Skewless Network Clock Synchronization Without Discontinuity: Convergence and Performance

Enrique Mallada, *Member, IEEE*, Xiaoqiao Meng, Michel Hack, Li Zhang, and Ao Tang, *Senior Member, IEEE*

***Abstract*—This paper examines synchronization of computer clocks connected via a data network and proposes a skewless algorithm to synchronize them. Unlike existing solutions, which either estimate and compensate the frequency difference (skew) among clocks or introduce offset corrections that can generate jitter and possibly even backward jumps, our solution achieves synchronization without these problems. We first analyze the convergence property of the algorithm and provide explicit necessary and sufficient conditions on the parameters to guarantee synchronization. We then study the effect of noisy measurements (jitter) and frequency drift (wander) on the offsets and synchronization frequency, and further optimize the parameter values to minimize their variance. Our study reveals a few insights, for example, we show that our algorithm can converge even in the presence of timing loops and noise, provided that there is a well-defined leader. This marks a clear contrast with current standards such as NTP and PTP, where timing loops are specifically avoided. Furthermore, timing loops can even be beneficial in our scheme as it is demonstrated that highly connected subnetworks can collectively outperform individual clients when the time source has large jitter. The results are supported by experiments running on a cluster of IBM BladeCenter servers with Linux.**

***Index Terms*—Distributed control, network clock synchronization, network time protocol, precision time protocol.**

I. Introduction

KEEPING consistent time among different nodes in a network is a fundamental requirement of many distributed applications. Nodes' internal clocks are usually not accurate enough and tend to drift apart from each other over time, generating inconsistent time values. Network clock synchronization allows these devices to correct their clocks to match a global reference of time, such as the Universal Coordinated Time (UTC), by performing time measurements through a network. For example, for the Internet, network clock synchronization has been an important subject of research, and several different protocols have been proposed [2]–[8]. These protocols are used in various applications with diverse precision requirements such as banking, communications, traffic monitoring, and security. In modern wireless cellular networks, for instance, time-sharing protocols need an accuracy of several microseconds to guarantee the efficient use of channel capacity. Another example is the recently announced Google Spanner [9], a globally distributed database, which depends on globally synchronized clocks within at most several-milliseconds drifts.

The current *de facto* standard for IP networks is the Network Time Protocol (NTP) proposed by Mills [2]. It is a low-cost, purely software-based solution whose accuracy mostly ranges from hundreds of microseconds to several milliseconds. On the other hand, IEEE 1588 (PTP) [4] gives superior performance by achieving submicrosecond or even nanosecond accuracy. However, it is relatively expensive as it requires special hardware support to achieve those accuracy levels and may not be fully compatible with legacy cluster systems.

Newer synchronization protocols have been proposed with the objective of balancing between accuracy and cost. For example, IBM Coordinated Cluster Time (CCT) [10] is able to provide better performance than NTP without additional hardware. Its success is based on a skew estimation mechanism [11] that progressively adapts the clock frequency without offset corrections. Another solution that achieves this objective is the RADclock [5], [8], which decouples skew compensation from offset corrections by decomposing the clock into a high performance difference clock for measuring time differences and a less precise absolute clock that provides UTC time.

There are two major difficulties that make the network clock synchronization problem challenging. First, the frequency of hardware clocks is sensitive to temperature and is constantly varying. Second, the latency introduced by the OS and network congestion delay results in errors in the time measurements that can be propagated through the network. Thus, most protocols introduce different ways of estimating the frequency mismatch (skew) [11], [12] and measuring the time difference (offset) [13], [14] while maintaining a simple network topology [2], [4]. This leads in particular to extensive literature on skew estimation [12], [15]–[17], which suggests that explicit skew estimation is necessary for clock synchronization.

This paper takes a different approach and shows that using skew estimation is unnecessary. We provide a simple algorithm that is able to compensate the clock skew without any explicit estimation of it. Our algorithm only uses current offset information and an exponential average of the past offsets. Thus, it neither needs to store long offset history nor perform expensive computations on them. The solution provided in this paper achieves microsecond-level accuracy without requiring

Manuscript received August 25, 2013; revised April 26, 2014; accepted July 04, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor X. Lin. A preliminary version of this paper appears in the Proceedings of the IEEE International Conference on Network Protocols (ICNP), Göttingen, Germany, October 7–10, 2013.

E. Mallada is with the Computational and Mathematical Sciences, California Institute of Technology (Caltech), Pasadena, CA 91125 USA.

X. Meng, M. Hack, and L. Zhang are with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.

A. Tang is with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853 USA (e-mail: atang@ece.cornell.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2345692

any special hardware. Since we do not explicitly estimate the skew, the implementation is simpler and more robust to noise than IBM CCT, and does not introduce offset corrections, which avoids the need of decomposing the clock into several components to reduce jitter as in RADclock.

By looking at the synchronization problem from a new angle, this paper also provides several new insights. For example, a common practice in the clock synchronization community is to avoid timing loops in the network [2, p. 6], [4, p. 16, Sec. 6.2]. This is because it is thought that timing loops can introduce instability as stated in [2]: *"Drawing from the experience of the telephone industry, which learned such lessons at considerable cost, the subnet topology\ldots must never be allowed to form a loop."* Even though, for some parameter values, loops can produce instability, we show that a set of proper parameters can guarantee convergence even in the presence of loops. Furthermore, we experimentally demonstrate in Section VI that timing loops among clients can actually help reduce the jitter of the synchronization error and are therefore desirable.

The rest of the paper is organized as follows. In Section II, we provide some background on how clocks are actually implemented in computers and how different protocols discipline them. Section III motivates and describes our algorithm together with an intuitive explanation of why it works. In Section IV, we analyze the convergence property of the algorithm and determine the set of parameter values and connectivity patterns under which synchronization is guaranteed. The parameter values that guarantee synchronization depend on the network topology, but there exists a subset of them that is independent of topology and therefore of great practical interest. The effect of noisy measurement and wander is studied in Section V, together with an optimization procedure that finds optimal parameter values. Experimental results evaluating the performance of the algorithm are presented in Section VI. We conclude in Section VII.

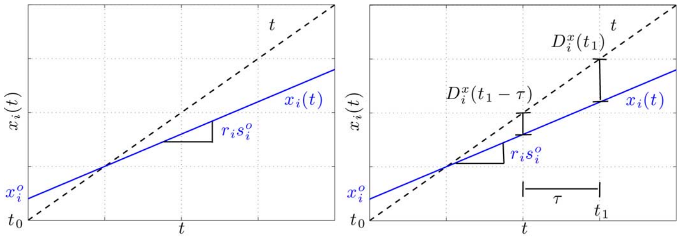
II. Computer Clocks and Synchronization

Most computer architectures keep their own estimate of time using a counter that is periodically increased by either hardware or kernel's interrupt service routines (ISRs). On Linux platforms for instance, there are usually several different clock devices that can be selected as the clock source by changing the *clocksource* kernel parameter. One particular counter that has recently been used by several clock synchronization protocols [5], [10] is the Time Stamp Counter (TSC) that counts the number of CPU cycles since the last restart of the system. For example, in the IBM BladeCenter LS21 servers, the TSC is a 64-bit counter that increments every $\delta^o = 0.416$ ns since the CPU nominal frequency $f^o = 1/\delta^o = 2399.711$ MHz.

Based on this counter, each server builds its own estimate $x_i(t)$ of the global time reference, UTC, denoted here by t. For example, if $c_i(t)$ denotes the counter's value of computer i at time t, then $x_i(t)$ can be computed using

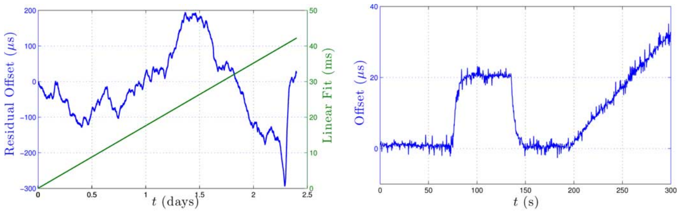
$$x_i(t) = \delta^o c_i(t) + x_i^o \tag{1}$$

where x^o is the estimate of the time when the server was turned on (t_0).



(a) (b)

Fig. 1. Time estimation and relative measurements. (a) Illustration of computer time estimate $x_i(t)$ and UTC time t. (b) Offset and relative skew measurements.



(a) (b)

Fig. 2. Comparison between two TSC counters and skew and offset corrections using *adjtimex()*. (a) Variation of the offset between two TSC counters changes on skew (r_i): The right y-axis represents the mean offset change in milliseconds, and the left y-axis the residual offset (offset minus the mean) in microseconds. (b) Example of skew and offset corrections on linux time: First a 20-μs offset is added and subtracted, and then a skew of 0.3 ppm is introduced.

Thus, synchronizing computer clocks implies correcting $x_i(t)$ in order to match t, i.e., enforcing $x_i(t) = t$. There are two difficulties on this estimation process. First, the initial time t_0 in which the counter starts is unknown. Second, the counter updating period δ_i ($\delta_i \approx \delta^0$) is usually unknown with enough precision and therefore presents a skew $r_i = \frac{x_i(t)-x_i(t_0)}{t-t_0} = \frac{\delta^0}{\delta_i}$. This is illustrated in Fig. 1(a) where $x_i(t)$ not only increases at a different rate than t, but also starts from a value different from t_0, represented by x_i^o.

In practice, $c_i(t)$ can be approximated by a real value since the time between increments is extremely small (0.416 ns) and the maximum count register value so large ($2^{64}-1$) that it would take more than 200 years to reach. Therefore, $x_i(t)$ can be described by the linear map of the global reference t, i.e.,

$$x_i(t) = r_i s_i^o (t - t_0) + x_i^o \tag{2}$$

where s_i^o is an additional skew correction implemented to compensate the skew. Equation (2) also shows that if one can set $s_i^o = 1/r_i$ and $x_i^o = t_0$, then we obtain a perfectly synchronized clock with $x_i(t) = t$.

The main problem is that not only neither t_0 nor r_i can be explicitly estimated, but also r_i varies with time as shown in Fig. 2(a). Thus, current protocols periodically update s_i^o and x_i^o in order to keep track of the changes of r_i. These updates are made using the *offset* between the current estimate $x_i(t)$ and the global time t, i.e., $D_i^x(t) = t - x_i(t)$, and the *relative frequency error* that is computed using two offset measurements separated by τ seconds, i.e.,

$$f_i^{\mathrm{err}}(t) := \frac{D_i^x(t) - D_i^x(t-\tau)}{x_i(t) - x_i(t-\tau)} = \frac{1 - r_i s_i^o}{r_i s_i^o}. \tag{3}$$

Fig. 1(b) provides an illustration of these measurements. In most protocols (see, e.g., [2], [5], and [10]), (3) goes through an additional filtering process to reduce the estimation noise. Here, we will use $f_i^{\mathrm{err}}(t_k)$ to denote either the measurement obtained using (3) or a filtered version of it. To understand the differences between current protocols, we first rewrite the evolution of $x_i(t)$ based only on the time instants t_k in which the clock corrections are performed. We allow the skew correction s_i^o to vary over time, i.e., $s_i(t_k)$, and write $x_i(t_{k+1})$ as a function of $x_i(t_k)$. Thus, we obtain

$$x_i(t_{k+1}) = x_i(t_k) + \tau r_i s_i(t_k) + u_i^x(t_k) \tag{4a}$$

$$s_i(t_{k+1}) = s_i(t_k) + u_i^s(t_k) \tag{4b}$$

where $\tau = t_{k+1} - t_k$ is the time elapsed between adaptations, also known as poll interval [2]. The values $u_i^x(t_k)$ and $u_i^s(t_k)$ represent two different types of corrections that a given protocol chooses to do at time t_k and are usually implemented within the interval (t_k, t_{k+1}). $u_i^x(t_k)$ is usually referred to as *offset correction* and $u_i^s(t_k)$ as *skew correction*.[1] See Fig. 2(b) for an illustration of their effect on the linux time.

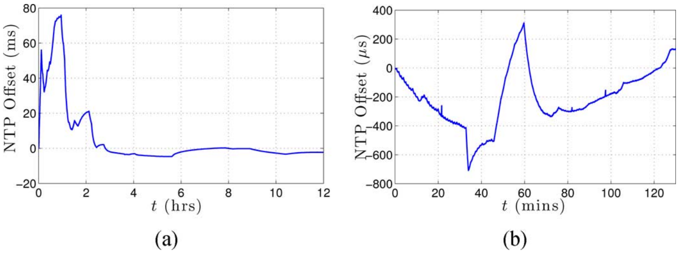
Remark 1: One of the implicit assumptions of the model (4) is that we require every server to update their clocks simultaneously at time instances $\{t_k\}$. This may seem unrealistic since its implementation would require sharing a common time reference, which is the whole purpose of the algorithm. However, the analysis presented in Section IV can be extended for pseudo-synchronous implementations as proposed in [18] where each node measures the offset with their neighbors and updates whenever $x_i(t) = kT$.

We now proceed to summarize the different types of adaptations implemented by current protocols. To simplify the comparison, we assume that each server can connect directly to the source of UTC time (t). This assumption will be dropped in Section III after we describe our solution. The main differences between current protocols lie on whether they use offset corrections, skew corrections, or both and whether they update using offset values $D_i^x(t_k)$, relative frequency errors $f_i^{\mathrm{err}}(t_k)$, or both.

