Implicit Network Timing Synchronization With Phase-Only Updates

Sriram Venkateswaran and Upamanyu Madhow
Department of ECE, University of California Santa Barbara, CA 93106
Email: {sriram, madhow}@ece.ucsb.edu

***Abstract*—Timing synchronization is a critical requirement for implementation of a number of performance-enhancing strategies in mesh networks, including Time Division Multiplexing (TDM) for more efficient resource usage, and sleep scheduling for power efficiency. In this paper, we propose a simple algorithm for *overhead-free* maintenance of timing synchronization and investigate its scalability with the size of the network. In the proposed scheme, the nodes only use information obtained from the timing of existing communication in the network: each node updates only its clock phase when it receives a packet, depending on the difference between the expected and actual packet arrival times. The uncompensated frequency differences across nodes lead to irreducible phase errors, which we bound by introducing the notion of an *averaged system*. We use a linear programming formulation to show that these phase errors can grow with the network size, thereby concluding that simple, phase-only adjustments might suffice to maintain synchrony in small networks, but additional frequency adjustments are necessary for larger networks.**

I. Introduction

Maintaining timing synchronization across a network of spatially dispersed nodes is important for a number of advanced functionalities, such as efficient resource sharing using Time Division Multiplexing (TDM), sleep scheduling for energy-efficient operation, and localization using time of arrival, or time difference of arrival, based techniques. In this paper, we investigate the limits of an *overhead-free* and *easy-to-implement* timing synchronization scheme in the context of TDM-based resource sharing in a mesh network. Specifically, we answer the following question: how well can timing synchronization between neighbors be maintained, when the nodes only adjust their clock phases (and not frequencies) based on the *implicit* timing information obtained from the TDM schedules themselves?

To sustain communication in a TDM-based mesh network, the difference between clocks at neighboring nodes must be less than a maximum tolerable value, called the *guard time*. This is typically achieved in two stages – first, by *establishing* and subsequently, by *maintaining* synchrony between neighbors. At startup, different nodes are assumed to be completely asynchronous, so that the first step is to use explicit signaling to establish a common time reference. Such explicit signaling can also be employed to get estimates of the propagation delays (inclusive of processing delays) between the nodes. Since network set-up times of seconds or even minutes are acceptable, the one-time overhead of this procedure is not expected to be a bottleneck. We therefore focus in this paper on the second step of timing reference *maintenance*. Clocks at different nodes run at slightly different rates relative to a nominal rate due to manufacturing tolerances, which are typically on the order of 10-100 parts per million (ppm). In addition, clock rates and phases can drift slowly over time due to temperature variations. Such phenomena causes the clock phases at different nodes to drift apart, and we must compensate for this drift "often enough" to maintain synchrony. One approach to timing reference maintenance is to simply rerun the first step of timing reference establishment at periodic intervals. However, this procedure incurs significant communication overhead. In this paper, we propose an algorithm for time reference maintenance where the nodes leverage information from the existing communication in a TDM-based network to periodically update their clock phases, thereby eliminating the need for communication overhead.

Algorithm: When a node $\mathcal{N}_{rx}$ receives a packet from its neighbor $\mathcal{N}_{tx}$, $\mathcal{N}_{rx}$ compares the *actual* reception time with the *expected* reception time based on its own clock. This allows the receiving node $\mathcal{N}_{rx}$ to *implicitly* estimate the timing error with its neighbor $\mathcal{N}_{tx}$, without requiring explicit synchronization beacons. The receiver $\mathcal{N}_{rx}$ then adjusts its clock phase based on the implicit measurement, so that the timing error with $\mathcal{N}_{tx}$ is reduced. Thus, our approach is completely distributed, with each node adjusting its clock independently. Note that these adjustments are all coupled through the network transmission schedule; the timing adjustments made by a node based on the timing of its received messages impact the times at which it transmits, and hence the adjustments made by nodes who receive these transmissions.

This approach to time reference maintenance has two key benefits: (1) it eliminates the overhead associated with dedicated communication for synchronization and (2) since the clocks only adjust their phases and *not* their frequencies, it is particularly easy to implement. This is especially true if the clocks are implemented in software, where recurrent phase adjustments simply correspond to changing the contents of a register occasionally.

While these phase adjustments tend to reduce the error between neighboring nodes, the uncompensated difference in frequencies increases the timing error. This translates directly to the overhead (guard time) needed to sustain communication. The main question we investigate in this paper is – are recurrent phase-only adjustments, based on implicit timing error measurements, sufficient to compensate for the frequency

978-1-4244-9848-2/11$26.00©2011 IEEE

drifts and sustain communication with "tolerable" overhead?

