Clock Synchronization: Open Problems in Theory and Practice

Christoph Lenzen, Thomas Locher, Philipp Sommer, and Roger Wattenhofer

Computer Engineering and Networks Laboratory TIK
ETH Zurich, 8092 Zurich, Switzerland
{lenzen,lochert,sommer,wattenhofer}@tik.ee.ethz.ch

Abstract. Clock synchronization is one of the most basic building blocks for many applications in computer science and engineering. The purpose of clock synchronization is to provide the constituent parts of a distributed system with a common notion of time. While the problem of synchronizing clocks in distributed systems has already received considerable attention from researchers and practitioners alike, we believe that there are many fascinating problems that remain unsolved. In this paper, we give a brief overview of previous work in this area, followed by a discussion of open clock synchronization problems in theory and practice.

1 Introduction

Although computer science is still a young discipline, certain signs of age are becoming apparent. As the discipline continues to prosper, computer scientists must become experts in some subjects—the days of the universal computer scientist seem to be coming to an end. One of the main dividing lines is between theory and practice (a.k.a. systems). Unfortunately, the gap between theory and practice even seems to be widening as both theory and practice are developing and advancing. One may worry that soon there will be little communication between the two camps because one side often considers questions or answers from the other as "not really relevant". This is unfortunate, since at least new trends, fundamental limits, or open problems should be of interest to the other camp.

In our research group, we try to find synergies between theory and practice, unfortunately with limited success. It is quite rare that the theorists in our group develop algorithms that have enough real-world advantages to justify an implementation. Likewise, building a system hardly ever reveals a beautiful theoretical problem. There are noteworthy exceptions: In the remainder of this invited paper we will discuss *clock synchronization*, a prototypical example that is inspiring from a theoretical as well as a practical point of view.

In clock synchronization, we are given a network of nodes that want to maintain a common notion of time. Having such a notion of time is important for many applications, in the Internet as well as, e.g., in wireless sensor networks. Each node may have its own hardware clock, which is not totally accurate, i.e., it experiences a certain variable clock drift. In order to ensure that the nodes agree more or less on the current time, the nodes must synchronize their drifting hardware clocks by perpetually exchanging messages containing information about their current state. It is easy to see that it is impossible to synchronize the clocks perfectly because the nodes never have current information about the other nodes' clock values due to fact that all messages arrive after an unknown and variable delay. Even if the message delays were always the same and the nodes knew this value exactly, they still could not synchronize the clocks perfectly because of the variable hardware clock drifts, i.e., a node cannot determine exactly how much another clock progressed since the last message arrived. This gives rise to a natural question, which is fundamental for many application domains: How well can nodes synchronize their clocks given that the clocks have variable but bounded drift, and given that messages have variable but bounded delay?

Naturally, one objective is to minimize the *clock skew* between any two nodes in the network, regardless of their relative distance in the network. Apart from minimizing this *global* skew, it is essential for several distributed applications that the clock skew between neighboring nodes is as small as possible. One could even think of applications where the global skew is not of great concern, but any node only needs to be well synchronized with its neighbors. Apparently this *local* skew that some neighboring nodes may experience depends on the (maximum) variance of the message delay and also on the (maximum) clock drift rate. Given that locally connected nodes can communicate directly, one may expect that the local skew depends *only* on these parameters. Surprisingly, this is not true! From a worst-case perspective, no matter what clock synchronization protocol is used, it is always possible that two neighboring nodes experience a clock skew that is a function of the network diameter. On the other hand, assuming random delays, there is a high chance that transmitting repeatedly yields better results. These results have an impact on the degree of synchronization that can be achieved in practical distributed systems. In the next section, we give a brief summary of known results, followed by discussions of various open clock synchronization problems in theory and practice.

2 Related Work

There is a large literature on clock synchronization in distributed systems, mostly focusing on (upper and lower) bounding the clock skews that can occur between any two nodes in the system (see, e.g., [19, 22, 25, 27]). A simple technique called *shifting* [19], where the local clock rates and the message delays are adjusted in order to produce indistinguishable executions, is often used to prove lower bounds. Using this technique it can be shown that a worst-case clock skew of $D/2$ cannot be avoided on any graph G of diameter D [2]. A stronger lower bound of $(1+\rho)D$ can be proved for all algorithms that must ensure that the clock values do not drift away from real time faster than the underlying hardware clocks, where ρ denotes the maximum clock drift rate [15].