A. Offset Corrections

These corrections consist in keeping the skew fixed and periodically introducing time changes of size $u_i^x(t_k) = \kappa_1 D_i^x(t_k)$ or $u_i^x(t_k) = \kappa_1 D_i^x(t_k) + \kappa_2 f_i^{\mathrm{err}}(t_k)$ where $\kappa_1, \kappa_2 > 0$. They are used by NTPv3 [19] and NTPv4 [2], respectively, under ordinary conditions.

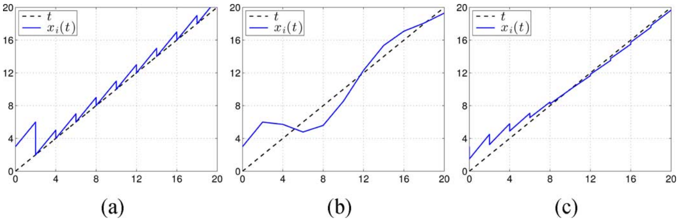
These protocols have in general a slow initialization period as shown in Fig. 3(a). This is because the algorithm must first obtain a very accurate estimate of the initial frequency error $f_i^{\mathrm{err}}(t_0)$. Furthermore, these updates usually generate nonsmooth time evolutions as in Figs. 3(b) and 4(a), and should be done carefully since they might introduce backward jumps $(x_i(t_{k+1}) < x_i(t_k))$, which can be problematic for some applications.



(a)

(b)

Fig. 3. Evolution of NTP time using TSC as reference. (a) NTP initialization period. (b) NTP in normal regime.



(a)

(b)

(c)

Fig. 4. Current protocols adaptation. (a) Offset corrections. (b) Skew corrections. (c) Offset and skew corrections.

B. Skew Corrections

Another alternative that avoids using steep changes in time is proposed by the IBM CCT solution [10]. This alternative does not introduce any offset correction, i.e., $u_i^x(t_k) = 0$, and updates the skew $s_i(t_k)$ by $u_i^s(t_k) = \kappa_1 D_i^x(t_k) + \kappa_2 f_i^{\mathrm{err}}(t_k)$.

The behavior of this algorithm is shown in Fig. 4(b). In [20], it was shown for a slightly modified version of it (used $r_i s_i(t_k) f_i^{\mathrm{err}}(t_k)$ instead of $f_i^{\mathrm{err}}(t_k)$), the algorithm can achieve synchronization for very diverse network architectures.

However, the estimation of $f_i^{\mathrm{err}}(t_k)$ is nontrivial as it is constantly changing with subsequent updates of $s_i(t_k)$ and it usually involves sophisticated computations [11], [12].

C. Skew and Offset Corrections

This type of correction allows dependence on only offset information $D_i^x(t_k)$ as input to $u_i^x(t_k)$ and $u_i^s(t_k)$. For instance, in [6], the updates $u_i^x(t_k) = \kappa_1 D_i^x(t_k)$ and $u_i^s(t_k) = \kappa_2 D_i^x(t_k)$ were proposed. This option allows the system to achieve synchronization without any skew estimation. However, the cost of achieving it is introducing offset corrections in $x_i(t)$ as shown in Fig. 4(c). Therefore, it suffers from the same problems discussed in Section II-B.

Another alternative that falls in into this category is the RADclock [5]. In this solution, the offset correction $u_i^x(t_k)$ is an exponential average of the past offsets, and the skew compensation $u_i^s(t_k)$ is a filtered version of $f_i^{\mathrm{err}}(t_k)$. The exponential average of offsets and filter stage in $f_i^{\mathrm{err}}(t_k)$ allows this solution to mitigate the jumps and become more robust to jitter. However, it does not necessarily prevent backward jumps unless the offset corrections are smaller than the precision of the clock.

III. Continuous Skewless Synchronization

We now present an algorithm that overcomes the limitations of the solutions described in Section II. In other words, our solution has the following two properties.

[1] These corrections can be implemented in Linux OS using the *adjtimex()* interface to update the system clock or by maintaining a virtual version of $x_i(t)$ and directly applying the corrections to it, as in IBM CCT [10] and RADclock [5]. The latter gives more control on how the corrections are implemented since it does not depend on kernel's routines.

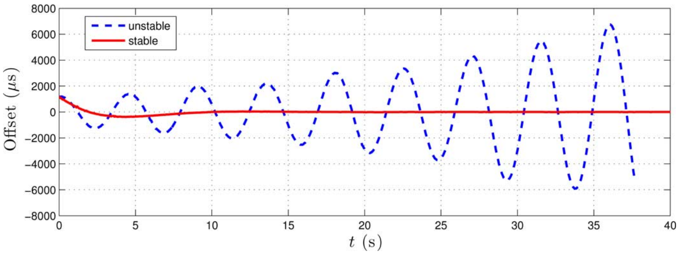


Fig. 5. Unstable clock steering using only offset information (5) and stable clock steering based on exponential average compensation (7).

1) Continuity: The protocol does not introduce steep changes on the time value, i.e., $u_i^x(t_k) \equiv 0$.
2) Skew independence: The protocol does not use skew information $f_i^{\rm err}(t_k)$ as input.

A solution with these properties will therefore prevent unnecessary offset corrections that produce jitter and will be more robust to noise by avoiding skew estimation. After describing and motivating our algorithm, we show how the updating rule can be implemented in the context of a network environment.

The motivation behind the proposed solution comes from trying to compensate the problem that arises when one tries to naively impose properties 1) and 2), i.e., using

$$u_i^x(t_k) = 0 \quad \text{and} \quad u_i^s(t_k) = \kappa_1 D_i^x(t_k). \tag{5}$$

Fig. 5 shows that this type of clock correction is unstable; the offset $D_i^x(t_k)$ of the slave clock oscillates with an exponentially increasing amplitude. The oscillations in Fig. 5 arise due to the fundamental limitations of using offset to update frequency. This is better seen in the continuous-time version of the system (4) with (5), i.e.,

$$\dot{x}_i(t) = r_i s_i(t) \quad \text{and} \quad \dot{s}_i(t) = \kappa_1 D_i^x(t)$$

where $\dot{x}(t) = \frac{d}{dt}x(t)$. If we consider the offset $D_i^x = t - x_i(t)$ as the system state, then we have

$$\dot{D}_i^x = 1 - r_i s_i \quad \text{and} \quad \ddot{D}_i^x = -\kappa_1 r_i D_i^x$$

with $\ddot{x}(t) = \frac{d^2}{dt^2}x(t)$. This is analogous to a spring mass system without friction. Thus, it has two purely imaginary eigenvalues that generate sustained oscillations; see [7] and [21] for similar examples.[2] One way to damp these oscillations in the spring-mass case is by adding *friction*. This implies adding a term that includes a frequency mismatch $f_i^{\rm err}(t)$ in our system, which is equivalent to the protocols of Section II-C, and is therefore undesired.

However, there are other ways to damp these oscillations using passivity-based techniques from control theory [22]. The basic idea is to introduce an additional state y_i that generates the desired *friction* to damp the oscillations.

Inspired by [22], we consider the exponentially weighted moving average of the offset

$$y_i(t_{k+1}) = pD_i^x(t_k) + (1-p)y_i(t_k) \tag{6}$$

and update $x_i(t_k)$ and $s_i(t_k)$ using

$$u_i^x(t_k) = 0 \quad \text{and} \quad u_i^s(t_k) = \kappa_1 D^x(t_k) - \kappa_2 y(t_k). \tag{7}$$

Fig. 5 shows how the proposed strategy is able to compensate the oscillations without needing to estimate the value of $f_i^{\rm err}(t_k)$. The stability of the algorithm will depend on how κ_1, κ_2, and p are chosen. A detailed specification of these values is given in Section IV-B.

Finally, since we are interested in studying the effect of timing loops, we move away from the client–server configuration implicitly assumed in Section II and allow mutual or cyclic interactions among nodes. The interactions between different nodes is described by a graph $G(V,E)$, where V represents the set of n nodes ($i \in V$) and E the set of *directed* edges ij; $ij \in E$ means node i can measure its offset with respect to j, $D_{ij}^x(t_k) = x_j(t_k) - x_i(t_k)$.

Within this context, a natural extension of (6) and (7) is to substitute $D_i^x(t_k)$ with the weighted average of i's neighbors offsets. Thus, we propose the following algorithm to update the clocks in the network.

Algorithm 1 (Alg1): For each computer node i in the network, perform the following actions.

— Compute the time offsets $(D_{ij}^x(t_k))$ from i to every neighbor j at time t_k.

— Update the skew $s_i(t_{k+1})$ and the moving average $y_i(t_{k+1})$ at time t_{k+1} according to

$$x_i(t_{k+1}) = x_i(t_k) + \tau_k r_i s_i(t_k) \tag{8a}$$

$$s_i(t_{k+1}) = s_i(t_k) + \kappa_1 \sum_{j\in\mathcal{N}_i} \alpha_{ij} D_{ij}^x(t_k) - \kappa_2 y_i(t_k) \tag{8b}$$

$$y_i(t_{k+1}) = p \sum_{j\in\mathcal{N}_i} \alpha_{ij} D_{ij}^x(t_k) + (1-p)y_i(t_k) \tag{8c}$$

where $\mathcal{N}_i$ represents the set of neighbors of i and the weights α_{ij} are positive.

Equation (8) can be interpreted as a discrete-time *second-order consensus* algorithm with an additional smoothing in which, besides using position information (time estimates $x_i(t_k)$), we use a smoothed version of the position errors $(y_i(t_k))$ to control speed $(s_i(t_k))$. Consensus algorithms have been a subject of intensive research since the seminar work of Jadbabaie *et al.* [23]; see ,e.g., [22] and references therein. In particular, application of consensus ideas to computer clock synchronization can be found in [6] (second-order consensus) and [18] (first-order consensus). Thus, the analysis presented in this paper also contributes to this rich literature by characterizing convergence of discrete-time consensus algorithms.

When using our algorithm, many servers can affect the final frequency of the system. Thus, when the system synchronizes, we have

$$x_i(t_k) \to x^{\rm ref}(t_k) := r^*(t_k - t_0) + x^* \qquad i \in V. \tag{9}$$

r^* and x^* are possibly different from their ideal values 1 and t_0. Their final values depend on the initial condition of all different clocks as well as the topology, which we assume to be a connected graph in this paper.

[2] In the discrete-time system, the oscillations increase in amplitude since there is a delay between the time the offset is measured t_k and the time the update is made t_{k+1}, which makes the system unstable.

1) Differences With RADclock: Although (8) seems to be similar to RADclock [5], there are some key differences that affect their behavior.

1) Even though both solutions used an exponentially weighted offset estimate, our filtering (8c) does not depend on the estimated offset error as in [5]. Moreover, while RADclock uses it to make offset corrections (changing $u_i^x(t_k)$), we use our weighted offset measurement $y_i(t)$ to make skew correction (changing $u_i^s(t_k)$). Therefore, neither the measurement itself nor its use are the same.
2) RADclock explicitly uses offset measurements to introduce correction on the offset ($u_i^x(t_k)$) and an estimation of the skew to compensate it ($u_i^s(t_k)$). Our algorithm only compensates the skew by using the last measured offsets $D_{ij}^x(t_k)$ and our filtered offset measurement $y_i(t_k)$. Thus, we have neither explicit estimation of the skew nor explicit compensation of the offset, which makes synchronization rather unintuitive.

Notation: We use $\mathbf{0}_{m\times n}$ ($\mathbf{1}_{m\times n}$) to denote the matrices of all zeros (ones) within $\mathbb{R}^{m\times n}$ and $\mathbf{0}_n$ ($\mathbf{1}_n$) to denote the column vectors of appropriate dimensions. $I_n \in \mathbb{R}^{n\times n}$ represents the identity matrix. Given a matrix $A \in \mathbb{R}^{n\times n}$ with Jordan normal form $A = PJP^{-1}$, let $n_A \leq n$ denote the total number of Jordan blocks J_l with $l \in \mathcal{I}(A) := \{1, \ldots, n_A\} = |\mu_1(A)|$. We use $\mu_l(A), l \in \{1, \ldots, n\}$ or just $\mu(A)$ to denote the eigenvalues of A, and order them decreasingly $|\mu_1(A)| \geq \cdots \geq |\mu_n(A)|$. The function $\rho(A)$ is the spectral radius of A or equivalently the largest absolute value of its eigenvalues $\rho(A) = \max_{l\in\mathcal{I}(A)} |\mu_l(A)|$. Finally, A^T is the transpose of A, A_{ij} is the element of the ith row and jth column of A, and a_i is the ith element of the column vector a, i.e., $a = [a_i]^T$.

IV. Convergence Analysis

We now analyze the asymptotic behavior of system (8) and provide a necessary and sufficient condition on the parameter values that guarantee its convergence to (9). Throughout this section, we shall assume that the internal skew r_i of each clock is constant and that the offset measurements $D_{ij}^x(t_k)$ can be obtained without incurring in any error. These assumptions will be relaxed in Section V.

