiver. Thus, all estimates that v derives on r's clock values are inflicted with exactly the same error due to jitter. Moreover, sending a different value than the one at the time of sending only meant that v had to guess, based on its local clock and the messages from r, the value of $H_v(t')$ at the time t' when r read the respective clock value. This however could only reduce the quality of the estimate. As we deliberately lifted any restrictions r had on sending times, there is no advantage in sending the message at a different time than t. Finally, since we excluded the use of any information on times earlier than $t-kB$ in the preconditions of the lemma, r has no valuable information to share except its hardware clock reading at time t.

In summary, r can at best send k messages containing t at time t, such that v will learn that r sent k messages at time t that have been registered at local times $H_v(t+X_i)$, where X_i, $i \in \{1,\dots,k\}$, are independent normally distributed random variables with zero mean and variance $\mathcal{J}^2$. Since X_i is unknown to v, it cannot determine $H_v(t)-t$. At best, it can read the k values $H_v(t+X_i)$ and take each value $H_v(t+X_i)-t$ as an estimate. This can be seen as k measurements of $o_v(t)$ suffering from independent normally distributed errors $h_v X_i \in \Theta(X_i)$ (as $|h_v-1|=\rho_v \leq \rho$ and $\rho<1$ is a constant). Hence, $\hat{o}_v(t)$ is (at best) the mean of v's clock readings minus t. By Lemma 1, this value is normally distributed with mean $o_v(t)$ and variance $\Theta(\mathcal{J}^2/k)$, which by Lemma 2 gives the claimed bound. ∎

At first glance, it seems tempting to circumvent this bound by just increasing the time interval information is taken from. Indeed this improves synchronization quality as long as the model assumption that clock rates and topology do not change remains (approximately) valid. As soon as conditions change quickly, the system will however require more time to adapt to the new situation, thus temporarily incurring larger clock skews.

The given bound on the synchronization quality between neighbors generalizes to multihop communication.

Corollary 6. *Assume that for a shortest path $(v_0 := r, v_1, \dots, v_d)$ and some $k \in \mathbb{N}$, kd messages are sent and received by the nodes on the path within a time interval $T := (t-kB, t] \subset \mathbb{R}_0^+$. Suppose v_d computes an estimate $\hat{o}_{v_d}(t)$ of its hardware clock offset $o_{v_d}(t)$ at time t that does not rely on any events before T. Then the probability that $\hat{o}_{v_d}(t)$ has an error of $\mathcal{J}\sqrt{d/k}$ is constant, i.e., $P[|\hat{o}_{v_d}(t)-o_{v_d}(t)| \geq \mathcal{J}\sqrt{d}/\sqrt{k}] \in \Omega(1)$.*

Proof: Assume w.l.o.g. that $h_{v_i} = 1$ for all $i \in \{1,\dots,d\}$. Denote by $o_i := o_{v_i}(t) - o_{v_{i-1}}(t)$, $i \in \{1,\dots,d\}$ the offset between the clocks of v_i and v_{i-1}. Consider the following scheme. First v_1 determines an estimate $\hat{o}_1(t)$ of $o_{v_1}(t) = o_1$, then v_2 an estimate $\hat{o}_2$ of the offset o_2 towards v_1, and so on. Thus, by incorporating the results into the messages, v_i, $i \in \{1,\dots,d\}$, can estimate $o_{v_i}(t)$ by $\hat{o}_{v_i}(t) = \sum_{j=1}^{i} \hat{o}_j(t)$. Since clocks do not drift and there are no "shortcuts" as $(v_0,\dots,v_d)$ is a shortest path, this scheme is at least as good as an optimal one (obeying the model constraints). Let k_i, $i \in \{1,\dots,d\}$, denote the number of messages node v_i receives from its predecessor. As seen in the proof of Lemma 5, $\hat{o}_i$ is normally distributed with mean o_i and variance $\mathcal{J}^2/k_i$. By Lemma 1, it follows that $\hat{o}_{v_d}$ is normally distributed with mean o_{v_d} and variance $\sum_{i=1}^{d} \mathcal{J}^2/k_i$. Because $\sum_{i=1}^{d} k_i = kd$, this variance is minimized by the choice $k_i = k$ for all $i \in \{1,\dots,d\}$. We get that $\mathrm{Var}[\hat{o}_{v_d}(t)] \geq \mathcal{J}^2 d/k$, which by Lemma 2 yields the desired statement. ∎

Next, we infer our lower bound on the global skew.

Theorem 7. *Suppose that $k \in \mathbb{N}$ and each node sends and receives on average at most one message in B time. If a clock synchronization algorithm determines $L_v(t)$ at all nodes $v \in V$ and times $t \in \mathbb{R}_0^+$ depending on events that happened after time $t-kB$ only, then at uniformly random times t from a sufficiently large time interval we have that $\mathbb{E}[|L_v(t)-t|] \in \Omega(\mathcal{J}\sqrt{d}/\sqrt{k})$, where d is the distance of v from the root.*

Proof: Let $(v_0 := r, v_1, \dots, v_d := v)$ denote a shortest path from r to v. Because all nodes receive on average at most one message in B time, for symmetry reasons we may w.l.o.g. assume that all estimates v obtains on its offset depend on messages along this path only. Let $\mathcal{E}$ be the event that at a time t sampled uniformly at random from a sufficiently large time period it holds that the nodes v_i, $i \in \{0,\dots,d-1\}$, sent and received in total at most $2kd$ messages during the interval $(t-kB, t]$. Because nodes send and receive on average at most one message in B time, linearity of expectation and Markov's bound imply that the probability of $\mathcal{E}$ must be at least $1/2$. If $\mathcal{E}$ occurs, Corollary 6 yields that any estimate v may compute of $H_v(t)-t$ has an error of $\mathcal{J}\sqrt{d}/\sqrt{2k}$ with at least constant probability. ∎

Seen from a different angle, this result states how quickly the system may adapt to dynamics. It demonstrates a trade-off between the contradicting goals of minimizing message frequency, global skew, and the time period logical clock values depend on. Given a certain stability of clock rates and having fixed a desired bound on the global skew, for instance, one can derive

a lower bound on the number of messages nodes must at least send in a given time period to meet these conditions. Similarly, the theorem yields a lower bound on the time span it takes until a node (re)joining the network may achieve optimal synchronization for a given message frequency, granted that the other nodes make no additional effort to support this end.