Contributions: Our key contributions are as follows:

(a) We analyze the efficacy of a *simple, overhead-free* synchronization maintenance algorithm that compensates for phase offsets in a recurrent fashion based on implicit timing error measurements with neighbors.

(b) While our algorithm is asynchronous, with different subsets of nodes performing updates in different slots, we provide fundamental insight by introducing and analyzing an averaged, synchronous system, in which nodes perform weighted updates, with weights depending on the transmission schedule in the original asynchronous system.

(c) For the averaged system, we prove that phase-only updates lead to irreducible phase errors and the magnitude of the worst-case pairwise phase offset depends on the distribution of clock skews in the network and grows with network size. We provide a linear programming formulation for finding the worst-case skews (from the point of maximizing the worst-case pairwise phase offset) for a given network topology.

(d) We show that the worst-case phase offsets for the averaged system are a lower bound on those for the original system. We conclude from our numerical results that phase-only updates may be acceptable in small networks, but do not scale to large networks.

Related Work: While there is a vast literature on timing synchronization in networks, we believe this is the first paper to introduce the concept of *implicit* timestamps based on ongoing communication and to *analyze* the scalability of a decentralized algorithm that only updates the clock phases. The firefly-inspired algorithm [1] can be made to work with implicit timestamps, but it is only designed for phase synchrony, and does not handle either propagation delays or oscillator skew. The Reachback Firefly Algorithm (RFA) in [2] adapts the firefly algorithm to account for propagation delays in its phase adaptation and uses periodic resynchronization to handle oscillator skew. However, reference [2] does not *analyze* the impact of oscillator skew or quantify the scaling law associated with the resulting phase error between neighbors and the size of the network.

A number of algorithms ([3], [4], [5]) that have been proposed for distributed timing synchronization can be broadly classified as "consensus algorithms" [6]. These algorithms typically proceed in two stages: (1) estimate relative offset and skew with each neighbor and (2) use neighbor's estimates of offset and skew (typically with respect to a reference node) along with pairwise estimates to arrive at an estimate of one's offset and skew. However, these algorithms are not amenable to an *implicit* implementation because all of them require *explicit* exchange of additional information to achieve consensus on *both frequency and phase*. The use of this information is critical for decoupling the problems of phase and frequency adjustments, in order to be able to run separate consensus algorithms for each of them. The proposed implicit algorithm is also consensus-like, but using phase-only updates makes it much simpler than the algorithms in [3], [4], [5]. Of course, leaving the frequency errors uncompensated means that, unlike most of the consensus literature, the quantities of interest (the phases, in our case) do not converge to a common value. Thus, an important new contribution here is to quantify the price paid for such sloppy consensus in terms of irreducible phase errors, in terms of the scaling of the phase error with network size.

In contrast to the distributed algorithms discussed so far, references [7] (reference broadcast from a transmitter to synchronize a cluster of receivers), [8] (hop-by-hop synchronization based on a rooted spanning tree) and [9] (linear regression to correct skew based on flooding of a timestamp from a dynamically elected root node) adopt more centralized approaches to synchronization. While [7], [8] also only adjust phases, they do not investigate the scaling of phase errors with network size.

II. System Model And Algorithm

We consider a network of N nodes, labeled $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_N$, each equipped with a clock that runs at a *nominal frequency* of f_{nom} Hz. The *actual* frequency of the clock at $\mathcal{N}_i$, denoted by f_i, differs slightly from the nominal rate of the clock f_{nom}, and is written as $f_i = f_{nom}(1 + \rho_i)$ where ρ_i is called the skew, or drift, of the clock. The magnitude of the skew is assumed to be less than ρ_{max}, which is typically on the order of 10-100 parts per million (ppm); for example, $\rho = 20 \times 10^{-6}$ corresponds to a skew of 20 ppm. Differences in the skews across nodes are caused by manufacturing tolerances. The variation of the skew at a given node with respect to time is typically very slow, hence we approximate ρ_i as a constant that takes values in $[-\rho_{max}, \rho_{max}]$.

Phase evolution with free running clocks: It is convenient to describe the system from the point of view of an external observer who possesses a clock that runs at exactly the nominal rate f_{nom}. When the time on the external observer's clock is t, let $\varphi_i(t)$ denote the measure of time at $\mathcal{N}_i$. We refer to $\varphi_i(t)$ as the clock *phase* of $\mathcal{N}_i$. Suppose that $\mathcal{N}_i$ does not adjust its clock phase in the time interval $[t_1, t_2]$ and allows its clock to "run freely". In this period, $\mathcal{N}_i$'s clock phase increases linearly as,

$$\frac{d\varphi_i(t)}{dt} = \frac{f_i}{f_{nom}} = (1 + \rho_i) \qquad t_1 < t < t_2 \tag{1}$$

We term the quantity f_i/f_{nom} as the *normalized frequency* and denote it by F_i.