The key insight of our analysis comes from decomposing the system (12) into two complementary systems that keep track of two different physical properties. In particular, we will use the scalars

$$\tilde{x}(t_k) := \gamma\sum_{i=1}^{n}\frac{\xi_i}{r_i}x_i(t_k) \quad \tilde{s}(t_k) := \gamma\sum_{i=1}^{n}\xi_i s_i(t_k)$$
$$\tilde{y}(t_k) := \gamma\sum_{i=1}^{n}\xi_i y_i(t_k) \tag{10}$$

to track the average behavior of the system and

$$\delta x_i(t_k) := x_i(t_k) - \tilde{x}(t_k) \quad \delta s_i(t_k) := s_i(t_k) - \frac{\tilde{s}(t_k)}{r_i}$$
$$\delta y_i(t) := y_i(t) - \frac{\tilde{y}(t_k)}{r_i} \tag{11}$$

to track how each individual clock deviates from the collective mean. Here, $\xi = [\xi_i]^T$ is the normalized ($\sum_i \xi_i = 1$) left eigenvector of the zero eigenvalue of the Laplacian matrix $L \in \mathbb{R}^n$ associated with $G(V, E)$, i.e.,

$$L_{ii} = \alpha_{ii} := \sum_{j\in\mathcal{N}_i}\alpha_{ij} \quad \text{and} \quad L_{ij} = \begin{cases} -\alpha_{ij}, & \text{if } ij \in E, \\ 0, & \text{otherwise} \end{cases}$$

and γ is the ξ_i-weighted harmonic mean of r_i, i.e., $\frac{1}{\gamma} = \mathbf{1}_n^T R^{-1}\xi = \sum_{i=1}^{n}\frac{\xi_i}{r_i}$. While in general ξ may not be unique, it becomes unique when $G(E, V)$ is connected.

It will be more convenient sometimes to use a vector form representation of (8) given by

$$z_{k+1} = Az_k \tag{12}$$

where $z_k := [x(t_k)^T s(t_k)^T y(t_k)^T]^T \in \mathbb{R}^{3n}$

$$A := \begin{bmatrix} I_n & \tau R & \mathbf{0}_{n\times n} \\ -\kappa_1 L & I_n & -\kappa_2 I_n \\ p(-L) & \mathbf{0}_{n\times n} & (1-p)I_n \end{bmatrix} \in \mathbb{R}^{3n\times 3n}.$$

$R \in \mathbb{R}^{n\times n}$ is the diagonal matrix with elements r_i. Similarly, we can express the evolution of $\tilde{z}_k := [\tilde{x}(t_k)\ \tilde{s}(t_k)\ \tilde{y}(t_k)]^T$ and $\delta z_k := [\delta x(t_k)^T\ \delta s(t_k)^T\ \delta y(t_k)^T]^T$ using

$$\delta z_{k+1} = \hat{A}\delta z_k \quad \text{and} \quad \tilde{z}_{k+1} = \tilde{A}\tilde{z}_k \tag{13}$$

where $\hat{A} := NA$, $N := \text{blockdiag}(N_1, N_2, N_2)$, $N_1 := I_n - \gamma\mathbf{1}_n\xi^T R^{-1}$, $N_2 := I_n - \gamma R^{-1}\mathbf{1}_n\xi^T$ and

$$\tilde{A} := \begin{bmatrix} 1 & \tau & 0 \\ 0 & 1 & -\kappa_2 \\ 0 & 0 & 1-p \end{bmatrix}. \tag{14}$$

The convergence analysis of this section is done in two stages. First, we provide necessary and sufficient conditions for synchronization in terms of the eigenvalues of A (Section IV-A), and then use Hermite–Biehler Theorem [24] to relate these eigenvalues with the parameter values that can be directly used in practice (Section IV-B). The proof details can be found in Appendixes A–C.

A. Asymptotic Behavior

We start by studying the asymptotic behavior of (12). That is, we are interested in finding under what conditions the series of elements $\{x_i(t_k)\}$ converge to (9) as t_k goes to infinity.

We will show that we can study (12) by looking at the evolution of (13). In particular, we will show that (9) is equivalent to

$$\delta x(t_k) \to \mathbf{0}_n \quad \delta s(t_k) \to \mathbf{0}_n \quad \delta y(t_k) \to \mathbf{0}_n \tag{15a}$$
$$\tilde{x}(t_k) \to x^{\text{ref}}(t_k) \quad \tilde{s}(t_k) \to r^* \quad \tilde{y}(t_k) \to 0. \tag{15b}$$

Consider the Jordan normal form [25] of

$$A = PJP^{-1} := [\zeta_1 \quad \ldots \quad \zeta_{3n}]\, J\, [\eta_1 \quad \ldots \quad \eta_{3n}]^T \tag{16}$$

where $J = \text{blockdiag}(J_l)_{l\in\mathcal{I}(A)}$, ζ_i and η_i are the right and left generalized eigenvectors of A such that

$$\zeta_i^T\eta_j = \begin{cases} 1, & \text{if } j = i \\ 0, & \text{otherwise.} \end{cases}$$

The following lemmas will allow us to connect the behavior of (13) with (12).

Lemma 1 (Eigenvalues of A and Multiplicity of $\mu(A) = 1$): A has an eigenvalue $\mu(A) = 1$ with multiplicity 2 if and only if the graph $G(V, E)$ is connected, $\kappa_1 \neq \kappa_2$ and $p > 0$.

Furthermore, $\mu_l(A)$ are the roots of

$$g_l(\lambda) = (\lambda - 1)^2(\lambda - 1 + p) + [(\lambda - 1)\kappa_1 + p(\kappa_1 - \kappa_2)]\nu_l \quad (17)$$

where $\nu_l = \mu_l(\tau LR)$ and satisfies

$$\nu_n = 0 < |\nu_l| \qquad \text{for } l \in \{1, \ldots, n-1\}. \quad (18)$$

Lemma 2 (Jordan Chains Properties): Under the conditions of Lemma 1, the right and left Jordan chains, (ζ_1, ζ_2) and (η_2, η_1) respectively, associated with $\mu(A) = 1$ and the eigenvectors ζ_3 and η_3 associated with $\mu(A) = 1 - p$ are given by

$$[\zeta_1 \; \zeta_2 \; \zeta_3] = \begin{bmatrix} \mathbf{1}_n & \mathbf{1}_n & -\frac{\tau\kappa_2}{p^2}\mathbf{1}_n \\ \mathbf{0}_n & \frac{(R^{-1}\mathbf{1}_n)}{\tau} & \frac{\kappa_2}{p}R^{-1}\mathbf{1}_n \\ \mathbf{0}_n & \mathbf{0}_n & R^{-1}\mathbf{1}_n \end{bmatrix} \quad (19)$$

$$[\eta_1 \; \eta_2 \; \eta_3] = \gamma \begin{bmatrix} R^{-1}\xi & \mathbf{0}_n & \mathbf{0}_n \\ -\tau\xi & \tau\xi & \mathbf{0}_n \\ \tau\kappa_2(\frac{1}{p} + \frac{1}{p^2})\xi & -\tau\frac{\kappa_2}{p}\xi & \xi \end{bmatrix}. \quad (20)$$

Moreover, given $\zeta_l = [x_l^T \;\; s_l^T \;\; y_l^T]^T$, with $l > 3$, then the following conditions must be satisfied:

$$\mathbf{1}_n\xi^T R^{-1}x_l = \mathbf{0}_n \quad \mathbf{1}_n\xi^T s_l = \mathbf{0}_n \quad \mathbf{1}_n\xi^T y_l = \mathbf{0}_n. \quad (21)$$

The proofs of Lemmas 1 and 2 can be found in Appendices A-A and A-B.

Lemmas 1 and 2 also provide further information of the structure of J in (16). That is, $J = \operatorname{blockdiag}(\hat{J}_1, \hat{J}_2)$ where

$$\hat{J}_1 := \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1-p \end{bmatrix}. \quad (22)$$

Moreover, direct application of (21) shows that $N = P[\operatorname{blockdiag}(\mathbf{0}_{3\times 3}, I_{3(n-1)})]P^{-1}$, which implies that A and N have the same eigenvectors and therefore

$$\hat{A} = NA = P[\operatorname{blockdiag}(\mathbf{0}_{3\times 3}, \hat{J}_2)]P^{-1}. \quad (23)$$

Similarly, it is easy to see that the Jordan normal form of $\tilde{A}$ is given by

$$\tilde{A} = \tilde{P}\hat{J}_1\tilde{P}^{-1}. \quad (24)$$

Therefore, under the conditions of Lemmas 1 and 2, δz_k and $\tilde{z}_k$ capture the behavior of a complementary set of eigenvalues of A. We are now ready to state our main convergence result.

Theorem 1 (Convergence): The following three statements are equivalent.

1) The graph $G(V, E)$ is connected, $\kappa_1 \neq \kappa_2$, $2 > p > 0$ and $\rho(\hat{J}_2) < 1$.
2) Condition (15) is satisfied.
3) The algorithm (12) achieves synchronization, i.e., (9) holds.

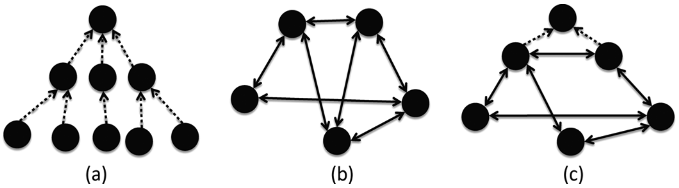


Fig. 6. Graphs with real eigenvalue Laplacians: (a) tree; (b) undirected; (c) undirected with leader.

Moreover, whenever the system synchronizes, we have

$$x^* = \gamma \sum_{i=1}^{n} \xi_i \left(\frac{1}{r_i} x_i(t_0) + \tau \frac{\kappa_2}{p^2} y_i(t_0) \right) \quad (25a)$$

$$r^* = \gamma \sum_{i=1}^{n} \xi_i (s_i(t_0) - \frac{\kappa_2}{p} y_i(t_0)). \quad (25b)$$

The proof of Theorem 1 can be found in Appendix B. Theorem 1 provides an analytical tool to understand the influence of the different nodes of the graph in the final offset x^* and frequency r^*. For example, suppose that we know that node 1 has perfect knowledge of its own frequency (r_1) and the UTC time at $t = t_0$ ($x_1(t_0) = t_0$), and configure the network such that node 1 is the unique leader like the top node in Fig. 6(a) and (c). It is easy to show that $\xi_1 = 1$ and $\xi_i = 0 \; \forall i \neq 1$. Then, using (25a)–(25b) and the definition of γ, we can see that $\gamma = r_1$ and

$$x^* = x_1(t_0) + r_1\tau\frac{\kappa_2}{p^2}y_1(t_0) \quad \text{and} \quad r^* = r_1 s_1(t_0) - \frac{r_1\kappa_2}{p}y_1(t_0).$$

However, since node 1 knows r_1 and t_0, it can choose $x_1(t_0) = t_0$, $s_1(t_0) = \frac{1}{r_1}$, and $y_1(t_0) = 0$. Thus, we obtain $x^* = t_0$ and $r^* = 1$, which implies by (9) that every node in the network will end up with $x_i(t) = t$. In other words, Theorem 1 allows us to understand how the information propagates and how we can guarantee that every server will converge to the desired time. Notice that the initial condition used for server 1 is equivalent to assuming that server 1 is a reliable source of UTC like an atomic clock for instance.

B. Necessary and Sufficient Conditions for Synchronization

We now provide necessary and sufficient conditions in terms of explicit parameter values (κ_1, κ_2 , τ and p) for Theorem 1 to hold. We will restrict our attention to graphs that have Laplacian matrices with real eigenvalues. This includes, for example, trees [Fig. 6(a)], symmetric graphs with $\alpha_{ij} = \alpha_{ji}$ [Fig. 6(b)], and symmetric graphs with a leader [Fig. 6(c)].

The proof consists on studying the Schur stability of $g_l(\lambda)$ and has several steps. We first perform a change of variable that maps the unit circle onto the left half-plane. This transforms the problem of studying the Schur stability into a Hurwitz stability problem that is solved using Hermite–Biehler Theorem, which says the following: Given the polynomial $P(s) = a_n s^n + \cdots + a_0$, let $P^r(\omega)$ and $P^i(\omega)$ be the real and imaginary part of $P(j\omega)$, i.e., $P(j\omega) = P^r(\omega) + jP^i(\omega)$. Then, $P(s)$ is a Hurwitz polynomial if and only if:

1) $a_n a_{n-1} > 0$; and
2) the zeros of $P^r(\omega)$ and $P^i(\omega)$ are all simple and real and interlace as ω runs from $-\infty$ to $+\infty$.

Theorem 2 (Parameter Values for Synchronization): Consider a connected graph $G(V,E)$ with real eigenvalue Laplacian matrix L. Then, the system (12) achieves synchronization if and only if:

(i) $|1-p|<1$ or equivalently $2>p>0$;

(ii) $\frac{2\kappa_1}{3p}>\kappa_1-\kappa_2>0$;

(iii) $\tau<\frac{p(\kappa_2-p(\kappa_1-\kappa_2))}{\mu_{\max}(\kappa_1-p(\kappa_1-\kappa_2))^2}$;

where $\mu_{\max}$ is the largest eigenvalue of LR.