VI. PulseSync

The central idea of the algorithm is to distribute information on clock values as fast as possible, while minimizing the number of messages required to do so. In particular, we would like to avoid that it takes $\Omega(BD)$ time until distant nodes learn about clock values broadcast by the root node r. Obviously, a node cannot forward any information it has not received yet, enforcing that information flow is directed. An intermediate node on a line topology has to wait for at least one message from a neighbor. On the other hand, after reception of a message it ought to forward the derived estimate as quickly as possible in order to spread the new knowledge throughout the network. Thus, we naturally end up with flooding a *pulse* through the network. In order to keep the number of hops small, the flooding takes place on a breadth-first search (BFS) tree.

To keep clock skews small at all times, each node $v \in V$ does not only minimize its offset towards the root whenever receiving a message, but also employs a drift compensation, i.e., tries to estimate h_v and increase its logical clock at the speed of H_v divided by this estimate. As we modeled H_v as an affine linear function and the fluctuations of message delays as independently normally distributed random variables, linear regression is a canonical choice as a means to compute $L_v(t)$ out of $H_v(t)$ and the k most recent clock updates received.

We need to specify how nodes that are not children of the root obtain accurate estimates of r's clock. Recall that nodes are not able to send a message at arbitrary times. Thus, it is necessary to account for the time span τ_v that passes between the time when node $v \in V$ receives a clock estimate from a parent and the time when it can send a (derived) estimate to its children. The most simple approach here is that if v obtains an estimate $\hat{t}$ of the root's clock value $L_r(t) = t$ from a message received at time t, it sends at time $t + \tau_v$ the value $\hat{t} + (H_v(t + \tau_v) - H_v(t))$ to its children. Thus, the quality of the estimate will deteriorate by at most $|(H_v(t+\tau_v)-H_v(t))-((t+\tau_v)-t)| = |h_v-1|\tau_v \leq \rho\tau_v$. We will refer to this as *simple forwarding*. Intuitively, granted that τ_v is small, i.e., $\max_{v\in V}\{\rho_v\tau_v\} \ll \mathcal{J}/\sqrt{D}$, the additional error introduced by simple forwarding is dominated by message jitter and thus negligible.

In our test setting, this technique already turned out to be sufficient for achieving good results. However, this might not be true in general, due to different hardware, larger networks, harsh environments, etc. Hence we devise a slightly more sophisticated scheme we call *compensated forwarding*. As discussed before, it is fatal to replace the term $H_v(t+\tau_v)-H_v(t)$ by $L_v(t+\tau_v)-L_v(t)$, i.e., approximate the progress of real time by means of the regression line that is computed partially based on the estimate $\hat{t}$ obtained at time t. Instead, we use an independent estimate $\hat{h}_v$ of h_v to compensate the drift. To this end, given $k \in 2\mathbb{N}$, node $v \in V$ computes the regression line defining L_v according to the $k/2$ most recent messages only. The remaining $k/2$ messages nodes may take information from are used to provide clock estimates with simple forwarding. From these values a second regression line is determined, whose slope s should be close to $1/h_v$. As we know that $h_v \in [1-\rho, 1+\rho]$, nodes set $\hat{h}_v$ to $1-\rho$ if the outcome is too small and to $1+\rho$ if it is too big. All in all,

$$\hat{h}_v := \begin{cases} 1-\rho & \text{if } 1/s \leq 1-\rho \\ 1+\rho & \text{if } 1/s \geq 1+\rho \\ 1/s & \text{otherwise.} \end{cases}$$

Apart from sending $\hat{t}+H_v(t+\tau_v)-H_v(t)$ at time $t+\tau_v$ after receiving a message at time t, node v now also includes the value $\hat{t}+(H_v(t+\tau_v)-H_v(t))/\hat{h}_v$ into the message. This (usually) more precise estimate is then used to derive the regression line defining L_v from the $k/2$ most recent messages. Obviously, one cannot use compensated forwarding until nodes received sufficiently many clock estimates; for simplicity, we disregard this in the pseudocode of the algorithm. We remark that a similar approach has been proposed for high latency networks where the drift during message transfer is a major source of error [15].

The pseudocode of the algorithm for non-root nodes is given in Algorithm 2. The root follows Algorithm 1. In the abstract setting, a message needs to contain the two estimates of the root's clock value only. For clarity, in the pseudocode we utilize sequence numbers $i \in \mathbb{N}$, initialized to one. In practice, a message may contain additional useful information, such as an identifier, the identifier of the (current) root, or the (current) depth of a node in the tree. For the root node, the logical clock is simply identical to the hardware clock. Any other node computes $L_v(t)$ as the linear regression of the $k/2$ most recently stored pairs of hardware clock values and the corresponding estimates with compensated forwarding, evaluated at $H_v(t)$. As stated before, the value $\hat{h}_v(t)$ is computed out of the $k/2$ estimates with simple forwarding from the preceding pulses, as the inverse slope of

the linear regression of these values.

Algorithm 1: Whenever $H_r(t) \bmod B = 0$ at the root node r.

wait until time $t + \tau_r$ when allowed to send
send $\langle t + \tau_r, t + \tau_r, i \rangle$ // $H_r(t + \tau_r) = t + \tau_r$
$i := i + 1$

Algorithm 2: Node $v \neq r$ receives its parent's message $\langle \hat{t}, \tilde{t}, i \rangle$ with sequence number i at local time $H_v(t)$.

delete $\langle \cdot, \cdot, \cdot, i - k + 1 \rangle$
store $\langle H_v(t), \hat{t}, \tilde{t}, i \rangle$
wait until time $t + \tau_v$ when allowed to send
send $\langle \hat{t} + H_v(t + \tau_v) - H_v(t), \tilde{t} + \frac{H_v(t+\tau_v) - H_v(t)}{\hat{h}_v}, i \rangle$
$i := i + 1$

VII. Implementation

We present a prototype implementation of the *PulseSync* protocol on Opal wireless nodes to verify the feasibility of our approach in practice.