Implicit Phase Error Measurement: The network employs a Time Division Multiplexed (TDM) schedule in which transmissions begin at integer multiples of a *slot time*, denoted by T_{slot}, according to the *transmitter's clock*. Suppose that $\mathcal{N}_j$ transmits to its neighbor $\mathcal{N}_i$ in slot s. Therefore, $\mathcal{N}_j$ starts transmitting when $\varphi_j(t) = sT_{slot}$. We assume that nodes have estimates of the propagation delays, including processing times, and subtract them out from the packet reception times. Thus, if $\mathcal{N}_i$ begins receiving this packet at $\varphi_i(t)$, it can implicitly conclude that its clock is behind $\mathcal{N}_j$'s clock by $\varphi_j(t) - \varphi_i(t) = sT_{slot} - \varphi_i(t)$.

978-1-4244-9848-2/11$26.00©2011 IEEE

Phase Adjustment Algorithm: $\mathcal{N}_i$ uses this implicit measurement to reduce its phase error with respect to $\mathcal{N}_j$ by making a linear phase adjustment. Letting t^- and t^+ denote the times on the external observer's clock just before and after the phase jump, we have,

$$\varphi_i(t^+) = \varphi_i(t^-) + \beta\big[\varphi_j(t^-) - \varphi_i(t^-)\big] \tag{2}$$

where $0 < \beta < 1$ is a design parameter.

Quasi-synchronous approximation: To simplify analysis, we assume that the phase and frequency adjustments are made at integer multiples of the slot times based on the external observer's clock, rather than on the receiver's clock (which keeps changing as we make phase and frequency adjustments). This approximation causes a second order error (because of measuring the phase offset between nodes at a time that is slightly offset from the true time) that is negligible compared to the phase offsets themselves. In simulations of the original system, we do not make this approximation, and verify that the quasi-synchronous approximation is indeed valid. With the quasi-synchronous approximation, the clock phases are adjusted as in (2) *only* at the slot boundaries ($t = sT_{slot}$), and then evolve linearly according to the relative frequency offsets of the nodes across a slot.

We define the *normalized mean frequency* to be,

$$\overline{F} = \frac{1}{N}\sum_{i=1}^{N} F_i \tag{3}$$

This is the common part of the clock phase increment over the sth slot, and hence does not impact phase differences across nodes. The latter are actually driven by the normalized *excess frequencies*

$$\delta_i = F_i - \overline{F}\ ,\ i = 1, \ldots, N$$

where δ_i is the amount by which the phase of the ith node drifts relative to the average over a slot.

Discrete time dynamics: Let $\varphi_i[s] = \varphi_i(sT_{slot}^-)$ denote the phase of the ith node just before the right edge of the sth slot. When node i receives a packet from its neighbor node j in slot $s+1$, its phase evolves as follows:

$$\varphi_i[s+1] = \varphi_i[s] + \beta\left(\varphi_j[s] - \varphi_i[s]\right) + \overline{F} + \delta_i \tag{4}$$

On the other hand, if $\mathcal{N}_i$ transmits a packet or is not involved in any transmissions in slot $s+1$, its clock runs freely and its phase evolves as,

$$\varphi_i[s+1] = \varphi_i[s] + \overline{F} + \delta_i. \tag{5}$$

According to the quasi-synchronous approximation, the phase updates occur synchronously at all nodes receiving packets, and can be conveniently expressed in vector form. Defining $\boldsymbol{\varphi}[s] = (\varphi_1[s], \varphi_2[s], \ldots, \varphi_N[s])$ and $\boldsymbol{\delta}[s] = (\delta_1[s], \delta_2[s], \ldots, \delta_N[s])$, we have

$$\boldsymbol{\varphi}[s+1] = G_s\boldsymbol{\varphi}[s] + \boldsymbol{\delta} + \overline{F}\mathbf{1} \tag{6}$$

where $\mathbf{1}$ denotes the vector with all components equal to 1, $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_N)$ is a vector containing the excess skews of the nodes and G_s is a matrix defined as follows: (1) If $\mathcal{N}_i$ receives a packet from $\mathcal{N}_j$ in slot s, $G_s(i,j) = \beta, G_s(i,i) = 1-\beta$ and $G_s(i,j') = 0\ \forall j' \neq i,j$ and (2) if $\mathcal{N}_i$ does not receive a packet from any node in slot s, $G_s(i,i) = 1$ and $G_s(i,j) = 0\ \forall j \neq i$. We refer to G_s as the *system matrix* at slot s. Note that G_s is a stochastic matrix (each row has nonnegative entries summing to one), so that $G_s\mathbf{1} = \mathbf{1}$ (i.e., $\mathbf{1}$ is an eigenvector of G_s with eigenvalue 1).