The proof of Theorem 2 can be found in Appendix C. Note that although $\mu_{\max}$ depends on r_i, which is in general unknown, it is easy to show that $\mu_l(LR)\le\hat{r}_{\max}\mu_l(L)$, where $\hat{r}_{\max}$ is an upper bound of the maximum rate deviation r_i. Furthermore, using Greshgorin's circle theorem, it is easy to show that $\mu_{\max}(L)\le 2\alpha_{\max}:=2\max_i\alpha_{ii}$. Therefore, if we set

$$\tau<\frac{p(\kappa_2-\delta\kappa p)}{2\alpha_{\max}\hat{r}_{\max}(\kappa_1-\delta\kappa p)^2} \tag{26}$$

convergence is guaranteed for *every connected graph* with real eigenvalues.

V. Network Delays and Clock Wander

In Section IV, we showed that in the absence of network delays and clock wander, the system was able to achieve synchronization on a wide variety of communication topologies. In other words, we assumed the internal clock skew r_i was fixed and that each computer could measure its offset with a neighbor $D^x_{ij}(t_k)$ without incurring in any error. We now study the behavior of our system when such assumptions are no longer true. We will model both, network delays and clock drifts using noise processes.

Network Delays: Since our algorithm does not perform skew estimation, the network errors only affect the offset measurements $D^x_{ij}(t_k)$ in (8). We use $g^w_{ij}w_{ij}(t_k)$ to denote the error incurred in estimating the offset between nodes i and j at time t_k, i.e., we replace $D^x_{ij}(t_k)$ with $D^x_{ij}(t_k)+g^w_{ij}w_{ij}(t_k)$ in (8). This can be produced for instance by a congested connection between the two different nodes or due to path delay asymmetries [5]. We assume that $w_{ij}(t_k)$ has stationary mean $E\left[w_{ij}(t_k)\right]=\bar{w}_{ij}\ \forall t_k$ and unit variance $E[(w_{ij}(t_k)-\bar{w}_{ij})^2]=1$ and use g^w_{ij} to weigh the different connections.

Clock Wander: We model the clock wander as a stochastic input to the clock skew adaptation. That is, instead of (4b), $s_i(t_k)$ changes according to

$$s_i(t_{k+1})=s_i(t_k)+u^s_i(t_k)+g^d_i d_i(t_k) \tag{27}$$

where $d_i(t_k)$ is a random variable with zero mean $E[d_i(t_k)]=0$ and unit variance $E[d_i(t_k)^2]=1$ and g^d_i a positive scalar use to model clock heterogeneity. Equation (27) can be derived from a linear approximation of (4) with a time-varying internal skew $r_i(t_k)$ driven by an auto regressive process [12], i.e., $r_i(t_{k+1})=(1-q_i)r_i+q_ir_i(t_k)+g^d_id_i(t_k)$. We omit the details of the derivation due to space constraints. This motivates the study of the stochastic process

$$z_{k+1}=Az_k+Be_k, v_k=Cz_k \tag{28}$$

where $e_k=[w^T_k\ d^T_k]^T$, $B=[B_w\ B_d]$ with

$$B_w=\begin{bmatrix}\mathbf{0}_{n\times m}\\ -\kappa_1B^-_G\mathrm{diag}[\alpha_{ij}g^w_{ij}]\\ -pB^-_G\mathrm{diag}[\alpha_{ij}g^w_{ij}]\end{bmatrix}\quad B_d=\begin{bmatrix}\mathbf{0}_{n\times n}\\ \mathrm{diag}[\beta_ig^d_i]\\ \mathbf{0}_{n\times n}\end{bmatrix} \tag{29}$$

$B^-_G=\min\{B_G,\mathbf{0}_{n\times m}\}$ and B_G being the incidence matrix of $G(V,E)$[3] and $w_k=[w_{ij}(t_k)]^T$. The matrix C maps the system state z_k to the performance metric v_k and will be specified in Section VI.

One interesting difference between network delays ($w_{ij}(t_k)$) and wander ($d_i(t_k)$) is that in order to obtain good performance, the algorithm should reject the noise from network delays $w_{ij}(t_k)$, but compensate the skew fast enough to follow $d_i(t_k)$.

In the remainder of this section, we first study the effect of biased network noise ($\bar{w}_{ij}\neq 0$) in the asymptotic frequency of the system and time offsets. In particular, we show that for arbitrarily distributed noise with stationary mean, the system's frequency tends to constantly drift unless there is a well-defined leader in the topology. We then proceed to study how the parameters and network topology affect the systems performance, which is represented by the output signal v_k of the stochastic process.

A. Frequency Drift and Time Offsets

Here, we concentrate on studying the evolution of the first moment of the stochastic process (28). That is, we want to understand how $E[z_k]$ evolves as $k\to+\infty$. This is equivalent to studying (28) in the case when the noise input e_k is constant $e_k=\bar{e}=[\bar{d}^T\ \bar{w}^T]^T$, where $\bar{d}=\mathbf{0}_n$ by definition.

Therefore, we will focus on understanding the effect of a constant input $\bar{e}$ on (28). Again, we will use $\tilde{z}_k$ to understand how $\bar{e}$ affects the collective behavior of the clocks and δz_k to understand how each individual clock deviates from the collective. The next theorem summarizes the effect of nonzero mean error on the collective behavior of the system.

Theorem 3 (Frequency Drift): In the presence of noise and under the condition of Theorem 1, the collective frequency $\tilde{s}(t_k)$ will constantly drift away from its mean with probability one (in the set of possible $\bar{w}$), unless the graph $G(V,E)$ has a *unique leader*[4]. Whenever $G(V,E)$ does have a leader, the mean frequency r^* is given by (25b).

The proof of Theorem 3 can be found in Appendix D.

Remark 2: Theorem 3 provides a precise characterization of how network delays and loops can produce instabilities. Similar results can be obtained for any protocol that controls clock speeds based on neighbors information. Therefore, this theorem shows that current industry practice is conservative and allows us to explore a wider set of topologies with timing loops (provided that such loops avoid the leader).

We now show how the deviations δz_k are affected by $\bar{e}$.

Theorem 4 (Time Offsets): Under the conditions of Theorem 1 and graph $G(V,E)$ with unique leader, the deviations δz_k converge to δz^* given by

$$\delta x^*=N_1L^\dagger\delta w\quad \delta s^*=\mathbf{0}_n\quad \delta y^*=\mathbf{0}_n$$

where $L^\dagger$ is the pseudo inverse of L and $\delta w:=-N_2B^-_G\mathrm{diag}[\alpha_{ij}g^w_{ij}]\bar{w}$.

The proof can be found in Appendix D.

[3] Notice that using this definition, we have $L=B^-_G\mathrm{diag}[\alpha_{ij}]B^T_G$.

[4] A leader is a node to whom every other node can reach through a directed path.

B. $\mathcal{H}_2$ Performance Optimization

We now proceed to study the effect of noisy measurements and wander on the output standard deviation of the system ($\|v_k\|_2$) when the input e_k is white noise ($E[e_k e_l^T] = I_{m+n}\delta(l-k)$). In other words, we seek to find parameter values that minimize

$$f(\kappa_1, \kappa_2, p, \alpha_{ij}) = \|v_k\|_2 = \sqrt{E\left[\lim_{N\to+\infty} \frac{1}{N}\sum_{k=0}^{N-1} v_k^T v_k\right]}.$$

Since in practice we want to avoid any frequency drift introduced by the noise, we will consider only topologies with a well-defined leader. Thus, all the randomness of the system is concentrated in δz_k, and we limit to study the stochastic process

$$\delta z_{k+1} = \hat{A}\delta z + \hat{B}e_k \quad v_k = \hat{C}\delta z_k$$

where $\hat{A} = NA$, $\hat{B} = NB$, and $\hat{C} = C$.

This optimization problem is standard in the control theory community, and it can be shown to be equivalent to

$$\min_{X,\kappa_1,\kappa_2,p,\alpha_{ij}} \; f(\kappa_1, \kappa_2, p, \alpha_{ij}) \tag{30a}$$

$$\text{subject to} \quad \rho(\hat{A}) \le \rho^* \tag{30b}$$

$$X = \hat{A}^T X \hat{A} + \hat{C}^T \hat{C} \tag{30c}$$

where $f(\kappa_1, \kappa_2, p, \alpha_{ij}) := \sqrt{\text{trace}[X\hat{B}\hat{B}^T]}$, $\hat{A}$ is a function of $(\kappa_1, \kappa_2, p, \alpha_{ij})$ and $\rho^* < 1$. The constrain (30b) has been added in order to maintain the stability of $\hat{A}$.

While it is not in general easy to find the global minimum of (30), there has been intensive research in designing optimization algorithms that find local minimums of the $\mathcal{H}_2$ norm of continuous-time [26] and discrete-time [27] systems. In this work, we solve (30) using a discrete-time version of the package Hifoo [26], [28] known as Hifood [29]. Several adaptations were needed to use Hifood to solve (30). The details of these changes are documented in Appendix E.

The output of this optimization problem will be used in Experiment 6 of Section VI to demonstrate that the standard belief that "clock precision degrades with the number of hops" is not necessarily true.

VI. Experiments

To test our solution and analysis, we implement an asynchronous version of Algorithm 1 (Alg1) in C using the IBM CCT solution as our code base. Each server issues a thread to handle the connection with each neighbor. Every τ seconds (using OS time), each client takes offset measurements with its assigned neighbor and reports it to the main thread. Similarly, the main thread wakes up every τ seconds and gathers the offset information from all the connections and performs the update described in (8). We do not perform any explicit filtering of offset values, besides discarding spurious offsets larger than 500 ms in comparison to previous measurement.[5]

Our program reads the TSC counter directly using the `rdtsc` assembly instruction to minimize reading latencies and maintains a virtual clock that can be directly updated. The list of neighbors is read from a configuration file, and whenever there is no neighbor, the program follows the local Linux clock. Finally, offset measurements are taken using an improved ping pong mechanism proposed in [10].

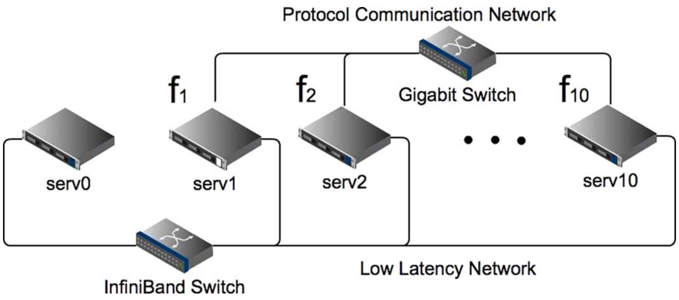


Fig. 7. Testbed of IBM BladeCenter blade servers.

We run our skewless protocol in a cluster of IBM BladeCenter LS21 servers with two AMD Opteron processors of 2.40 GHz, and 16 GB of memory. As shown in Fig. 7, the servers serv1–serv10 are used to run the protocol. The offset measurements are taken through a Gigabit Ethernet switch.

Server serv0 is used as a common reference. It runs the same program that implements Alg1, but without skew adaptations, to measure the offset between itself and the other servers (serv1–serv10). These measurements are obtained through a 10-Gb/s Cisco 4x InfiniBand Switch to minimize network latencies. Since the offset measurements performed by serv0 are done at different instances for different servers, we use linear interpolation to compensate this error. To compute the offset between two servers, say serv1 and serv2 ($x_1(t) - x_2(t)$), we subtract offset measurements obtained form serv0, i.e., $(x_1(t) - x_0(t)) - (x_2(t) - x_0(t))$. Finally, we also eliminate spurious measurements that generate offsets bigger than 1 ms as these are clearly due to network or OS latencies.

We use this testbed to validate the analysis in Sections IV and V. First, we illustrate the effect of different parameters and analyze the effect of the network configuration on convergence (Experiment 1). Then, we present a series of configurations that demonstrate how connectivity between clients is useful in reducing the jitter of a noisy clock source (Experiment 2). We compare the performance of our protocol with respect to NTP version 4 (Experiment 3) and IBM CCT (Experiment 4). Finally, we verify the presence frequency drift in the absence of a leader (Experiment 5) and study the interplay between network delays, wander, and parameter values (Experiment 6).

The output performance signal v_k will be the vector of offset differences between the leader 1 and every other node i, i.e., $v_i(t_k) = x_i(t_k) - x_1(t_k)$ with $i \in \{2, \dots, n\}$. We will use a normalized version of it, referred to here as *mean relative deviation* $\sqrt{S_n}$, as a performance metric. To make these comparisons fair among different servers, we correct our performance value by the empirical mean deviation to compensate biases due to path asymmetries. In other words

$$S_n = \frac{1}{n-1}\sum_{i=2}^{n} \left\langle (x_i - x_1 - < x_i - x_1 >)^2 \right\rangle \tag{31}$$

[5] An offset change of 500 ms within a τ of even 50 s implies a skew of 10 000 ppm, which is our maximum skew accepted.

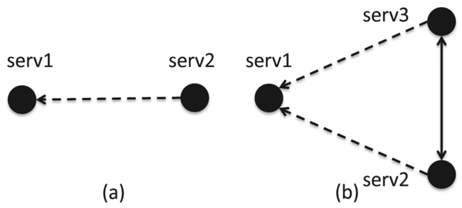


Fig. 8. Effect of topology on convergence: (a) client–server configuration; (b) two clients connected to server and mutually connected.