Hardware Platform. The hardware platform used for the implementation of *PulseSync* is the Opal wireless sensor network platform [16]. It features a low-power Atmel SAM3U microcontroller, which is based on an ARM Cortex-M3 core providing 256 KBytes of program flash and 52 KBytes of RAM. Two separate radio transceivers (Atmel AT86RF212/231) provide multi-band communication capabilities in the 900 MHz and 2.4 GHz ranges according to the IEEE 802.15.4 standard. The clock frequency of the Cortex-M3 core can be scaled up to 96 MHz by multiplying the signal from a 12 Mhz external quartz crystal using a phased-locked loop. We configured the microcontroller to further divide the core clock frequency to generate a 1 Mhz clock used as an input for the Timer Counter block of the SAM3U. The hardware clock is built on top of a 16-bit hardware clock register plus an additional 16-bit software counter that is increased at every overflow of the underlying counter register. Thus, the local clock of the node is a 32-bit integer that is being increased at a frequency of 1 Mhz.

PulseSync Implementation. We implemented *PulseSync* on the Opal platform using TinyOS 2.x [17]. The protocol implementation provides access to a globally synchronized logical clock for other components of the application running on the node.

After startup, the *PulseSync* component is initialized by setting the logical clock to the current hardware clock value and nodes start overhearing the wireless channel for synchronization messages. If no synchronization messages have been received for a certain time, the node will declare itself as the reference node and starts to broadcast periodic synchronization pulses. Each synchronization message contains a global timestamp, sequence number, and root node identifier. After the bootstrap phase, a node will receive periodic pulses from nodes located closer to the reference node. No synchronization pulses will be received if the reference node stops broadcasting due to a hardware failure or the node gets disconnected from the rest of the network. When a node does not receive synchronization messages from a reference node with lower id for the duration of several consecutive beacon intervals, it starts to advertise itself as the new reference node. When a node learns about a change in the network topology, e.g., a reference node with a lower id than the existing one is advertised, it forwards the synchronization packet immediately in order to propagate topology changes as quickly as possible through the network.

Depending on the network topology, a node receives synchronization messages from multiple neighbors. Only the first packet of each pulse arriving at a node will be forwarded. It is very likely that this pulse has been forwarded on a path with minimal hop count from the root node and, therefore, the jitter accumulated along this path is assumed to be smaller than on longer paths. To detect duplicate pulses, we insert a sequence number field into the synchronization message which is increased by the reference node r after each sent pulse.

MAC Layer Timestamping. To minimize clock skews, it is essential that the computed clock estimates are unbiased. Timestamping radio packets at the MAC layer allows to reduce uncertainties in the message delay as much as possible and to account for the expected delay by applying respective corrections [4], [5]. For example, the TinyOS driver for the RF231 radio employs a random backoff between 320 and 4960 microseconds for medium access. In addition, processing delays (e.g., transferring the packet between microcontroller and radio) can account for up to several hundred microseconds.

Ultimately, the accuracy of MAC layer timestamping depends on the resolution of the local hardware clock and the latency of the interrupt occurring on the start of frame delimiter (SFD). Since the hardware clock is discrete, the estimate obtained with an imagined continuous hardware clock is rounded up to the next integer.

Furthermore, the SFD interrupt might be delayed when the microcontroller is within an atomic section of another task. Here comes the tightly concentrated flooding of pulses to our advantage: Following the slotted

programming approach proposed in [18], we can ensure that the nodes are ready to handle synchronization messages without blocking efficient operation of application code. Under this assumption, it is reasonable to expect that the mean interrupt handling time is, again up to minor errors, fixed for any given combination of hardware and message handling code. Such error terms may strongly depend on the combination of radio transceiver, microcontroller, and message handling code, but are uniform in time and independent of the specific sensor node. Hence, it is feasible to determine the (expected) error experimentally during a calibration phase. Note that this has the additional advantage of accounting for possible unidentified further contributions to the error made at each hop. Overall, MAC layer timestamping typically results in an effective jitter of roughly one clock tick, which is equal to 1 µs in our setup.

Wireless Flooding. *PulseSync* seeks to quickly disseminate time information through the network. Therefore, nodes start to transmit synchronization pulses shortly after reception of a new pulse from a neighbour closer to the root. However, in a wireless network radio packets used for these pulses might collide due to interference. This problem is known as the broadcast problem in wireless networks and has been studied extensively in the literature. For the sake of brevity, we keep the discussion concise and refer to [19] for additional details.

In the context of wireless sensor networks, the ability to quickly flood packets through the network is a fundamental primitive for various network protocols and applications, such as data dissemination [20] or time synchronization [4], [7]. Several protocols have been proposed to increase the reliability of flooding, e.g., the *Robust Broadcast Protocol (RBP)* [21] or *Collective Flooding (CF)* [22], which exploit information on the neighborhood of a node.

Rather than trying to avoid interference completely, other approaches employ concurrent transmissions to reduce the latency of flooding. Thereby, the Flash flooding protocol [23] exploits the power capture effect, where a receiver is able to successfully decode the strongest packet amongst multiple concurrent transmissions [24]. Recently, Ferrari et al. proposed *Glossy* [9], which provides a flooding latency of a few milliseconds for networks with a small diameter by exploiting constructive interference by precisely timing the forwarding of the same radio packet at different nodes.

Although we did not employ these sophisticated algorithms in our prototype implementation, they demonstrate that efficient solutions to tackle the broadcast problem are feasible in practice. With such techniques it is possible to complete pulses in less than B time on any real-world topology. Even if the previous pulse has not been completed before the next one is started, the spatial separation of two pulses is large enough not to cause additional collisions.