Network Start-up: While our focus here is to understand the limits of implicit synchronization *maintenance* with phase-only updates, our simulations do include a gateway-led start-up scheme for coarse initial synchronization (no attempt is made to optimize this). A gateway node broadcasts its time and other nodes in its neighborhood "set" their clocks to this value. Each of these nodes then broadcasts its time enabling nodes that are two hops from the gateway to set their clocks. This broadcast is made at a random instant in an interval $[\tau_{min}, \tau_{max}]$ after its clock was set. If $\mathcal{N}_i$'s clock is already set when it hears $\mathcal{N}_j$'s broadcast, it simply adjusts its clock to the mean value. Specifically, if $\mathcal{N}_j$'s time when it makes the broadcast is φ_j and at this point, $\mathcal{N}_i$'s time is φ_i, $\mathcal{N}_i$ adjusts its clock to $(\varphi_j + \varphi_i)/2$. Each node makes one such broadcast and the network can be coarsely synchronized in this fashion in a time period that scales linearly with the number of nodes. We assume throughout the paper that nodes have good estimates of the propagation delay to their neighbors. This can be done by exchanging multiple packets with neighbors at the time of network startup and estimating the propagation delay in a manner similar to [10]. Any residual errors in estimating the propagation delay are considered to be subsumed in the phase jitter in our model.

III. Averaged System & Analysis

Each time node $\mathcal{N}_i$ receives a packet, it adjusts its clock phase to reduce the timing error with the transmitter. Our goal is to understand whether such recurrent phase adjustments suffice to keep the timing error between neighbors (induced by the differing clock frequencies) in check. However, the node that transmits a packet to $\mathcal{N}_i$ changes over slots, thereby introducing "fine details" into the dynamics of $\mathcal{N}_i$'s phase evolution. In an effort to smooth out these fine details and understand the behavior of the system on the average, we introduce the notion of a fictitious *averaged system*. In each slot of the averaged system, *every* node updates its phase based on a weighted average of the phases of *all* its neighbors. The weights are derived from the communication pattern in the actual system and do not change with time – this allows us to quantify the performance through a linear programming formulation. We capture the performance of the algorithm by the worst-case pairwise phase error between neighbors, which is precisely the overhead (guard-time) needed to maintain a TDM schedule. In the next section, we show that this metric provides a lower bound on the guard time required in the actual system. We now describe the model for the averaged system and then analyze it.

978-1-4244-9848-2/11$26.00©2011 IEEE

Averaged System Model: We consider a random communication pattern for the actual system and model the system matrices $\{G_s\}_{s=0}^{\infty}$ (equivalently the set of links that are chosen concurrently) as being chosen independently from a set $\mathcal{S}_{matching} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$. We denote the system matrix for the averaged system by $\overline{\mathcal{G}}$ and define it to be $\overline{\mathcal{G}} = \sum_{i=1}^{M} p_i \mathcal{G}_i$. The phase evolution in the averaged system is therefore exactly as in (6), except that G_s is replaced by $\overline{\mathcal{G}}$.

$$\boldsymbol{\varphi}[s+1] = \overline{\mathcal{G}}\boldsymbol{\varphi}[s] + \boldsymbol{\delta} + \overline{F}\mathbf{1} \tag{7}$$

For simplicity, we assume throughout this paper that the averaged system matrix $\overline{\mathcal{G}}$ is symmetric. This corresponds to the links in the network having the same probability of being active in either direction.

Analysis of averaged system: Since the timing error between nodes depends only on the *difference* between clock phases, it is useful to decompose the phases into mean and excess phases, as follows:

$$\boldsymbol{\varphi}[s] = \overline{\varphi}[s]\mathbf{1} + \boldsymbol{\varphi}_{ex}[s] \tag{8}$$

where $\overline{\varphi}[s] = \mathbf{1}^T\boldsymbol{\varphi}[s]/N = \frac{1}{N}\sum_{i=1}^{N}\varphi_i(s)$ is the average phase and $\boldsymbol{\varphi}_{ex}[s]$ represents the excess phase. For the purposes of analysis, the average phase can be discarded since it does not contribute to the timing error between nodes, which drives our algorithm.