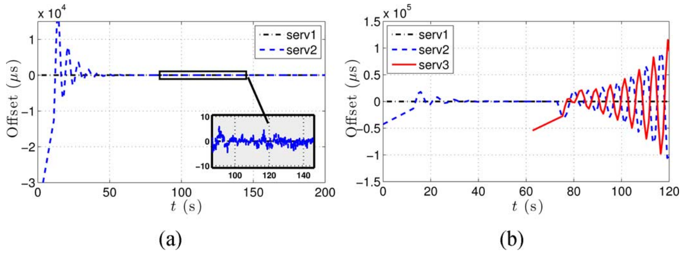


Fig. 9. Loss of stability by change in the network topology. (a) Client–server configuration with $\tau = 1$ s. The client converges, and the algorithm is stable. (b) Two clients mutually connected with $\tau = 1$ s. The algorithm becomes unstable.

where $< \cdot >$ amounts to the sample average. We will also use the 99% Confidence Interval CI_{99} and the maximum offset (CI_{100}) as metrics of accuracy. For example, if $CI_{99} = 10$ μs, then 99% of the offset samples will be within 10 μs of the leader.

Unless explicitly stated, the default parameter values are

$$p = 0.99 \quad \kappa_1 = 1.1 \quad \kappa_2 = 1.0 \quad \alpha_{ij} = \frac{c}{|\mathcal{N}_i|}. \tag{32}$$

The scalar c is a commit or gain factor that will allow us to compensate the effect of τ. Notice that by definition of α_{ij}, $\alpha_{ii} = c$ for every node that is not the leader.

Moreover, these values immediately satisfy (i) and (ii) of Theorem 2 since $1 - p = 0.01$ and $\frac{2\kappa_1}{3p} = 0.7407 > \kappa_1 - \kappa_2 = 0.1$. The remaining condition can be satisfied by modifying τ or equivalently c. Here, we choose to fix $c = 0.7$, which makes condition (iii)

$$\tau < \frac{890.1}{\mu_{\max}} \text{ ms.}$$

For fixed polling interval τ, the stability of the system depends on the value of $\mu_{\max}$, which is determined by the underlying network topology and the values of α_{ij}.

Experiment 1 (Convergence): We first consider the client–server configuration described in Fig. 8(a) with a time-step $\tau = 1$ s. In this configuration, $\mu_{\max} \approx c = 0.7$, and therefore condition (iii) becomes $\tau < 1.2717$ s. Fig. 9(a) shows the offset between serv1 (the leader) and serv2 (the client) in microseconds. There, we can see how serv2 gradually updates $s_2(t_k)$ until the offset becomes negligible.

Fig. 9(a) tends to suggest that the set of parameters given by (32) and $\tau = 1$ s is suitable for deployment on the servers. This is in fact true provided that the network is a directed tree as in Fig. 6(a). The intuition behind this fact is that in a tree, each client connects only to one server. Thus, those connected to the leader will synchronize first, and then subsequent layers will follow.

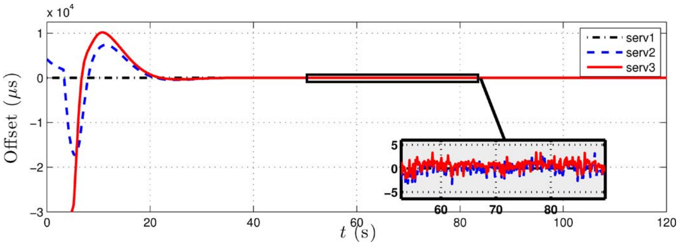


Fig. 10. Configuration of Fig. 8(b) with $\tau = 500$ ms. The algorithm becomes stable after reducing τ from 1 s to 500 ms.

However, once loops appear in the network, there is no longer a clear dependency since two given nodes can mutually get information from each other. This type of dependency might make the algorithm unstable. Fig. 9(b) shows an experiment with the same configuration as Fig. 9(a), in which serv2 synchronizes with serv1 until a third server (serv3) appears after 60 s. At that moment, the system is reconfigured to have the topology of Fig. 8(b), introducing a timing loop between serv2 and serv3. This timing loop makes the system unstable.

The instability arises since after serv3 starts, the new topology has $\mu_{\max} \approx 1.5c = 1.05$. Thus, the time-step condition (iii) becomes $\tau < 847.8$ ms, which is no longer satisfied by $\tau = 1$ s.

This may be solved for the new topology [Fig. 8(b)] by using any τ smaller than 847.8 ms. However, if we want a set of parameters that is independent of the topology, we can use (26) and notice that $\alpha_{\max} = c$ and $\hat{r}_{\max} \approx 1$. We choose

$$\tau = 500 \text{ ms} < \frac{890.2}{2\alpha_{\max}} \text{ ms} = \frac{890.2}{2c} \text{ ms} = 635.9 \text{ ms.}$$

Fig. 10 shows how now serv2 and serv3 can synchronize with serv1 after introducing this change.

Experiment 2 (Timing Loops Effect): We now show how timing loops can be used to collectively outperform individual clients when the time source is noisy.

We run Alg1 on 10 servers (serv1 through serv10). The connection setup is described in Fig. 11. Every node is directly connected unidirectionally to the leader (serv1) and bidirectionally to $2K$ additional neighbors. When $K = 0$, then the network reduces to a star topology, and when $K = 4$, the servers serv2 through serv10 form a complete graph. The dashed arrows in Fig. 11 show the connections where jitter was introduced. To emulate a link with jitter, we added random noise η with values taken uniformly from $\{0, 1, \ldots, \mathrm{Jitter}_{\max}\}$ on both directions of the communication

$$\eta \in \{0, 1, \ldots, \mathrm{Jitter}_{\max}\} \text{ ms.} \tag{33}$$

Notice that the arrow only shows a dependency relationship; the ping pong mechanism sends packets in both direction of the physical communication. We used a value of $\mathrm{Jitter}_{\max} = 10$ ms. Since the error was introduced in both directions of the ping pong, this is equivalent to a standard deviation of 6.05 ms.

Fig. 12 illustrates the relative offset between the two extreme cases: The star topology ($K = 0$) is shown in Fig. 12(a), and the complete subgraph ($K = 4$) is shown in Fig. 12(b).

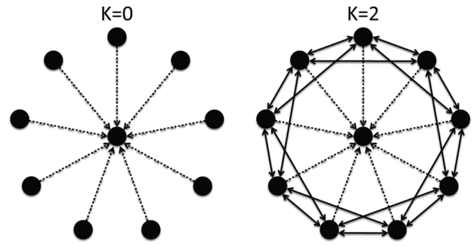
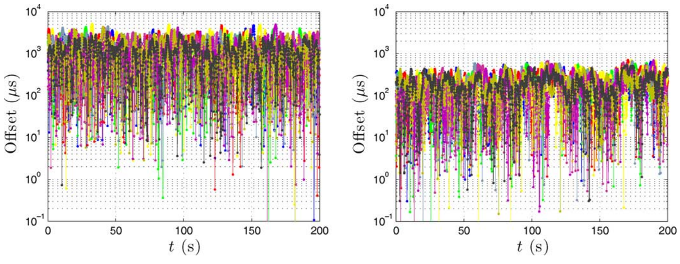


Fig. 11. Leader topologies with $2K$ neighbors connection. Connections to the leader (serv1) are unidirectional, while the connections among clients (serv2 through serv10) are bidirectional.



(a)

(b)

Fig. 12. Offset of the nine servers connected to a noisy clock source. (a) Star topology ($K = 0$). (b) Complete subgraph ($K = 4$).

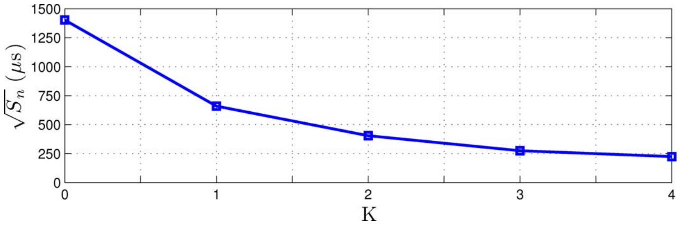


Fig. 13. Effect of the client's communication topology on the mean relative deviation. As the connectivity increases (K increases), the mean relative deviation is reduced by factor of 6.26, i.e., a noise reduction of approximately 8 dB.

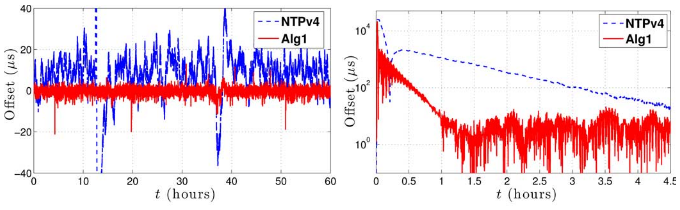
The worst-case offset for $K = 0$ is $CI_{100} = 5.1$ ms, which is on the order of the standard deviation of the jitter. However, when $K = 4$, we obtain a worst-case offset of $CI_{100} = 690.8$ μs, an *order of magnitude improvement.*

The change on the mean relative deviation $\sqrt{S_n}$ as the connectivity among clients increases is studied in Fig. 13. The results presented show that even without any offset filtering mechanism, the network itself is able to perform a distributed filtering that achieves an improvement of up to a factor of 6.26 in $\sqrt{S_n}$, or equivalently a noise reduction of almost 8 dB.

Experiment 3 (Comparison to NTPv4): We now perform a thorough comparison between our protocol (Alg1) and NTPv4. We used a one-hop configuration using serv1 as leader running an NTPv4 server and Alg1, and serv9 and serv10 as clients, connected only to serv1, running NTPv4 and Alg1, respectively.

In order to make a fair comparison, we need both algorithms to use the same polling interval. Thus, we fix $\tau = 16$ s. This can be done for NTP by setting the parameters `minpoll` and `maxpoll` to 4 ($2^4 = 16$ s). The remainder of parameter values for Alg1 were obtained with our optimization framework (using $g_{ij}^{w} = 100$ and $g_{ij}^{d} = 1e-3$) and are given by

$$p = 1.98 \quad \kappa_1 = 1.388 \quad \text{and} \quad \kappa_2 = 1.374. \qquad (34)$$



(a)

(b)

Fig. 14. Performance evaluation between our solution (Alg1) and NTPv4. (a) Offset values of NTPv4 and Alg1 for a period of 60 h. (b) Offset values of NTPv4 and Alg1 after a 25-ms offset introduced in serv1.

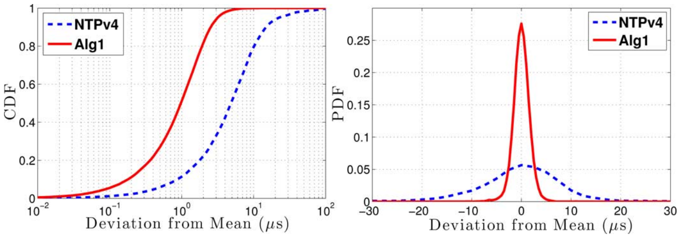


Fig. 15. Empirical CDF and PDF of Alg1 and NTPv4.

Fig. 14(a) shows the time differences between the clients running NTPv4 and Alg1 (serv9 and serv10) and the leader (serv1) over a period of 60 h. It can be seen that Alg1 is able to track serv1's clock keeping an offset smaller than 5 μs for most of the time, while NTPv4 incurs in larger offsets during the same period of time. This difference is produced by the fact that Alg1 is able to react more rapidly to frequency changes, while NTPv4 incurs in more offset corrections that generate larger jitter.

The mean offsets of Alg1 and NTPv4 are -0.48 and 9.00 μs. This difference in mean is mainly due to an asymmetric path on the measurements from serv9 to serv1. After compensating this bias, Alg1 achieves a performance of $\sqrt{S_n} = 1.3$ μs, $CI_{99} = 4.9$ μs, and a maximum offset of $CI_{100} = 20.6$ μs, while NTPv4 obtains $\sqrt{S_n} = 6.4$ μs, $CI_{99} = 74.5$ μs, and a maximum offset of $CI_{100} = 1.4$ ms. Thus, not only does Alg1 achieve a reduction of $\sqrt{S_n}$ by a factor of 5.0 (-7 dB) with respect to NTPv4, but it also obtains smaller confidence intervals and maximum offset values.

A more detailed and comprehensive analysis is presented in Fig. 15 where we plot the cumulative distribution function (CDF) and probability density function (PDF) of the samples. The improvement of Alg1 with respect to NTPv4 is again clearly seen here.

Finally, we investigate the speed of convergence. Starting from both clients synchronized to server serv1, we introduce a 25-ms offset. Fig. 14(b) shows how Alg1 is able to converge to a 20-μs range within 1 h, while NTPv4 needs 4.5 h to achieve the same synchronization precision.

Experiment 4 (Comparison to IBM CCT): We now proceed to compare the performance of Alg1 with respect to IBM CCT. Notice that unlike IBM CCT, our solution does not perform any previous filtering of the offset samples; the filtering is performed instead by calibrating the parameters. Here, we use $c = 0.70$, $\tau = 250$ ms, $\kappa_1 = 0.1385$, $\kappa_2 = 0.1363$, and $p = 0.62$.