Energy Efficiency. Energy is one of the limiting factors one has to keep in mind when designing applications and protocols for sensor networks. Current time synchronization algorithms, e.g., FTSP, broadcast periodic beacons at a certain rate (e.g., every 10 seconds). However, the exact time point when a node broadcasts the next time synchronization beacon is not coordinated with its neighbors. One advantage of this approach is its simplicity since neighboring nodes do not need to agree on a common schedule when beacons can be sent or received. However, a drawback of uncoordinated sending in low-power operation scenarios is that topology changes result in changes in the listening schedules that need to propagate through the network.

In contrast, *PulseSync* simplifies low-power operation, since pulses will arrive only during a specific and short time interval, independently of changes in the flooding tree. A synchronization beacon can be transmitted in roughly 1.4 ms using the RF231 radio. Forwarding a received packet requires only a few milliseconds depending on the amount of random backoff introduced by the MAC protocol. This permits to operate the radio chip at a very low duty-cycle, even with guard times of a few hundred microseconds. For example, having the radio active for receiving and transmitting beacons during a period of 500 ms (an entire pulse in our 31-node network) within every beacon interval of 10 seconds results in a duty-cycle of 5%. As long as a node is synchronized to the rest of the network, it can easily calculate when the next synchronization pulse will arrive and switch on the radio accordingly. Significant further improvements can be achieved by exploiting knowledge on the transmission schedule (for our setup: ≈10 ms radio activity, resp. 0.1% duty cycle) or by increasing the beacon interval B; the latter comes at the expense of a slower initialization and response to possible dynamics. What is more, this approach allows for a temporal decoupling of sensing, computation, and communication tasks, as proposed in the slotted programming approach [18].

VIII. Analysis

In this section, we prove a strong probabilistic upper bound on the global skew of PulseSync. To this end, we will first derive a bound on the accuracy of the estimates nodes compute of their hardware clock rates. Then we will proceed to bounding the clock skews themselves.

Definition 8 (Pulses)**.** *Pulse $i \in \mathbb{N}$ is* complete *when all messages with sequence number i have been sent. We say that pulses are* locally separated *if for all $i \in \mathbb{N}$ each node sends its message with sequence number i at least αB time before receiving the one with sequence number $i+1$, where $\alpha \in \mathbb{R}^+$ is a constant.*

After the initialization phase is over, i.e., once all nodes filled their regression tables, nodes are likely to have good estimates on their clock rates. Interestingly, the quality of the estimates is independent of the clock drifts, as the respective systematic errors are the same for all estimates of the root's clock and thus cancel out when computing the slope of the regression line.

Lemma 9. *For $v \in V$ and arbitrary $\delta \in \mathbb{R}^+$ we define that $\Delta_h := \min\{2\rho, \delta\mathcal{J}\sqrt{D}/(k^{3/2}B)\}$. Suppose that pulses are locally separated. Then, at any time t when at least $k \in 2\mathbb{N}$ pulses are complete, it holds that $P[|h_v/\hat{h}_v(t) - 1| \leq \Delta_h] \in 1 - e^{-\Omega(\delta^2 \log \delta)}$.*

Proof: Assume that $(v_0 := r, v_1, \dots, v_d := v)$ is the path from r to v in the BFS tree (i.e., in particular $d \leq D$). Consider a simply forwarded estimate $\hat{t}$ that has been received by v at time t. Backtracking the sequence of messages leading to this value and applying Lemma 1, we see that r sent its respective message at a time that is normally distributed around $t - \sum_{i=0}^{d-1} \tau_{v_i}$ with variance $d\mathcal{J}^2$. Thus, since node v_i increases each simply forwarded estimate at rate h_{v_i}, $\hat{t} - t$ is normally distributed with mean $\sum_{i=0}^{d-1}(h_{v_i} - 1)\tau_{v_i}$ and variance at most $\sum_{i=0}^{d-1}((1+\rho_{v_i})\mathcal{J})^2 \leq 4d\mathcal{J}^2$. By Theorem 4, thus the slope $\hat{s}$ of the regression line v computes is normally distributed with mean $1/h_v$ and variance $\mathcal{O}(d\mathcal{J}^2/(h_v k^3 B^2)) \subseteq \mathcal{O}(D\mathcal{J}^2/(k^3B^2))$. Here we used that pulses are locally separated, implying that $\sum_{i=1}^{N}(x_i - \bar{x})^2 \in \Omega(h_v k^3 B^2)$ (in terms of the notation from the theorem). Recall that $h_v \geq 1 - \rho_v \geq 1 - \rho > 0$ and we made sure that $\hat{h}_v \in [1-\rho, 1+\rho]$. Thus, we can infer that the error $|h_v/\hat{h}_v - 1| = |h_v s - 1|$ is bounded both by $1/(1-\rho) \in \mathcal{O}(1)$ times the deviation of the slope from its mean and 2ρ. Hence, the claim follows by Lemma 2. ∎

Based on the preceding observations, we can now prove a bound on the skew between a node and the root.

Theorem 10. *Suppose pulses are locally separated. Denote by $(v_0 := r, v_1, \dots, v_d := v)$ the shortest path from the root to $v \in V$ along which estimates of r's clock values are forwarded. Set $\mathcal{T}_v := \sum_{i=1}^{d} \tau_{v_i}$, i.e., the expected time an estimate "travels" along the path. Suppose $t_1 < t_2$ are two consecutive times when v receives a message and suppose that at time t_1 at least $3k/2$, $k \in 2\mathbb{N}$, pulses are complete. Then, for any $\delta, \varepsilon \in \mathbb{R}^+$ and Δ_h as in Lemma 9, it holds that*

$$P\left[\forall t \in [t_1, t_2) : |L_v(t) - t| \leq \varepsilon\mathcal{J}\sqrt{\tfrac{D}{k}} + \Delta_h\mathcal{T}_v\right] \in 1 - \tfrac{kD}{2}e^{-\Omega(\delta^2 \log \delta)} - e^{-\Omega(\varepsilon^2 \log \varepsilon)}.$$

Proof: Since at least $3k/2$ pulses are complete, according to Lemma 9 during the last $k/2$ pulses we had at any time t and for any node v_i, $i \in \{0, \dots, d-1\}$, that $|h_{v_i}/\hat{h}_{v_i}(t) - 1| \leq \Delta_h$ with probability $1 - e^{-\Omega(\delta^2 \log \delta)}$. Denote by $\mathcal{E}$ the event that the last $k/2$ estimates with compensated forwarding that v received have been increased at *all* nodes on the way at a rate differing by no more than Δ_h from 1. Since $\hat{h}_{v_i}$ only changes when a message is received, we can apply the union bound to see that $\mathcal{E}$ occurs with probability at least $1 - kDe^{-\Omega(\delta^2 \log \delta)}/2$.