$\overline{\mathcal{G}}$ is a stochastic matrix, being a convex combination of stochastic matrices. Therefore, its largest eigenvalue is one, with $\mathbf{1}/\sqrt{N}$ being the corresponding unit norm eigenvector. For convergence of consensus style algorithms, this largest eigenvalue must not be repeated. The condition for this is that the graph corresponding to $\overline{\mathcal{G}}$ must be connected [11] (clearly needed for network-wide convergence). The evolution of the excess phase can be obtained by excising the first eigenmode from $\overline{\mathcal{G}}$ to get the matrix,

$$\overline{\mathcal{G}}_{ex} = \overline{\mathcal{G}} - \frac{\mathbf{1}\mathbf{1}^T}{N} \tag{9}$$

Using the fact that $\overline{\mathcal{G}}$ is symmetric and that excess frequencies, being deviations from the mean, sum to zero ($\mathbf{1}^T\boldsymbol{\delta} = 0$), we can show that the excess phases evolve as (details omitted for lack of space),

$$\boldsymbol{\varphi}_{ex}[s+1] = \overline{\mathcal{G}}_{ex}\boldsymbol{\varphi}_{ex}[s] + \boldsymbol{\delta} \tag{10}$$

Iterating (10) and using the fact that all eigenvalues of the excised matrix $\overline{\mathcal{G}}_{ex}$ are less than 1 in magnitude, we can show that the excess phases in "steady-state" ($s \to \infty$) are given by,

$$\boldsymbol{\varphi}_{ex,\infty} = \sum_{k=0}^{\infty}\overline{\mathcal{G}}_{ex}^{k}\boldsymbol{\delta} = (\mathbb{I} - \overline{\mathcal{G}}_{ex})^{-1}\boldsymbol{\delta} \tag{11}$$

Thus, the "directed" phase error between neighboring nodes $\mathcal{N}_i$ and $\mathcal{N}_j$, in steady-state, can be obtained from the excess phases $\boldsymbol{\varphi}_{ex,\infty}$ as $e_{ij} = \boldsymbol{c}_{ij}^T\boldsymbol{\varphi}_{ex,\infty}$, where $\boldsymbol{c}_{ij}$ picks out the elements in the ith and jth location of $\boldsymbol{\varphi}_{ex,\infty}$ and subtracts them i.e. $\boldsymbol{c}_{ij}[i] = 1, \boldsymbol{c}_{ij}[j] = -1, \boldsymbol{c}_{ij}[l] = 0\ \forall l \neq i, j$. For the worst-case phase error between $\mathcal{N}_i$ and $\mathcal{N}_j$, we maximize $e_{ij} = \boldsymbol{c}_{ij}^T\boldsymbol{\varphi}_{ex,\infty}$ over the excess frequencies $\boldsymbol{\delta}$ and the mean frequency $\overline{F}$, with the frequencies satisfying the following constraints:

(a) The frequencies $F_i = \overline{F} + \delta_i$ are bounded: $1 - \rho_{max} \leq \overline{F} + \delta_i \leq 1 + \rho_{max}\ \ \forall i$.

(b) Since the frequencies are bounded, the mean frequency $\overline{F}$ is also bounded: $1 - \rho_{max} \leq \overline{F} \leq 1 + \rho_{max}$.

(c) The excess frequencies must sum to zero: $\mathbf{1}^T\boldsymbol{\delta} = 0$.

Since the objective function e_{ij} as well as the constraints are linear in the decision variables $\{\overline{F}, \boldsymbol{\delta}\}$, we can use linear programming to find the frequency distribution (across nodes) that maximizes the phase error between nodes $\mathcal{N}_i$ and $\mathcal{N}_j$. To obtain the maximum phase error between *any* pair of neighboring nodes in steady-state, we solve such a linear program for each pair of neighbors, and pick the largest among the resulting errors.

We consider the special case of $\overline{F} = 1$ to provide some insight into the solution of these linear programs. Substituting $\overline{F} = 1$ into the above conditions, the set of allowed excess frequencies Λ is given by $\Lambda = \{\boldsymbol{\delta} : \mathbf{1}^T\boldsymbol{\delta} = 0, |\delta_i| \leq \rho_{max}\ \forall i\}$. It is known [12] that one of the *extreme points* of the feasible set Λ is an optimizer of the linear program. Let $\boldsymbol{\delta}_{ex}$ denote *any* extreme point of Λ. Using the definition of an extreme point [12], we can show that (details omitted): (a) If the number of nodes N is even, then exactly $N/2$ entries of $\boldsymbol{\delta}_{ex}$ equal ρ_{max} and the other $N/2$ entries equal $-\rho_{max}$ and (b) If the number of nodes N is odd, then $(N-1)/2$ entries of $\boldsymbol{\delta}_{ex}$ equal ρ_{max}, another $(N-1)/2$ entries equal $-\rho_{max}$ and there is one entry that is equal to zero. Since one of the extreme points is an optimizer, we conclude that the largest directed phase error between neighbors occurs with roughly half the nodes running at the maximum frequency and the other half running at the minimum frequency for *any* network topology. We use this characterization to solve the linear programs efficiently.