In Fig. 16(a), we present the mean relative deviation $\sqrt{S_n}$ for two clients connected directly to the leader as the jitter is

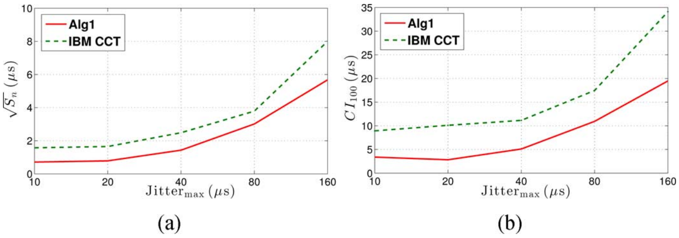


Fig. 16. Performance evaluation between our solution (Alg1) and IBM CCT. (a) Mean relative deviation $\sqrt{S_n}$. (b) Maximum offset.

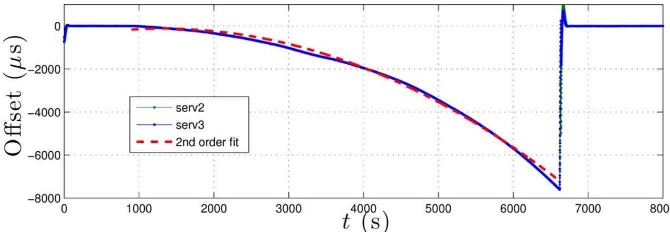


Fig. 17. Frequency drift.

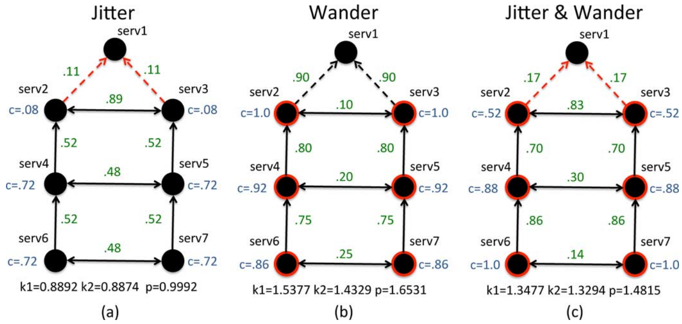


Fig. 18. Network scenarios and optimal parameters. (a) Jitter. (b) Wander. (c) Jitter and wander.

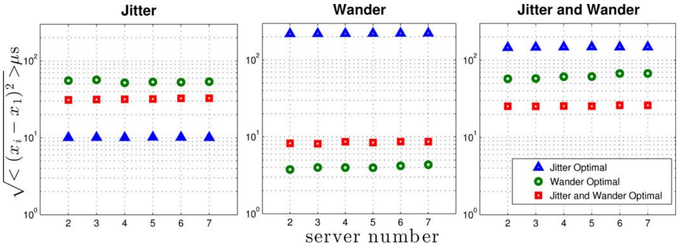


Fig. 19. $\mathcal{H}_2$ Performance optimization: offset variance versus server number.

increased from $\text{Jitter}_{\max} = 10$ μs to $\text{Jitter}_{\max} = 160$ μs, doubling $\text{Jitter}_{\max}$ each time, with a granularity in the random generator of 1 μs. The worst-case offset is shown in Fig. 16(b). Each data point is computed using a sample run of 250 s.

Our algorithm consistently outperforms IBM CCT in terms of both $\sqrt{S_n}$ and worst-case offset. The performance improvement is due to two reasons. First, the noise filter used by the IBM CCT algorithm is tailored for noise distributions that are mostly concentrated close to zero with sporadic large errors. However, it does not work properly in cases where the distribution is more homogeneous as in this case. Second, by choosing $\delta\kappa = \kappa_1 - \kappa_2 = 0.002 \ll 1$, the protocol becomes very robust to offset errors.

Experiment 5 (Frequency Drift Without Leader): We now proceed to experimentally verify that without a leader, the system tends to constantly drift the frequency. Our analysis predicts that even the minor bias in the offset measurements will produce this effect. To verify this phenomenon, we use the network topology in Fig. 8(b) with $\tau = 0.5$ s and wait for the system to converge.

After 1000 s, the timing process of serv1 is turned off. Fig. 17 shows how the offsets of serv2 and serv3 start to grow in a parabolic trajectory characteristic of a constant acceleration, i.e., constant drift. After 6600 s, serv1 is restarted, and the system quickly recovers synchronization. A second-order fit of the faulty trajectory was performed, obtaining a drift of approximately -250 ns/s^2. While this is not quite significant in the first few minutes, it becomes significant as time goes on.

Experiment 6 (Jitter and Wander Tradeoff): Finally, we use the proposed $\mathcal{H}_2$ optimization scheme to show how the optimal parameter values depend on the different noise conditions within the network described in Fig. 18. We consider three different noise scenarios in which we either add jitter between server serv1 and servers serv2 and serv3, and/or add wander on servers serv2–serv7. In all the cases, we used $\tau = 0.5$ s and make offset measurements through the InfiniBand switch to minimize any additional source of noise.

The jitter is generated by adding in both directions of the physical communication a random value η similarly to Experiment 2 [cf. (33)], but with a $\text{Jitter}_{\max} = 100$ μs. This generates an aggregate offset measurement noise of zero mean and standard deviation of 40.8 μs. On the other hand, the wander is generated by adding Gaussian noise with zero mean and standard deviation of 0.2 ppm in the $s_i(t_k)$ adaptations. As discussed in Section V, this noise can be used to emulate the wander of a bad-quality clock.

We used different values of g_{ij}^w and g_i^d to differentiate the noise conditions in the optimization scheme. The large jitter scenario is represented by $g_i^d = 1e-3$ $\forall i$, $g_{21}^w = g_{31}^w = 100$, and $g_{ij}^w = 1$ otherwise. The large wander scenario is represented by $g_i^d = 1e-1$ $\forall i$ and $g_{ij}^w = 1$. Finally, the large jitter and wander scenario is represented using $g_i^d = 1e-1$ $\forall i$, $g_{21}^w = g_{31}^w = 100$, and $g_{ij}^w = 1$ otherwise. The output parameter values for all three cases are present also in Fig. 18.

Fig. 19 shows the standard deviation of the offset between servers serv2–serv7 and serv1 in the three experimental scenarios and for the three different sets of parameters shown in Fig. 18. It can be seen that although the configuration tuned for jitter performs very well in cases with large jitter, it performs quite poorly in scenarios with large wander. Similarly, the configuration tuned for wander does not perform well in high jitter scenarios. However, the configuration tuned for jitter and wander is able to provide acceptable performance in all three experimental scenarios. Thus, we experimentally demonstrate a fundamental tradeoff between jitter and wander.

Finally, it is interesting to notice that due to the fact the optimization is solved using $v_i(t_k) = x_i(t_k) - x_1(t_k)$ as performance metric, the choice of parameters does not degrade the performance of each clock with the hop count.

VII. Conclusion

This paper presents a clock synchronization protocol that is able to synchronize networked nodes without explicit estimation of the clock skews and steep corrections on the time. Our solution is guaranteed to converge even in the presence of timing loops that allow different clients to share timing information and even collectively outperform individual clients when the time source has large jitter. The system is robust to noisy measurements and wander provided that the topology has a well-defined leader, and we can optimize the parameter values to minimize noise variance. We implemented our solution on a cluster of IBM BladeCenter servers and empirically verified our predictions and our protocol's supremacy over some existing solutions.

Further evaluation of our protocol is needed. In particular, we are interested in comparing our solution to other protocols such as PTP and RADclock, as well as studying its robustness under stressed scenarios. Another interesting direction is to devise a distributed algorithm, exploiting our optimization framework, that can adapt the parameter values depending on the network condition.

Appendix A
Proof of Lemmas

A. Proof of Lemma 1

Proof: We first compute the characteristic polynomial

$$\begin{aligned}
&\det(\lambda I_{3n} - A)\\
&= \begin{vmatrix} (\lambda-1)I_n & -\tau R & \mathbf{0}_{n\times n} \\ \kappa_1 L & (\lambda-1)I_n & \kappa_2 I_n \\ pL & 0 & (\lambda-1+p)I_n \end{vmatrix}\\
&= (\lambda-1)^n \begin{vmatrix} (\lambda-1)I_n + \frac{\tau\kappa_1}{\lambda-1}LR & \kappa_2 I_n \\ \frac{\tau p}{\lambda-1}LR & (\lambda-1+p)I_n \end{vmatrix}\\
&= \det\left((\lambda-1)^2(\lambda-1+p)I_n + [(\lambda-1)\kappa_1 + p(\kappa_1-\kappa_2)]\tau LR\right)\\
&= \prod_{l=1}^{n} g_l(\lambda)
\end{aligned}$$

where $g_l(\lambda)$ is as defined in (17) and we have iteratively used the determinant property of block matrices $\det(A) = \det(A_{11})\det(A\backslash A_{11})$ where $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and $A\backslash A_{11} = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement of A_{11} [25].

Thus, $\lambda = 1$ is a double-root of the characteristic polynomial if and only if $\kappa_1 \neq \kappa_2$, $p > 0$, and τLR has a simple zero eigenvalue, i.e., (18). Now, since R is nonsingular, (18) must hold for the eigenvalues of L as well, which is in fact true if and only if the directed graph $G(V,E)$ is connected [20]. ■

B. Proof of Lemma 2

Proof: We start by computing the right Jordan chain. By definition of ζ_1, $(A - I)\zeta_1 = 0_n$. Thus, if $\zeta_1 = [x_1^T \ s_1^T \ y_1^T]^T$, then the following system of equations must be satisfied:

$$\tau R s_1 = \mathbf{0}_n \tag{35a}$$

$$-\kappa_1 L x_1 - \kappa_2 y_1 = \mathbf{0}_n \tag{35b}$$

$$-pLx_1 - py_1 = \mathbf{0}_n. \tag{35c}$$

Equation (35a) implies $s_1 = 0$. Now, since $p > 0$, (35c) implies $Lx_1 = -y_1$, which when substituted in (35b) gives $(\kappa_2 - \kappa_1)y_1 = \mathbf{0}_n$. Thus, since $\kappa_1 \neq \kappa_2$, $y_1 = \mathbf{0}_n$ and $x_1 \in \ker(L)$. By choosing $x_1 = \alpha_1 \mathbf{1}_n$ (for some $\alpha_1 \neq 0$), we obtain $\zeta_1 = \alpha_1 \left[\mathbf{1}_n^T \ \mathbf{0}_n^T \ \mathbf{0}_n^T\right]^T$.

Notice that the computation also shows that ζ_1 is the unique eigenvector of $\mu(A) = 1$, which implies that there is only one Jordan block, of size 2. The second member of the chain, ζ_2, and ζ_3 can be computed similarly by solving $(A - I_n)\zeta_2 = \zeta_1$ and $(A - (1-p)I_n)\zeta_3 = \mathbf{0}_n$. This gives

$$\zeta_2 = \begin{bmatrix} \alpha_2 \mathbf{1}_n \\ \frac{\alpha_1}{\tau} R^{-1}\mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix} \quad \zeta_3 = \alpha_3 \begin{bmatrix} -\frac{\tau\kappa_2}{p^2}\mathbf{1}_n \\ \frac{\kappa_2}{p} R^{-1}\mathbf{1}_n \\ R^{-1}\mathbf{1}_n \end{bmatrix}.$$

In computing $\zeta_3 = [x_3^T \ s_3^T \ y_3^T]^T$, we obtain $Lx_3 = 0$ and $Rx_3 = -\frac{\tau}{p}s_3 = -\frac{\kappa_2\tau}{p^2}y_3$. ζ_3 follows by taking $y_3 = \alpha_3 R^{-1}\mathbf{1}_n$.

The vectors η_1, η_2 and η_3 can be solved in the same way using $\eta_2^T(A - I) = \mathbf{0}_n^T$, $\eta_1^T(A - I) = \eta_2^T$, and $\eta_3^T(A - (1-p)I) = \mathbf{0}_n^T$. This gives $\eta_1 = \left[\frac{\beta_2}{\tau}R^{-1}\xi^T \ \beta_1\xi^T \ (-\frac{\kappa_2}{p}\beta_1 + \frac{\kappa_2}{p^2}\beta_2)\xi^T\right]^T$, $\eta_2 = \beta_2\left[\mathbf{0}_n^T \ \xi^T \ \frac{\kappa_2}{p}\xi^T\right]^T$, and $\eta_3 = \beta_3\left[\mathbf{0}_n^T \ \mathbf{0}_n^T \ \xi^T\right]^T$. We set $\alpha_1 = \alpha_2 = \alpha_3 = 1$; this can be done without loss of generality provided we still satisfy $\eta_l^T\zeta_l = 1$ and $\eta_l^T\zeta_h = 0$ for $l \neq h$. Finally, $\eta_1^T\zeta_1 = 1$ gives $\beta_2 = \gamma\tau$, $\eta_3^T\zeta_3 = 1$ gives $\beta_3 = \gamma$, and $\eta_1^T\zeta_2 = 0$ gives $\beta_1 = -\beta_2 = -\gamma\tau$. ■

Appendix B
Proof of Theorem 1

Proof:

1) $\Longrightarrow$ 2): Since we are under the conditions of Lemmas 1 and 2, then we can use (23), and since $\rho(\hat{J}_2) < 1$, all the eigenvalues of $\hat{A}$ are within the unit circle, i.e., $\rho(\hat{A}) < 1$. Therefore, it follows that $\delta z_k = \hat{A}^k z_0 \to \mathbf{0}_{3n\times 3n} z_0 = \mathbf{0}_{3n}$. To show (15b), we first notice that

$$\tilde{x}(t_{k+1}) = \tilde{x}(t_k) + \tau\tilde{s}(t_k) \tag{36a}$$

$$\tilde{s}(t_{k+1}) = \tilde{s}(t_k) - \kappa\tilde{y}(t_k) \tag{36b}$$

$$\tilde{y}(t_{k+1}) = (1-p)\tilde{y}(t_k). \tag{36c}$$

Therefore, since $|1 - p| < 1$, (36c) implies that $\tilde{y}(t_k) \to 0$. Thus, by (36b) we also have $\tilde{s}(t_k) \to s^*$ for some s^*, which also implies that $\tilde{x}(t_{k+1}) - \tilde{x}(t_k) \to \tau s^*$ giving $\tilde{x}(t_k) \to x^{\mathrm{ref}}(t_k) = x^* + (t_k - t_0)s^*$ for some x^*.