Assume now that $\mathcal{E}$ happened and also that $1 - kDe^{-\Omega(\delta^2 \log \delta)}/2 > 0$ (as otherwise the bound is trivially satisfied). Consider the errors of the above $k/2$ estimates. Each estimate has been "on the way" for expected time $\mathcal{T}_v$, i.e., the absolute of the mean of its error is bounded by $\Delta_h\mathcal{T}_v$. The remaining fraction of the error is due to message jitter. Note that if the estimates $\hat{h}_{v_i}$ are very bad, this might amplify the effect of message jitter. However, since $1 - \rho \leq \hat{h}_{v_i} \leq 1 + \rho$, we can account for this effect by multiplying $\mathcal{J}$ with a constant. Thus, we can still assume that at each hop, a normally distributed random variable with zero mean and variance $\mathcal{O}(\mathcal{J}^2)$ is added to the respective estimate of r's current clock value, yielding a random experiment which stochastically dominates the true setting (with respect to clock skews). In summary, by Lemma 1 w.l.o.g. each estimate that v obtains suffers an independently and normally distributed error with mean $\mu \in [-\Delta_h\mathcal{T}_v, \Delta_h\mathcal{T}_v]$ and variance $\mathcal{O}(d\mathcal{J}^2) \subseteq \mathcal{O}(D\mathcal{J}^2)$.

By Theorem 4, the slope of the regression line used to compute L_v suffers a normally distributed error of zero mean and standard deviation $\mathcal{O}(\mathcal{J}\sqrt{D}/(k^{3/2}B))$. Denote by $\bar{t}$ the mean of the times when v received the $k/2$ messages it computes the regression from. As for all times $\bar{t} < t \in [t_1, t_2)$ we have that $t - \bar{t} \leq (k/2+1)B$, we can bound[3]

$$|L_v(t) - t| \leq |L_v(\bar{t}) - \bar{t}| + \left|\left(\frac{h_v(t)}{\hat{h}_v} - 1\right)(\bar{t} - t)\right|,$$

where the second term is normally distributed with zero mean and standard deviation $\mathcal{O}(\mathcal{J}\sqrt{D/k})$.

[3]Note that the use of the expression $L_v(\bar{t})$ here is an abuse of notation, as we refer to the y-value the regression line that v computes at time t assigns to x-value $H_v(\bar{t})$.

Again by Theorem 4, the deviation of the line at the time $\bar{t}$ is normally distributed with mean μ and standard deviation $\mathcal{O}(\mathcal{J}\sqrt{D/k})$. Thus, applying Lemma 2 and the union bound yields that, conditional to $\mathcal{E}$, the event $\mathcal{E}'$ that $\forall t \in [t_1, t_2) : |L_v(t) - t| \leq \varepsilon\mathcal{J}\sqrt{D/k} + \Delta_h\mathcal{T}_v$ occurs with probability $1 - e^{-\Omega(\varepsilon^2+\log\varepsilon)}$. We conclude

$$\begin{aligned} P[\mathcal{E}'] &\geq P[\mathcal{E}] \cdot P[\mathcal{E}'|\mathcal{E}] \\ &\in \left(1 - \frac{kD}{2}e^{-\Omega(\delta^2+\log\delta)}\right)\left(1 - e^{-\Omega(\varepsilon^2+\log\varepsilon)}\right) \\ &\subseteq 1 - \frac{kD}{2}e^{-\Omega(\delta^2+\log\delta)} - e^{-\Omega(\varepsilon^2+\log\varepsilon)}, \end{aligned}$$

as claimed. ∎

The term $\mathcal{T}_v$ occurring in the bound provided by the theorem motivates the following definition.

Definition 11 (Pulse Time)**.** *Denote for $v \in V$ by $(v_0 := r, v_1, \ldots, v_d := v)$ the shortest path from r to v along which PulseSync sends messages and set $\mathcal{T}_v := \sum_{i=0}^{d-1} \tau_{v_i}$. The* pulse time *is $\mathcal{T} := \max_{v\in V}\{\mathcal{T}_v\}$.*

Theorem 10 implies that the proposed technique is indeed optimal provided a comparatively weak relation between B and P is satisfied.

Corollary 12. *Suppose pulses are locally separated and that $B \geq \sqrt{\mathcal{T}^2\log(kD)/(k^2\log\log(kD))}$. Then the expected clock skew of any node $v \in V$ in distance d from the root at any time t when at least $3k/2$ pulses are complete is bounded by $\mathbb{E}[|L_v(t) - t|] \in \mathcal{O}(\mathcal{J}\sqrt{d/k})$.*

Proof: W.l.o.g., we assume that $d = D$ (otherwise just consider the subgraph induced by all nodes within distance d from r). For all $i \in \mathbb{N}$, we define that $\Delta_h(i) := \frac{i\mathcal{J}}{k^{3/2}B}\sqrt{\frac{\log(kD)D}{\log\log(kD)}}$. By assumption, we have $\Delta_h(i)\mathcal{T}_v \leq \frac{i\mathcal{J}\mathcal{T}}{k^{3/2}B}\sqrt{\frac{\log(kD)D}{\log\log(kD)}} \leq i\mathcal{J}\sqrt{\frac{D}{k}}$, giving by Theorem 10 for $\delta = i\sqrt{\log(kD)/\log\log(kD)}$ and $\varepsilon = i$ that $P\left[|L_v(t) - t| > 2i\mathcal{J}\sqrt{D/k}\right] \in e^{-\Omega(i^2\log i)}$. It follows that