IV. Bound on Actual System

We now show that the worst-case error from the averaged system provides a bound on the errors between neighbors in the actual system. Consider a fixed set of excess frequencies $\boldsymbol{\delta}$ and suppose that the system matrices G_t are picked from the set $\mathcal{S}_{matching} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$ with probabilities $\{p_1, p_2, \ldots, p_M\}$. Let the system matrices $G_0, G_1, \ldots, G_s$ correspond to one such realization over s slots. Iterating (6), we obtain that the phase of the actual system evolves as follows:

$$\begin{aligned}\boldsymbol{\varphi}[s] = G_{s-1}G_{s-2}\ldots G_0\boldsymbol{\varphi}[0] + s\overline{F}\mathbf{1} \\ + \big(\mathbb{I} + \sum_{k=1}^{s-1} G_{s-1}G_{s-2}\ldots G_{s-k}\big)\boldsymbol{\delta} \quad s \geq 2\end{aligned} \tag{12}$$

The second term only contributes to the mean phase. Under a suitable connectedness condition [11] which is needed for consensus, we can show that the same is true for the first term as well as s gets large:

$$\lim_{s\to\infty} G_sG_{s-1}\ldots G_0 = \mathbf{1}\boldsymbol{\gamma}^T \tag{13}$$

978-1-4244-9848-2/11$26.00©2011 IEEE

where the elements of $\boldsymbol{\gamma}$ are nonnegative and $\boldsymbol{\gamma}^T\mathbf{1} = 1$. As before, we can therefore throw out the first and second terms in (12) when considering the evolution of the excess phases, and obtain

$$\boldsymbol{\varphi}_{ex}[s] \approx \left(\mathbb{I} + \sum_{k=1}^{s-1} G_{s-1}G_{s-2}\ldots G_{s-k}\right)\boldsymbol{\delta} \qquad \text{large } s \tag{14}$$

Consider the excess phases at the end of s slots $\boldsymbol{\varphi}_{ex}[s]$ over many realizations and denote their *empirical* average by $\bar{\boldsymbol{\varphi}}_{ex}[s]$. Using the Law of Large Numbers (LLN) in (14), the empirical average tends to the ensemble average and we have,

$$\bar{\boldsymbol{\varphi}}_{ex}[s] = \mathbb{E}\left(\left(\mathbb{I} + \sum_{k=1}^{s-1} G_{s-1}G_{s-2}\ldots G_{s-k}\right)\boldsymbol{\delta}\right) \tag{15}$$

We can show that $\mathbb{E}\left(G_{s-1}G_{s-2}\ldots G_{s-k}\right) = \overline{\mathcal{G}}^k$ where $\overline{\mathcal{G}} = \sum_i p_i\mathcal{G}_i$ is the averaged system matrix. Using this fact and the linearity of expectation in (15), we obtain, $\overline{\boldsymbol{\varphi}}_{ex}[s] \approx \left(\mathbb{I} + \sum_{k=1}^{s-1}\overline{\mathcal{G}}^k\right)\boldsymbol{\delta}$. As before, we can excise the first eigenmode of $\overline{\mathcal{G}}$ to obtain,

$$\overline{\boldsymbol{\varphi}}_{ex}[s] \approx \left(\mathbb{I} + \sum_{k=1}^{s-1}\overline{\mathcal{G}}_{ex}^k\right)\boldsymbol{\delta} \tag{16}$$

The maximum value of the pairwise difference between neighbors' phases (or equivalently, excess phases) is given by$||\mathbf{C}\boldsymbol{\varphi}_{ex}[s]||_\infty$, where matrix multiplication by $\mathbf{C}$ corresponds to all possible pairwise differences across neighbors and $||\mathbf{x}||_\infty$ denotes the infinity norm, or maximum element, of a vector $\mathbf{x}$. The infinity norm $||\boldsymbol{v}||_\infty$ is a convex function of its argument $\boldsymbol{v}$. Therefore, we can apply Jensen's inequality to conclude that,

$$\begin{aligned}\mathbb{E}(||\mathbf{C}\boldsymbol{\varphi}_{ex}[s]||_\infty) &\geq ||\ \mathbb{E}(\mathbf{C}\boldsymbol{\varphi}_{ex}[s])||_\infty \\ &= ||\mathbf{C}\mathbb{E}(\boldsymbol{\varphi}_{ex}[s])||_\infty = ||\mathbf{C}\overline{\boldsymbol{\varphi}}_{ex}[s]||_\infty\end{aligned} \tag{17}$$

For the given set of skews $\boldsymbol{\delta}$, we recognize that $||\mathbf{C}\overline{\boldsymbol{\varphi}}_{ex}[s]||_\infty$ is the maximum pairwise phase difference for the averaged system and $\mathbb{E}(||\mathbf{C}\boldsymbol{\varphi}_{ex}[s]||_\infty)$ is the average of the maximum pairwise phase difference in the original system. Therefore, the maximum pairwise phase difference for the averaged system is a lower bound on the average of the maximum pairwise phase difference in the original system (averaging across realizations). Thus, any set of skews that are "bad" for the averaged system are guaranteed to be bad for the actual system as well.