Not surprisingly, large clock skews may occur between nodes that are far apart in the communicaton network, which means that these nodes experience a significant delay in their communication. An important question is how well the clocks of nodes that are close-by can be synchronized. In their seminal work that introduced the problem of synchronizing clocks of nodes that are close-by as accurately as possible, Fan and Lynch [8] showed that no algorithm can prevent a clock skew of $\Omega(\log_b D)$ between neighboring nodes, where $b \in \mathcal{O}((\log D)/\rho)$. The only constraint is that nodes are required to increase their clock values at a given minimum progress rate. Note that this constraint is quite natural because it ensures that all clocks steadily make progress. Subsequently, the base of the logarithm has been improved to $b \in \Theta(1/\rho)$, i.e., it has been shown that a clock skew of $\Omega(\log_{1/\rho} D)$ between some neighboring nodes cannot be avoided [15]. Moreover, if the progress rate of all clocks must always lie in an interval $[1 - \mathcal{O}(\rho), 1 + \mathcal{O}(\rho)]$, there are indistinguishable executions such that in one of these executions some neighboring nodes experience a clock skew of $\Omega(\log D)$ [15].

The clock synchronization algorithm by Srikanth and Toueg [27] guarantees a bound of $\mathcal{O}(D)$ on the clock skew between any two nodes at all times and is thus asymptotically optimal. The algorithm is further fault-tolerant and achieves an accuracy with respect to real time that is also optimal. However, their algorithm incurs a skew of $\Theta(D)$ between neighboring nodes in the worst case. The first algorithm, which is based on a technique called *blocking*, that guarantees a sublinear bound on the clock skew between neighboring nodes achieves a bound of $\mathcal{O}(\sqrt{\rho D})$ [17, 18]. The same technique can also be applied to dynamic settings, where nodes and edges can appear and disappear continuously [13]. In the static scenario, the upper bound has been improved to $\mathcal{O}(\log D)$ [14] and subsequently to $\mathcal{O}(\log_{1/\rho} D)$ [15], which matches the lower bound. The algorithm that achieves this tight bound also guarantees an optimal bound on the clock skew between any two nodes of $(1 + \rho)D$ plus a small term that depends on the frequency of communication. The additional term becomes zero as the frequency of communication tends to infinity.

There has also been a lot of practical work on clock synchronization for specific computing environments. For example, techniques to synchronize the nodes in wireless sensor networks have been studied extensively [7, 10, 20, 23]. It can be argued that in wireless sensor networks message delays are not only bounded, but also distributed (independently) at random. This assumption has a considerable impact on the achievable skew bounds: Under this assumption the skew between any two clocks can be upper bounded by $\tilde{\mathcal{O}}(\sqrt{D})$ w.h.p. [16]. The same work also shows that on most graphs at any point in time there is a constant probability that a clock skew of $\Omega(\sqrt{D})$ can be observed between some nodes. Moreover, some initial practical work on synchronizing nodes that are close-by particularly well has been carried out in the context of wireless sensor networks [26]. Clock synchronization has also been studied in other distributed systems such as the Internet [21] or systems-on-a-chip [9]. In processor design, where one seeks to control signal delays by means of placement and wiring (see, e.g., [12] and references therein), synchronizing devices that are close-by is essential. Furthermore, there has been a considerable amount of work on fault-tolerant clock synchronization for multiprocessor systems where processors communicate among each other via shared memory [5, 6, 11, 24].

3 Model: Worst Case vs. Reality

In theoretical work on clock synchronization, a distributed system is typically modeled as a connected graph $G = (V, E)$, where nodes represent computational devices and edges represent communication links. Communication is usually considered to be bidirectional (i.e., the edges are undirected), but it may also be reasonable to assume unidirectional communication channels. The nodes can communicate directly with their neighboring nodes by exchanging messages, which arrive at their destination after a certain *delay.* In general, a message delay consists of two parts: a constant, known delay and an additional variable delay ("jitter"). A common simplification is to assume that the message delay can be any value in the range $[0, 1]$ and the nodes do not know the normalized upper bound of 1. The second essential aspect of clock synchronization is how *clock drifts* are modeled. Typically, it is assumed that each node has a hardware clock with a bounded drift. A common way to model the clock drift is to define that all hardware clock rates are always in the interval $[1 - \rho, 1 + \rho]$ for a constant $\rho \in (0, 1)$.

Due to the assumption that any node can only read its hardware clock (and not modify it), each node also has a *logical clock* whose value depends on its hardware clock value and the information received from its neighboring nodes. A *clock synchronization algorithm* specifies how the logical clock value is adapted based on the hardware clock and the received information. The main objective of the algorithm is to ensure that the clock skews, both between distant nodes and neighboring nodes, are always as small as possible. Theoretical work typically considers worst-case scenarios, where hardware clock rates and message delays happen in a way that maximizes clock skews.