$$\begin{aligned} &\mathbb{E}[|L_v(t) - t|] \\ \leq\ &\sum_{i=0}^{\infty} P\left[|L_v(t) - t| > 2i\mathcal{J}\sqrt{\frac{D}{k}}\right] 2\mathcal{J}\sqrt{\frac{D}{k}} \\ \in\ &\left(1 + \sum_{i=1}^{\infty} e^{-\Omega(i^2\log i)}\right) 2\mathcal{J}\sqrt{\frac{D}{k}} \subseteq \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D}{k}}\right), \end{aligned}$$

as claimed. ∎

Note that Corollary 12 requires a *lower* bound on B, while in practice we are interested in choosing B large to minimize energy consumption. Of course, a beacon interval that is too large is undesired because one wants the system to adapt quickly to dynamics. However, the pulse time is a trivial lower bound on this response time, i.e., it does not make sense to choose $Bk \in o(\mathcal{T})$. Thus, we remain with a small gap of $\mathcal{O}\left(\sqrt{\log(kD)/\log\log(kD)}\right)$ to countervail the fact that the drift compensations the nodes on a path of length D employ are dependent, making best use of the limited number of recent clock estimates that are available.

In practice, it is important to bound clock skews at *all* times and *all* nodes, as algorithms may fail if presumed bounds are violated even once. Naturally, a probabilistic bound cannot hold with certainty; indeed, in our model arbitrary large skews must occur if we just wait for sufficiently long. However, for time intervals that are bounded by a polynomial in n times B we can state a strong bound that holds with high probability.

Corollary 13. *Let for $i \in \mathbb{N}_0$ t_i denote the time when the i^{th} pulse is complete. Provided that the prerequisites of Theorem 10 are satisfied, the total number of pulses p is polynomial in n, and $B \geq \mathcal{T}/k$, we have that*

$$\max_{t\in[t_{3k/2},\, t_p]}\{\mathcal{G}(t)\} \in \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D\log n}{k\log\log n}}\right)$$

with probability $1-1/n^c$ for an arbitrary constant $c > 0$.

Proof: Observe that $3k/2 \leq p$ or nothing is to show because $t_p < t_{3k/2}$. As also $D < n$, we have that kD is polynomially bounded in n. Thus, values $\delta, \varepsilon \in \mathcal{O}\left(\sqrt{\log n/\log\log n}\right)$ exist such that $1 - kD/2 \cdot e^{-\Omega(\delta^2\log\delta)} - e^{-\Omega(\varepsilon^2\log\varepsilon)} \geq 1 - 1/(pn^{c+1})$. We apply Theorem 10 to each node $v \in V$ and each pulse $i \in [3k/2, p]$. Due to the bound $B \geq \mathcal{T}/k \geq \mathcal{T}_v/k$ and the definition of Δ_h, we get for all times t from the respective pulse that

$$\begin{aligned} |L_v(t) - t| &\in \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D\log n}{k\log\log n}} + \Delta_h\mathcal{T}_v\right) \\ &\subseteq \mathcal{O}\left(\mathcal{J}\sqrt{\frac{D\log n}{k\log\log n}}\right) \end{aligned}$$

with probability at least $1 - 1/(pn^{c+1})$. The statement of the corollary then follows by the union bound applied to all nodes and all pulses $i \in [3k/2, p]$. ∎

Note that when considering all nodes, the lower bound on B relaxes to $\mathcal{T}/k$, i.e., we can achieve the stated bound despite the fastest possible adaption to dynamics. Moreover, comparing the previous bounds to the results from simulation and implementation of the algorithm (Figures 1, 3, and 6), we find the predictions on the synchronization quality of the algorithm met well.

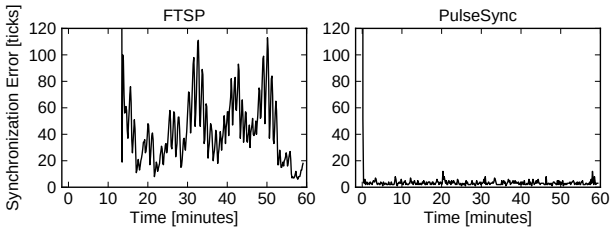

Fig. 3. Maximal synchronization error among the first 15 hops from r during the first hour of the test runs. The 0 on the x-axis is slightly shifted to better show the behaviour of PulseSync on initialization.

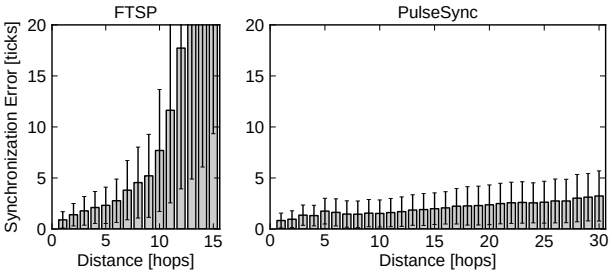

Fig. 4. Time-average of maximal synchronization error from the root for FTSP and PulseSync. Bars indicate standard deviation over time.

IX. Evaluation

In this section, we evaluate our implementation [3] of the PulseSync protocol in a wireless testbed. In order to demonstrate the behaviour of PulseSync in face of large network diameters, nodes are connected such that they form a line topology. We stress that Corollaries 12 and 13 show that the distance from the root is the parameter that is essential with respect to skews. We evaluate the synchronization accuracy using PulseSync and compare our results with the Flooding Time Synchronization Protocol (FTSP) [4], the currently most common clock synchronization protocol for wireless sensor networks.

A. Testbed Setup

We use an indoor testbed of 31 Opal nodes, which are placed in close proximity, thus forming a single broadcast domain. We enforce a line topology in software, resulting in a network diameter of 30 hops. To evaluate the synchronization accuracy of the two different clock synchronization protocols (PulseSync and FTSP), we use the radio packets of the reference node as an external event common to all other nodes. Since the propagation delay of such a probe packet is negligible for such short distances, we can assume that the packet is received simultaneously by all nodes. On reception of such a time probe, each node records the timestamp of the packet reception and converts it to a logical clock value (slightly decreasing the accuracy due to potentially differing processing times). The interval between time probe events is the same as the beacon interval (10 seconds).