V. Simulation Results

We now describe the models and parameters used in our simulations, followed by the results.

Topologies: We consider the *ring* topology (each node has two neighbors) and the rectangular *grid* topology (each node has four neighbors) as canonical examples of two-dimensional networks with large and small diameter, respectively.

Interference Model: We use the node exclusive interference model (two links can be active if they do not have a node in common) as an abstraction for "highly directional" networks (such as mm-wave networks [13]). For omnidirectional networks, we use the 2-hop interference model [14]: two links can be active simultaneously as long as they do not have a node in common, and the nodes involved in the two links are not neighbors.

Communication Patterns: We consider TDM schedules in which a randomly chosen maximal matching is active in each slot; a *matching* is a set of links that can be simultaneously active without interfering with one another, and a *maximal matching* is a matching to which no link can be added without interfering with one of the active links. For large networks (e.g., a 64 node grid with directional links), it is infeasible to enumerate all maximal matchings. In this case, for each link, we randomly choose 120 maximal matchings in which it participates. The overall set of maximal matchings $\mathcal{S}_{matching}$ is obtained by taking the union of the sets generated for each link.

Simulation Parameters: We fix $\beta = 0.5$ in all our simulations. Our choice of timescales in directional networks is motivated by Gigabit rate mm-wave networks: we choose the slot time $T_{slot} = 10\mu s$ and for the coarse synchronization procedure at startup, we choose $\tau_{min} = 10\mu s$ and $\tau_{max} = 300\mu s$ (see Section II). The timescales in omnidirectional settings are motivated by WiFi-style networks: we choose $T_{slot} = 10ms$ and $\tau_{min} = 10ms$ and $\tau_{max} = 300ms$. We set the maximum value of the skew ρ_{max} to 50 ppm.

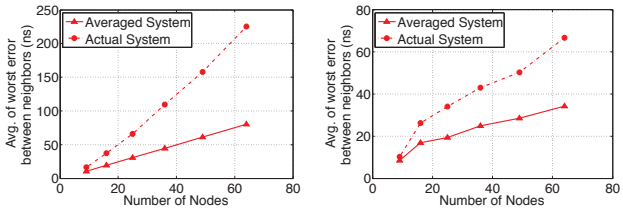
Performance Metric: Since TDM slotting overhead depends on the phase error among communicating nodes, the key performance metric is the *worst neighbor timing error,* which is the largest magnitude of phase error *between neighbors*.

Simulation Methodology: We consider directional networks and omnidirectional networks consisting of 9-64 nodes set in ring and grid topologies. For each topology, we generate a set of "bad" skews by solving linear programs that optimize the phase error on each link of the averaged system. We choose these "bad" skews to be the distribution of skews across nodes in the actual system. We then average the worst error between nodes in the actual system at the end of 3000 slots over 2000 realizations of communication patterns. The results for the directional and omnidirectional settings are shown in Figures 1 and 2 respectively and we make the following observations:

- The errors for the averaged system are a lower bound for those in the actual system, and the worst neighbor error increases with network size.
- The error grows faster with the number of nodes N for a ring topology when compared to the grid topology. For directional ring networks (Figure 1(a)), the error grows as N in the averaged system and as $N^{1.33}$ (approximately) in the actual system. For directional grid networks (Figure 1(b)), the error grows only as $N^{0.66}$ in the averaged system and as $N^{0.875}$ (approximately) in the actual system.

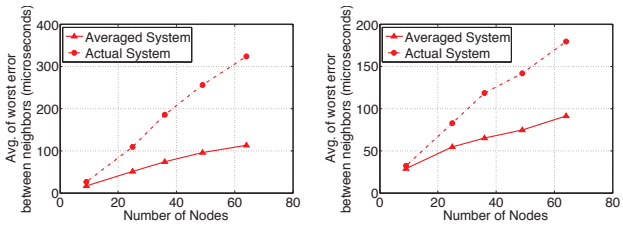
Typical numbers for directional networks: From Figure 1, we see that the error between neighbors in a 64 node network set in a ring topology can be as large as 220 ns. This is comparable in magnitude to the guard interval needed

978-1-4244-9848-2/11$26.00©2011 IEEE



(a) Ring Topology (b) Grid Topology

Fig. 1. Worst error between neighbors for the actual system and the averaged system with only phase adjustments in a *directional network*.