2) $\Longrightarrow$ 3): This follows directly from (10) and (11).

3) $\Longrightarrow$ 1): The algorithm achieves synchronization whenever (9) holds. Then, it follows from (12) and (9) that asymptotically the system behaves according to

$$\begin{aligned}
z_k = \begin{bmatrix} x_k \\ s_k \\ y_k \end{bmatrix} &= \begin{bmatrix} x^*\mathbf{1}_n \\ r^*R^{-1}\mathbf{1}_n \\ \mathbf{0}_n \end{bmatrix} + k\begin{bmatrix} \tau r^*\mathbf{1}_n \\ \mathbf{0}_n \\ \mathbf{0}_n \end{bmatrix}\\
&= (\tau r^*\zeta_2 + (x^* - \tau r^*)\zeta_1) + kr^*\tau\zeta_2.
\end{aligned}$$

Thus, since $P := [\zeta_1 \ \ \ldots \ \ \zeta_{3n}]$ is invertible, its columns ζ_l are linearly independent. Therefore, if the system synchronizes for arbitrary initial condition, then it must be the case that the effect of the remaining modes $\mu_l(A)$ vanishes, which can only happen if for every $\mu_l(A) \neq 1$, $|\mu_l(A)| < 1$ and the multiplicity of $\mu_l(A) = 1$ is two, i.e., $\rho(\hat{J}_2) < 1$.. Now suppose that either $G(V, E)$ is not connected, $\kappa_1 = \kappa_2$, $p = 0$. Then, by Lemma 1, the multiplicity of $\mu_l(A) = 1$ is not two, which is a contradiction. Similarly, if $p > 2$, $p < 0$, then the system has at least one eigenvalue $|\mu_l(A)| > 1$. Thus, we must have $\rho(\hat{J}_2) < 1$, $\kappa_1 \neq \kappa_2$, $2 > p > 0$, and G connected whenever the system synchronizes for arbitrary initial condition.

Finally, to obtain (25), we use a similar computation to the one of Lemma 1 to show that $\tilde{P}$ and $\tilde{P}^{-1}$ in (24) are given by

$$\tilde{P} = \begin{bmatrix} 1 & 1 & -\tau\frac{\kappa_2}{p^2} \\ 0 & \frac{1}{\tau} & \frac{\kappa_2}{p} \\ 0 & 0 & 1 \end{bmatrix} \quad \tilde{P}^{-1} = \begin{bmatrix} 1 & -\tau & (\frac{1}{p^2}+\frac{1}{p})\kappa_2\tau \\ 0 & \tau & -\tau\frac{\kappa_2}{p} \\ 0 & 0 & 1 \end{bmatrix}.$$

Thus, since $\tilde{A}^k = \tilde{P}\hat{J}_1^k\tilde{P}^{-1}$, a direct computation shows that

$$\tilde{A}^k = \tilde{\zeta}_1\tilde{\eta}_1^T + \tilde{\zeta}_2\tilde{\eta}_2^T + k\tilde{\zeta}_1\tilde{\eta}_2^T + (1-p)^k\tilde{\zeta}_3\tilde{\eta}_3^T$$

where $\tilde{P} = [\tilde{\zeta}_1\tilde{\zeta}_2\tilde{\zeta}_3]$ and $\tilde{P}^{-1} = [\tilde{\eta}_1\tilde{\eta}_2\tilde{\eta}_3]^T$. Therefore

$$\tilde{z}_k = \tilde{A}^k\tilde{z}_0 \to \begin{bmatrix} 1 & \tau k & -\tau k\frac{\kappa_2}{p} + \tau\frac{\kappa_2}{p^2} \\ 0 & 1 & -\frac{\kappa_2}{p} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \tilde{x}(t_0) \\ \tilde{s}(t_0) \\ \tilde{y}(t_0) \end{bmatrix}$$

which implies that $\tilde{y}(t_k) \to 0$, $\tilde{s}(t_k) \to \tilde{s}(t_0) - \frac{\kappa_2}{p}y(t_0)$ and $\tilde{x}(t_k) \to \tilde{x}(t_0) + \tau\frac{\kappa_2}{p^2}\tilde{y}(t_0) + \tau k(s(t_0) - \frac{\kappa_2}{p}y(t_0))$. Result follows by definition of x^* and r^* in (25) and definition of $x^{\text{ref}}(t_k)$ in (9). ■

APPENDIX C
PROOF OF THEOREM 2

Proof: We will show that when $G(V, E)$ is connected with $\mu(L) \in \mathbb{R}$, then (i)–(iii) are equivalent to the conditions of Theorem 1.

Since $G(V, E)$ is connected and (i)–(ii) satisfies $p > 0$ and $\kappa_1 \neq \kappa_2$, the conditions of Lemma 1 are satisfied. Therefore, the multiplicity of $\mu(A) = 1$ is two, and by (18) these are the roots of $g_n(\lambda) = (\lambda - 1)^2(\lambda - 1 + p)$, which corresponds to the case $\nu_n = 0$. Thus, to satisfy Theorem 1, we need to show that the remaining eigenvalues are strictly in the unit circle. This is true for the remaining root of $g_n(\lambda)$ if and only if (i).

For the remaining $g_l(\lambda)$, this implies that they are Schur polynomials. Thus, we will show that $g_l(\lambda)$ is a Schur polynomial if and only if (i)–(iii) hold. We drop the subindex l for the rest of the proof.

We first transform the Schur stability problem into a Hurwitz stability problem. Consider the change of variable $\lambda = \frac{s+1}{s-1}$. Then, $|\lambda| < 1$ if and only if $\mathbb{R}[s] < 0$.

Now, since $\nu > 0$ by (18), let

$$P(s) = \frac{(s-1)^3}{\delta\kappa p\nu} g\left(\frac{s+1}{s-1}\right) = s^3 + \left(\frac{2\kappa_1}{\delta\kappa p} - 3\right)s^2 + \left(\frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p}\right)s + \frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1$$

where $\delta\kappa = \kappa_1 - \kappa_2$.

We will apply Hermite–Biehler Theorem to $P(s)$, but first let us express what 1) and 2) of the theorem mean here.

Condition 1) becomes

$$\frac{2\kappa_1}{\delta\kappa p} - 3 > 0. \tag{37}$$

Now let $P^r(\omega)$ and $P^i(\omega)$ be as in Hermite–Biehler Theorem, i.e., let

$$P^r(\omega) = -\left(\frac{2\kappa_1}{\delta\kappa p} - 3\right)\omega^2 + \frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1$$
$$P^i(\omega) = -\omega^3 + \left(\frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p}\right)\omega.$$

The roots of $P^r(\omega)$ and $P^i(\omega)$ are given by $\omega_0 = \pm\sqrt{\omega_0^r}$ and $\omega_0 \in \{0, \pm\sqrt{\omega_0^i}\}$, respectively, where

$$\omega_0^r := \frac{\frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1}{\frac{2\kappa_1}{\delta\kappa p} - 3} \quad \omega_0^i := \frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p}. \tag{38}$$

Since the roots $P^r(\omega)$ and $P^i(\omega)$ must be real, we must have $\omega_0^r > 0$ and $\omega_0^i > 0$. Therefore, by monotonicity of the square root, the interlacing condition 2) is equivalent to

$$0 < \omega_0^r < \omega_0^i. \tag{39}$$

Thus, we will show: (i)–(iii) hold $\iff$ (37) and (39) hold.

It is straightforward to see that, using (i) and (ii), we can get (37). On the other hand, $\omega_o^i > 0$ from (39) together with (37) gives $0 < \frac{4}{\delta\kappa\nu} + 3 - \frac{4\kappa_1}{\delta\kappa p} < \frac{4}{\delta\kappa\nu}$, which implies that $\delta\kappa > 0$, and therefore (ii) follows.

Now using (37) and the definition of ω_0^r in (38), $\omega_0^r > 0$ becomes $\frac{4(2-p)}{\delta\kappa p\nu} + \frac{2\kappa_1}{\delta\kappa p} - 1 > 0$, which always holds under (i) and (ii) since the first term is always positive and $\frac{2\kappa_1}{\delta\kappa p} - 1 > \frac{2\kappa_1}{\delta\kappa p} - 3 > 0$ by (37).

Using (38), $\omega_0^r < \omega_0^i$ is equivalent to

$$\nu < \frac{p(\kappa_2 - \delta\kappa p)}{(\kappa_1 - \delta\kappa p)^2}. \tag{40}$$

Finally, $\nu_l = \mu_l(\tau LR) = \tau\mu_l(LR)$. Thus, since (40) should hold $\forall l \in \{1, \ldots, n-1\}$, then

$$\tau < \min_l \frac{p(\kappa_2 - \delta\kappa p)}{\mu_l(LR)(\kappa_1 - \delta\kappa p)^2} = \frac{p(\kappa_2 - \delta\kappa p)}{\mu_{\max}(\kappa_1 - \delta\kappa p)^2}$$

which is exactly (iii). ■

APPENDIX D
PROOF OF THEOREMS 3 AND 4

Proof of Theorem 3: Using (11), (28), and (29) we can modify (36) to get

$$\tilde{x}(t_{k+1}) = \tilde{x}(t_k) + \tau\tilde{s}(t_k) \tag{41a}$$
$$\tilde{s}(t_{k+1}) = \tilde{s}(t_k) - \kappa_2\tilde{y}(t_k) + \kappa_1\tilde{w} \tag{41b}$$
$$\tilde{y}(t_{k+1}) = (1-p)\tilde{y}(t_k) + p\tilde{w} \tag{41c}$$

where $\tilde{w} = -\xi^T B_G^- \text{diag}[\alpha_{ij}g_{ij}^w]\bar{w} = \sum_{i=1}^n \xi_i \sum_{j\in\mathcal{N}_i} \alpha_{ij}g_{ij}^w\bar{w}_{ij}$. It follows then that (41c) implies that $\tilde{y}(t_k) \to \tilde{w}$, which implies that $\tilde{s}(t_{k+1}) - \tilde{s}(t_k) \to (\kappa_1 - \kappa_2)\tilde{w}$.

Therefore, since $\kappa_1 \neq \kappa_2$, $\tilde{s}(t_k)$ constantly drifts unless

$$\tilde{w} = -\xi^T B_G^- \text{diag}[\alpha_{ij}g_{ij}^w]\bar{w} = 0. \tag{42}$$

Finally, there are two different scenarios in which (42) can be satisfied.

1) G has a unique leader (say $i = 1$): In this case, we have $\mathcal{N}_1 = \emptyset$, i.e., $\alpha_{1j} = 0\ \forall j$, $\xi_1 = 1$ and $\xi_j = 0\ \forall j \neq 1$. That is $-\xi^T B_G^- \text{diag}[\alpha_{ij} g_{ij}^w]\bar{w} = \xi_1 0 = 0$
2) G does not have a well-defined root: Thus, there are at least two nodes with $\xi_i \neq 0$, and $\bar{w}$ is such that $\xi^T B_G^- \text{diag}[\alpha_{ij} g_{ij}^w]\bar{w} = 0$.