B. Measurement Results

For both PulseSync and FTSP, we report measurement results for a four hour run. Roughly 44,600 data points were collected during each experiment. Since nodes were placed in close proximity to each other, we observed almost no packet loss during the experiment runs. We observe that both protocols transmit roughly the same number of beacons (FTSP: 42,289, PulseSync: 42,040).

Nodes running the standard implementation of FTSP will clear the content of their regression table if the synchronization error between the received beacon and the estimated global time of the node exceeds a certain threshold repeatedly. For example, the implementation of FTSP in TinyOS 2.1.1 uses a threshold of 500 clock ticks. After the regression table has been cleared, such a node will stop sending its own synchronization beacons until it has re-established synchronization to its reference node. Although this fallback can mitigate the effects of single outliers, it will effectively disconnect the network in our case, and do so permanently once the synchronization quality of FTSP deteriorates beyond the threshold due to a large hop count. Since this behaviour prohibits to examine the behaviour of FTSP's underlying synchronization mechanism for larger diameters, we slightly modified FTSP to prevent it from clearing its regression table during our experiments.

Initialization Phase. The experiments demonstrate that FTSP requires a significant amount of time to bootstrap the synchronization tree. We observe that it takes FTSP roughly 15 minutes until all nodes within the first 15 hops are synchronized to the reference node, while PulseSync establishes synchronization within a few seconds since time information is propagated within a single beacon interval through the network (see Figure 3). We denote $t = 0$ as the time when the root node has broadcasted its first synchronization beacon, neglecting small differences in the exact start time of different sensor nodes. The issue of slow initialization because of slow flooding is aggravated by FTSP due to the fact that nodes need to collect several clock estimates before they can start to broadcast (accurate) estimates of the root's clock by reading the values of their regression lines.

Synchronization Accuracy. To evaluate the synchronization accuracy of both protocols, for every received time probe we calculated the maximum synchronization error between the reference node and each other node.

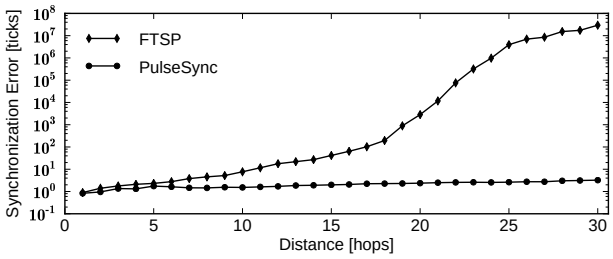

Fig. 5. Time-average of maximal synchronization error from the root node for FTSP and PulseSync plotted on a logarithmic scale.

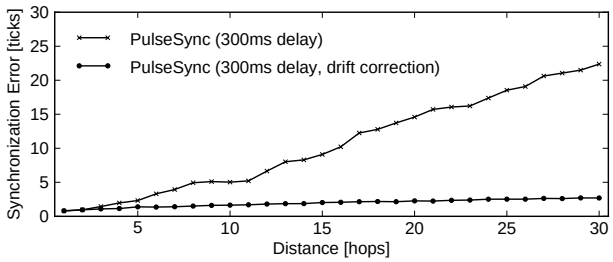

Fig. 6. Time-average of synchronization error for PulseSync with and without drift correction in case of slow forwarding.

In order to avoid the effect of transient behavior, we ignored all probes during the first 1000 seconds of the experiment. PulseSync has a worst-case synchronization error of 19 µs (2.06 µs on average) on our 31-node testbed. Although PulseSync and FTSP have the same message complexity (one message per node and synchronization period), PulseSync achieves a significantly improved synchronization accuracy on the same network topology (see Figures 4 and 5). Our experiments confirm our finding that the synchronization error to the reference node increases rapidly when using FTSP, while it grows slowly when employing PulseSync. On the same network topology, the performance of FTSP in terms of the synchronization error is comparable when we consider only nodes with a small hop distance from the root, but the gap widens soaringly with increasing hop count.

Slow Forwarding of Clock Estimates. With different hardware, environment, network topology, or application constraints, it might not be possible to perform flooding as fast as in our setup. In such scenarios, the independent drift estimation proposed in Section VI may lead to considerable improvements. To confirm this experimentally, we repeated the previous experiment twice with the modification that nodes wait 300 milliseconds before forwarding the received clock estimates (see Figure 6). In the first run, we used simple forwarding (like in the run shown in Figure 3), in the second compensated forwarding. Neglecting the initialization phase, simple and compensated forwarding resulted in maximal (average) global skews of 33 (10.60) and 20 (1.94) ticks, respectively. Compensated forwarding thus achieves essentially the same performance as immediate simple forwarding.

Note that, as it computes a second regression line, compensated forwarding uses twice the number of data points for computing the logical clock. Using the same total number of data points would increase the standard deviation of skews by factor $\sqrt{2}$ and thus yield slightly worse results. Still, compensated forwarding is clearly beneficial if immediate forwarding is not possible.

X. Conclusions

We analyzed the clock skew that can be obtained in networks without external clock reference. We established tight asymptotic bounds on the trade-offs between accuracy, latency, and efficiency via analysis of an abstract model and demonstrated by experimentation that it captures the salient properties of real-world networks.

The experiments demonstrate that *PulseSync*, our protocol tailored to wireless sensor networks, offers much better accuracy and latency than FTSP. This is achieved without additional overhead in terms of communication, computation, or required hardware, making *PulseSync* a promising candidate to replace FTSP as standard in clock synchronization for such networks. The derived theory is however more general, and allows for slower dissemination of information in lack of a fast broadcast mechanism. In this setting, the linear term in the bound derived in Theorem 10 might dominate the synchronization accuracy also for networks of realistic size. Similarly, one can employ *PulseSync* also if the tree induced by the flow of information changes quickly between pulses, at the expense of potentially reducing the quality of the drift compensation. Here, the algorithm offers a convenient, implicit adaption to changing topology that is hard to realize in an energy-efficient manner with previous algorithms. We believe that the presented techniques are of practical merit also in these scenarios.