(a) Ring Topology (b) Grid Topology

Fig. 2. Worst error between neighbors for the actual system and the averaged system with only phase adjustments in an *omnidirectional network*. The network operates in the ONLYINTENDED mode.

to handle propagation delays. For example, the envisioned Gigabit rate outdoor mesh networks with link ranges on the order of 200 m require guard intervals of $200\text{m}/(3 \times 10^8\text{m/s}) \approx 666\text{ns}$ to handle propagation delays. Therefore, an additional synchronization error of 220 ns represents a 33% increase in the overhead. On the other hand, for small directional networks in a ring topology (say, 16 nodes) or reasonably large networks in a grid topology (36 nodes), we see that the largest phase error between neighbors will only be 40 ns. Therefore, the maximum increase in the overhead is only 6 %. To summarize, phase-only adjustments with implicit timestamps may suffice in small directional networks with linear topologies or moderately sized networks in grid topologies, but frequency adjustments are necessary for large networks, especially in linear topologies.

Recommendations for omnidirectional networks: Since the slot duration T_{slot} is longer in omnidirectional networks and the phase error scales in proportion to T_{slot}, we see from Figure 2 that the errors with phase-only adjustments are also correspondingly larger ($\sim$ 10's of μs even for small networks). However, the guard interval needed to handle propagation delays is still $\approx 1\mu s$, since the link ranges are on the order of hundreds of meters. Therefore, phase-only adjustments are not sufficient to guarantee phase synchrony on the order of the guard interval in omnidirectional networks. However, a more practical definition of the tolerable overhead is to define it as a fraction of the payload (slot size) rather than the guard interval for propagation delay. Phase-only adjustments might then suffice since the phase errors grow in proportion to T_{slot} and their *ratio* is simply a constant. Thus, here too, phase-only adjustments might suffice for small-to-moderate sized networks.

VI. CONCLUSIONS

The two novel features of the proposed synchronization algorithm, relative to prior work on network timing synchronization, is that it is implicit, leveraging ongoing communication, and that it is extremely simple, because it aims for a *sloppy* consensus on phases rather than requiring network-wide convergence. While analysis of the original system is complicated, we are able to quantify the penalty due to this sloppiness by introducing an averaged system for which the worst-case pairwise phase error between neighbors can be bounded using a linear program. Our numerical results show that the resulting overhead is tolerable for small networks. However, since the error grows with network size, for larger networks, frequency adjustments are necessary in addition to the phase updates considered here. Indeed, in work to be reported in subsequent publications, we have shown that frequency adjustments can be layered on top of phase adjustments to obtain implicit schemes for network-wide synchrony. These theoretical results motivate further investigation into translating these ideas into practice, starting with more detailed simulations that explicitly model traffic patterns, medium access control, and initial establishment of coarse timing synchronization (including estimation of propagation delays).

REFERENCES

[1] R. Mirollo and S. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM J. on Appl. Math.*, pp. 1645–1662, 1990.
[2] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proc. ACM SenSys 2005*, 2005, pp. 142–153.
[3] R. Solis, V. Borkar, and P. Kumar, "A new distributed time synchronization protocol for multihop wireless networks," in *Proc. of the 45th IEEE CDC*, 2006.
[4] L. Schenato and G. Gamba, "A distributed consensus protocol for clock synchronization in wireless sensor network," in *Proc. IEEE CDC 2007*.
[5] P. Sommer and R. Wattenhofer, "Gradient Clock Synchronization in Wireless Sensor Networks," in *Proc. ACM/IEEE IPSN*, 2009.
[6] R. Olfati-Saber and R. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.
[7] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 147–163, 2002.
[8] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings ACM SenSys 2003*. ACM New York, NY, USA, 2003, pp. 138–149.
[9] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. ACM SenSys 2004*, 2004, pp. 39–49.
[10] N. Freris and P. Kumar, "Fundamental limits on synchronization of affine clocks in networks," in *Invited paper in the 46th IEEE CDC*, 2007.
[11] W. Ren and R. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, p. 655, 2005.
[12] G. Dantzig and M. Thapa, *Linear Programming: Theory and extensions*. Springer Verlag, 2003.
[13] S. Singh, R. Mudumbai, and U. Madhow, "Distributed coordination with deaf neighbors: efficient medium access for 60 GHz mesh networks," *IEEE INFOCOM 2010*, 2010.
[14] G. Sharma, R. Mazumdar, and N. Shroff, "On the complexity of scheduling in wireless networks," in *Proc. ACM MobiCom 2006*.

978-1-4244-9848-2/11$26.00©2011 IEEE