However, 2) is only satisfied by a set of values of $\bar{w}$ with zero measure. Thus, there should be a unique leader for synchronization. ■

Proof of Theorem 4: Similar to the proof of Theorem 3, the evolution of δz_k can be described using

$$\delta x_{k+1} = \delta x_k + \tau R \delta s_k \tag{43a}$$

$$\delta s_{k+1} = -\kappa_1 L \delta x_k + \delta s_k - \kappa_2 \delta y_k + \kappa_1 \delta w \tag{43b}$$

$$\delta y_{k+1} = -pL\delta x_k + (1-p)\delta y_k + p\delta w. \tag{43c}$$

Now, since $\rho(NA) < 1$, then $\delta z_k \to \delta z^*$, where δz^* is a fixed point of (43). Thus, (43a) implies that $\delta \bar{s}^* = 0$ and (43b) $-\frac{\kappa_1}{p}$ (43c) gives $(\kappa_1 - \kappa_2)\delta \bar{y}^* = 0$, which implies $\delta \bar{y}^* = 0$ since $\kappa_1 > \kappa_2$. Finally, using (43c) again, we have

$$L\delta \bar{x}^* = \delta w \tag{44}$$

$$L^\dagger L \delta \bar{x}^* = L^\dagger \delta w \tag{45}$$

$$N_1 N_3 \delta \bar{x}^* = N_1 L^\dagger \delta w \tag{46}$$

$$\delta \bar{x}^* = N_1 L^\dagger \delta w \tag{47}$$

where in (45) we multiplied by $L^\dagger$, in (46) we used $N_3 := L^\dagger L = (I_n - \frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T)$ and left multiply by N_1, and in (47) we used identities $N_1 N_3 = N_1$ and $N_1 \delta \bar{x} = N_1^2 \bar{x} = N_1 \bar{x} = \delta \bar{x}$. ■

Appendix E
$\mathcal{H}_2$ Optimization Using Hifood

The software package Hifood [29] does not solve (30) directly. Instead, it solves

$$\min_{K,X} \quad f(K) := \sqrt{\text{trace}[X\bar{B}\bar{B}^T]} \tag{48a}$$

$$\text{subject to} \quad \rho(\bar{A}) \leq \rho^* \tag{48b}$$

$$X = \bar{A}^T X \bar{A} + \bar{C}^T \bar{C} \tag{48c}$$

where $\bar{A} := A_1 + B_2 K C_2$, $\bar{B} := B_1 + B_2 K D_{21}$, and $\bar{C} := C_1$. In this formulation, δz_k is interpreted as evolving according to the closed-loop standard-form system

$$\begin{aligned} \delta z_{k+1} &= (A_1 + B_2 K C_2)\delta z_k + (B_1 + B_2 K D_{21}) e_k \\ v_k &= C_1 \delta z_k \end{aligned}$$

and the optimization variable K is the static-output feedback matrix.

Therefore, to use Hifood, we first need to rewrite (30) using (48). This can be done by setting

$$A_1 = \hat{A},\ C_1 = \hat{C}, C_2 = \begin{bmatrix} B_G^T & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times n} \\ \mathbf{0}_{n\times n} & I_n & \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times n} & \mathbf{0}_{n\times n} & I_n \end{bmatrix}$$

$$B_2 = \begin{bmatrix} N_1 R & \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} & \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times n} & B_G^- & N_2 & \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times n} & \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} & B_G^- & N_2 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times m} & \text{diag}[g_i^d] \\ \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} \end{bmatrix} \quad D_{21} = \begin{bmatrix} \text{diag}[g_{ij}^w] & \mathbf{0}_{m\times n} \\ \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} \\ \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} \end{bmatrix}$$

$$K = \begin{bmatrix} \mathbf{0}_{n\times m} & \tau I_n & \mathbf{0}_{n\times n} \\ -\kappa_1 \text{diag}[\alpha_{ij}] & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times n} \\ \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} & -\kappa_2 I_n \\ -p\text{diag}[\alpha_{ij}] & \mathbf{0}_{m\times n} & \mathbf{0}_{m\times n} \\ \mathbf{0}_{n\times m} & \mathbf{0}_{n\times n} & -pI_n \end{bmatrix}.$$

Using these definitions, it is straightforward to verify that $(A_1 + B_2 K C_2) = \hat{A}$, $B_1 + B_2 K D_{21} = \hat{B}$, and $C_1 = \hat{C}$.

The main difficulty in solving (30) instead of (48) is that our controller K is a nonlinear function of the parameters $K(\kappa_1, \kappa_2, p, \alpha)$ and cannot be readily obtained using (48). Furthermore, the main source of nonlinearity comes from the products $\kappa_1 \text{diag}[\alpha_{ij}]$ and $p\text{diag}[\alpha_{ij}]$. This structure is not currently supported by traditional software distributions, which usually only support sparsity patterns, and therefore needs to be implemented.

Fortunately, Hifood only uses gradient information in their implementation of BGS and gradient bundle stages. Thus, to implement discrete-time $\mathcal{H}_2$ optimization, we generated a new MATLAB subroutine that evaluated the $\mathcal{H}_2$ norm f as well as its gradients.

The evaluation of the gradient is performed in three stages using the chain rule. We first compute the gradients of f with respect to $\bar{A}$, $\bar{B}$ and $\bar{C}$, which are given by

$$\nabla_{\bar{A}} f = \frac{1}{f} X \bar{A} Y \quad \nabla_{\bar{B}} f = \frac{1}{f} X \bar{B} \quad \nabla_{\bar{C}} f = \frac{1}{f} \bar{C} Y$$

where Y is the solution to $Y = \bar{A} Y \bar{A}^T + \bar{B}\bar{B}^T$.

Once $\nabla_{\bar{A}} f$, $\nabla_{\bar{B}} f$, and $\nabla_{\bar{C}} f$ are computed, we can use the subroutines of Hifood to compute $\frac{\partial \bar{A}}{\partial K}$, $\frac{\partial \bar{B}}{\partial K}$, and $\frac{\partial \bar{C}}{\partial K}$. Finally, we obtain

$$\begin{aligned} &\nabla_{\kappa_1} f \\ &= \text{trace}\left[\left(\nabla_{\bar{A}} f^T \frac{\partial \bar{A}}{\partial K} + \nabla_{\bar{B}} f^T \frac{\partial \bar{B}}{\partial K} + \nabla_{\bar{C}} f^T \frac{\partial \bar{C}}{\partial K}\right) \frac{\partial K}{\partial \kappa_1}\right] \end{aligned}$$

and similarly for other parameters.

References

[1] E. Mallada, X. Meng, M. Hack, L. Zhang, and A. Tang, "Skewless network clock synchronization," in *Proc. 21st IEEE ICNP*, 2013, pp. 1–10.

[2] D. Mills *et al.*, "Network Time Protocol version 4: Protocol and algorithms specification," Internet Engineering Task Force, Tech. Rep., 2010.

[3] A. Sobeih, M. Hack, Z. Liu, and L. Zhang, "Almost peer-to-peer clock synchronization," in *Proc. IEEE IPDPS*, 2007, pp. 1–10.

[4] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002, 2008.

[5] D. Veitch, J. Ridoux, and S. B. Korada, "Robust synchronization of absolute and difference clocks over networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 2, pp. 417–430, Apr. 2009.

[6] R. Carli and S. Zampieri, "Network clock synchronization based on the second-order linear consensus algorithm," *IEEE Trans. Autom. Control*, vol. 59, no. 2, pp. 409–422, Feb. 2014.

[7] E. Mallada and A. Tang, "Distributed clock synchronization: Joint frequency and phase consensus," in *Proc. 50th IEEE CDC-ECC*, 2011, pp. 6742–6747.

[8] J. Ridoux, D. Veitch, and T. Broomhead, "The case for feed-forward clock synchronization," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 231–242, Feb. 2012.

[9] J. C. Corbett *et al.*, "Spanner: Google's globally distributed database," *Trans. Comput. Syst.*, vol. 31, no. 3, pp. 1–22, Aug. 2013.

[10] S. Froehlich *et al.*, "Achieving precise coordinated cluster time in a cluster environment," in *Proc. IEEE ISPCS*, 2008, pp. 54–58.

[11] L. Zhang, Z. Liu, and C. Honghui Xia, "Clock synchronization algorithms for network measurements," in *Proc. 21st Annu. IEEE INFOCOM*, 2002, pp. 160–169.

[12] H. Kim, X. Ma, and B. R. Hamilton, "Tracking low-precision clocks with time-varying drifts using Kalman filtering," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 257–270, Feb. 2012.

[13] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Winter, 2002.

[14] D. Hunt, G. Korniss, and B. K. Szymanski, "Network synchronization in a noisy environment with time delays: Fundamental limits and trade-offs," 2010 [Online]. Available: http://www.arXiv.org

[15] H. Marouani and M. R. Dagenais, "Internal clock drift estimation in computer clusters," *J. Comput. Syst., Netw., Commun.*, vol. 2008, p. 583162, Jan. 2008.

[16] S. B. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," in *Proc. 18th Annu. IEEE INFOCOM*, 1999, pp. 227–234.

[17] M. D. Lemmon, J. Ganguly, and L. Xia, "Model-based clock synchronization in networks with drifting clocks," in *Proc. Pacific Rim Int. Symp. Dependable Comput.*, 2000, pp. 177–184.

[18] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz, "Distributed synchronization in wireless networks," *IEEE Signal Process. Mag.*, vol. 25, no. 5, pp. 81–97, Sep. 2008.

[19] D. Mills, "Network Time Protocol version 3: Specification, implementation and analysis," Internet Engineering Task Force, Tech. Rep., 1992.

[20] D. Xie and S. Wang, "Consensus of second-order discrete-time multi-agent systems with fixed topology," *J. Math. Anal. Appl.*, vol. 387, no. 1, pp. 8–16, 2012.

[21] E. Mallada and F. Paganini, "Stability of node-based multipath routing and dual congestion control," in *Proc. 47th IEEE CDC*, 2008, pp. 1398–1403.

[22] W. Ren and R. Beard*, Distributed Consensus in Multi-Vehicle Cooperative Control*. New York, NY, USA: Springer, 2007.

[23] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Trans. Autom. Control*, vol. 48, no. 6, pp. 988–1001, Jun. 2003.

[24] S. P. Bhattacharyya, H. Chapellat, and L. H. Keel*, Robust Control*, ser. The Parametric Approach. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.

[25] R. A. Horn and C. R. Johnson*, Matrix Analysis*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[26] D. Arzelier *et al.*, "H2 for HIFOO," 2010 [Online]. Available: http://www.arXiv.org

[27] E.-S. M. E. Mostafa, "Computational design of optimal discrete-time output feedback controllers," *J. Oper. Res. Soc. Jpn.*, vol. 51, no. 1, pp. 15–28, 2008.

[28] S. Gumussoy *et al.*, "Multiobjective robust control with HIFOO 2.0," 2009 [Online]. Available: http://www.arXiv.org

[29] A. P. Popov, H. Werner, and M. Millstone, "Fixed-structure discrete-time $\mathrm{H}\infty$ controller synthesis with HIFOO," in *Proc. 49th IEEE CDC*, 2010, pp. 3152–3155.

Enrique Mallada (S'09–M'13) received the Ingeniero en Telecommunications degree from Universidad ORT, Montevideo, Uruguay, in 2005, and the Ph.D. degree in electrical and computer engineering with a minor in applied mathematics from Cornell University, Ithaca, NY, USA, in 2014.

He is currently a Postdoctoral Fellow with the Center for the Mathematics of Information (CMI), California Institute of Technology, Pasadena, CA, USA. His research interests include networks, control, nonlinear dynamics, and optimization, with applications to power and information systems.

Dr. Mallada was a recipient of the Organization of American States (OAS) Academic Scholarship from 2008 to 2010, the Jacobs Fellowship from Cornell University in 2011, and the Cornell ECE Director's Thesis Research Award in 2014.

Xiaoqiao Meng received the B.S. degree in electrical engineering from the University of Science and Technology of China, Hefei, China, in 1998, the M.S. degree in pattern recognition and intelligent systems from the Chinese Academy of Sciences, Beijing, China, in 2000, and the Ph.D. degree in computer science from the University of California, Los Angeles (UCLA), CA, USA, in 2006.

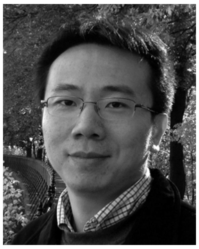
From 2006 to 2008, he was a research staff member with NEC Laboratories America, Princeton, NJ, USA. Since 2008, he has been a research staff member with the IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. His research interests include cloud computing and noSQL database.

Michel Hack received the Ingénieur Civil degree from the Ecole Nationale Supérieure des Télécommunications, Paris, France, in 1969, and the M.S. and Ph.D. degrees in Petri nets from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1972 and 1976, respectively.

He then joined IBM, Yorktown Heights, NY, USA, as a Research Staff Member. He has worked mostly on low-level system programming, debugging, and operating system issues and participated in the development of an experimental operating system for the S/370. He developed correctly rounding decimal/binary conversion algorithms, now encapsulated in a Z/architecture machine instruction. He is currently involved in clock synchronization projects for mainframe and blade center clusters, as well as in floating-point and high-speed arithmetic. He was a member of the IEEE P754 project (IEEE 754-2008 Floating-Point standard), and is now a member of the IEEE P1788 project on Interval Arithmetic.

Li Zhang received the B.S. degree in mathematics from Beijing University, Beijing, China, the M.S. degree in mathematics from Purdue University, West Lafayette, IN, USA, and the Ph.D. degree in operations research from Columbia University, New York, NY, USA, in 1997.

He is the manager of the System Analysis and Optimization Group, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He has coauthored more than 100 technical articles and over 30 patents. His research interests include control and performance analysis of computer systems, statistical techniques for traffic modeling and prediction, and scheduling and resource allocation in parallel and distributed systems. He also worked on measurement-based clock synchronization algorithms.



Ao Tang (S'01–M'07–SM'11) received the B.E. degree (Honors) in electronics engineering from Tsinghua University, Beijing, China, in 1999, and the M.S. and Ph.D. degrees in electrical engineering with a minor in applied and computational mathematics from the California Institute of Technology, Pasadena, CA, USA, in 2002 and 2006, respectively.

He is currently an Associate Professor with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA, where he conducts research on the control and optimization of engineering networks including communication networks, power networks, and on-chip networks.

Dr. Tang's recent awards include the Cornell Engineering School Michael Tien72 Excellence in Teaching Award in 2011, the Young Investigator Award from the Air Force Office of Scientific Research (AFOSR) in 2012, and the Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House in 2012.