Future Work. It would be of interest to study *PulseSync* on a wider range of topologies, especially in conjunction with dynamics. Some results have been reported in [25], but examining dynamic multi-hop networks remains a challenge in terms of implementation as well as measurements. Moreover, there are questions related to the robustness of the protocol. For instance, the root node constitutes a single point of failure. It would be desirable that synchronization is not lost due to a crashing node (unless the network gets disconnected) or to make the protocol fully self-stabilizing. Ultimately, the goal is a fully-fledged implementation of *PulseSync* offering

good trade-offs between scalability, energy-efficiency, accuracy, and robustness for a wide range of settings.

Acknowledgements. Christoph Lenzen has been supported by the Swiss National Science Foundation, the Swiss Society of Friends of the Weizmann Institute of Science, and the German Research Foundation (reference number Le 3107/1-1).

REFERENCES

[1] D. L. Mills, "Internet Time Synchronization: the Network Time Protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[2] S. Biaz and J. Welch, "Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions," *Information Processing Letters*, vol. 80, no. 3, pp. 151–157, 2001.

[3] P. Sommer, "PulseSync Source Code," Available online on GitHub, 2014, http://github.com/phsommer/pulsesync.

[4] M. Maróti, B. Kusý, G. Simon, and A. Lédeczi, "The Flooding Time Synchronization Protocol," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 39–49.

[5] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync Protocol for Sensor Networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, 2003, pp. 138–149.

[6] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002, pp. 147–163.

[7] J. Sallai, B. Kusý, A. Lédeczi, and P. Dutta, "On the Scalability of Routing Integrated Time Synchronization," *Proceedings of the 3rd European Workshop on Wireless Sensor Networks (EWSN)*, pp. 115–131, 2006.

[8] B. Kusý, P. Dutta, P. Levis, M. Maróti, A. Lédeczi, and D. Culler, "Elapsed Time on Arrival: A Simple and Versatile Primitive for Canonical Time Synchronization Services," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 1, no. 4, pp. 239–251, 2006.

[9] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient Network Flooding and Time Synchronization with Glossy," in *Proceedings of the 10th International Conference on Information Processing in Sensor Networks (IPSN)*, 2011, pp. 73–84.

[10] T. Schmid, P. Dutta, and M. B. Srivastava, "High-Resolution, Low-Power Time Synchronization an Oxymoron No More," in *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, 2010, pp. 151–161.

[11] T. K. Srikanth and S. Toueg, "Optimal Clock Synchronization," *Journal of the ACM*, vol. 34, no. 3, pp. 626–645, 1987.

[12] C. Lenzen, T. Locher, and R. Wattenhofer, "Tight Bounds for Clock Synchronization," *Journal of the ACM*, vol. 57, no. 2, pp. 1–42, 2010.

[13] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004, pp. 95–107.

[14] T. Schmid, Z. Charbiwala, R. Shea, and M. Srivastava, "Temperature Compensated Time Synchronization," *IEEE Embedded Systems Letters*, vol. 1, no. 2, pp. 37–41, 2009.

[15] A. Syed and J. Heidemann, "Time Synchronization for High Latency Acoustic Networks," in *Proceedings of the 25th International Conference on Computer Communications (INFOCOM)*, 2006, pp. 1–12.

[16] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brunig, "Opal: A Multiradio Platform for High Throughput Wireless Sensor Networks," *IEEE Embedded Systems Letters*, vol. 3, no. 4, pp. 121–124, 2011.

[17] P. Levis, S. Madden, and J. Polastre, "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*. Springer, 2005, pp. 115–148.

[18] R. Flury and R. Wattenhofer, "Slotted Programming for Sensor Networks," in *Proceedings of the 9th International Conference on Information Processing in Sensor Networks (IPSN)*, 2010, p. 24.

[19] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal Clock Synchronization in Networks," in *Proceedings of the 7th Conference on Embedded Networked Sensor Systems (SenSys)*, 2009, pp. 225–238.

[20] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, 2004, pp. 15–28.

[21] F. Stann, J. Heidemann, R. Shroff, and M. Zaki, "RBP : Robust Broadcast Propagation in Wireless Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006, pp. 85–98.

[22] T. Zhu, Z. Zhong, T. He, and Z.-l. Zhang, "Exploring Link Correlation for Efficient Flooding in Wireless Sensor Networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010, pp. 49–63.

[23] J. Lu and K. Whitehouse, "Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks," in *Proceedings of the 28th Conference on Computer Communications (INFOCOM)*, 2009, pp. 2491–2499.

[24] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler, "Exploiting the Capture Effect for Collision Detection and Recovery," in *Proceedings of the 2nd Workshop on Embedded Networked Sensors (EmNetS)*, 2005, pp. 45–52.

[25] P. Sommer, "Wireless Embedded Systems: Time, Location, and Applications," Ph.D. dissertation, ETH Zurich, 2011.

Christoph Lenzen received a diploma in mathematics from the University of Bonn, Germany, in 2007 and a PhD in computer science from ETH Zurich, Switzerland, in 2011. Currently he is a postdoctoral fellow at MIT. His research interests include fault-tolerance, clock synchronization, and distributed graph algorithms. He has co-authored 25 papers. He received the Best Paper Award at PODC 2009 and an ETH medal for his PhD thesis.

Philipp Sommer received a Master of Science (2007) and a PhD degree (2011) in Electrical Engineering and Information Technology from ETH Zurich, Switzerland. Since 2011 he is a postdoctoral fellow at CSIRO Computational Informatics in Brisbane, Australia. He is also an adjunct lecturer at the University of Queensland, Australia. He has co-authored 17 papers. His research interests include time synchronization and localization in wireless networks.

Roger Wattenhofer is a full professor at the Information Technology and Electrical Engineering Department, ETH Zurich, Switzerland. He received a diploma (1995) and a doctorate (1998) in Computer Science from ETH Zurich. His research interests cover a variety of algorithmic and systems aspects in computer science and information technology. He published over 200 papers in different communities: distributed computing, networking, and theory.