Ideally, an algorithm that guarantees strong worst-case bounds is also a suitable candidate to maintain tightly synchronized clocks in real-world systems. However, the models employed in theoretical work are often too pessimistic compared to the situations that occur in practice. Clock drift rates change only gradually depending on factors such as the ambient temperature or the supply voltage. Even when operating a system in harsh environments, clock drifts will not change arbitrarily during a single synchronization interval. Similarly, assuming that an adversary takes control of variations in message delays does not reflect reality well. Using sophisticated mechanisms, e.g., timestamping at the MAC layer [20], the effect of variances in the delay can be mitigated up to a few clock ticks. This remaining uncertainty seems to be better captured by a probabilistic rather than a worst-case approach. A more practice-oriented model must take such considerations into account.

4 Dynamic Networks

As briefly outlined in Section 2, it is well understood what the best possible bounds on the worst-case clock skews are that any clock synchronization algorithm can guarantee in static networks. However, in practice distributed systems are often dynamic in the sense that both devices and communication channels can appear and disappear. Thus, the static theoretical model is too simplistic for many practical applications. A more appropriate model must allow for on-going changes to the network topology in order to bridge this chasm between theory and practice.

The necessity of taking network dynamics into account has also been pointed out in [13]. Of course, there are fundamental limits to the degree of synchronization that can be achieved in a completely dynamic setting, where nodes and edges can appear and disappear in an arbitrary manner: According to the lower bound for static networks, a clock skew of roughly D can occur on any graph between some nodes v and w. By adding the edge $\{v, w\}$ between these two nodes, the network suddenly experiences a worst-case clock skew of D between neighboring nodes. Obviously, this situation cannot be avoided. The problem is that a newly added edge may always cause a large clock skew between the two nodes that it connects. However, once the nodes are aware of this situation, they can react to it and reduce the clock skew on this particular edge over time. This means that while we cannot get a sublinear bound on the clock skew between nodes that are connected through a new edge, we can in fact guarantee a better bound for edges that existed for a certain period of time.

It is not known whether the same asymptotic bounds as in the static case can be achieved. The algorithm presented in [13] guarantees an upper bound of $\mathcal{O}(\sqrt{\rho n})$ (where $n := |V|$) on the worst-case clock skew between any two neighboring nodes v and w provided that the edge $\{v, w\}$ has been part of the network for $\Omega(\sqrt{n/\rho})$ time. This bound is exponentially weaker than the bound of $\mathcal{O}(\log_{1/\rho} n)$ in the static setting. It is an interesting open problem whether the same asymptotic bound can be achieved in dynamic networks. Not surprisingly, the proof techniques that are used to prove the bound of $\mathcal{O}(\log_{1/\rho} n)$ cannot be used directly for dynamic graphs. An intriguing aspect of this problem is, however, that these proof techniques cannot even deal with the *removal* of edges. This is quite counterintuitive as one might think that removing edges cannot cause problems because the adjacent nodes are not neighbors anymore and, given the increased distance between these nodes, their clock values are allowed to deviate more. The problem is that the proof relies on the fact that a node v always has a neighbor w that forces v to increase its clock value quickly if v is on the verge of violating the skew bounds. In the dynamic setting, this neighbor w may leave the system at a critical moment. The goal is to show that this is indeed a real problem, by proving a new lower bound, or to prove that the clock skews can nevertheless be kept (asymptotically) as small as in the static case.

This is an open theoretical problem that again considers the skew bounds in the worst case. The dynamics in real-world networks may be benign in comparison and therefore different approaches may be employed. The right choice will likely depend on the nature of the considered distributed system. Coping with dynamics in various (practical) types of distributed systems is another potential direction for future research.

5 Fault-Tolerance

There has been substantial work on fault-tolerant clock synchronization in single-hop networks (see, e.g., references in [28]). A lot of progress has been made over the years on the *digital clock synchronization* problem in shared memory models [1, 4–6, 24], where nodes try to maintain a synchronized counter (digital clock) with certain progress guarantees in spite of crash, Byzantine, or transient failures. In the message passing world, such counters are known as (fault-tolerant) synchronizers, which provide a weaker form of timing information than a "regular" clock, in particular if the diameter of the graph is not constant. Moreover, quite frequently some kind of a priori synchronization of the system is assumed, such as synchronous rounds, bounded message delays, or periodic "beats". Applications for this kind of synchronization algorithms can be found e.g. in fault-tolerant chip design [9] or multiprocessor systems.

To the best of our knowledge, little is known about fault-tolerant clock synchronization in multi-hop environments. From the theory side, it is easy to observe that many protocols assuming full connectivity can be generalized to multi-hop scenarios if the network satisfies certain connectivity constraints in order to handle, e.g., crash or Byzantine failures. This approach might however be unsatisfactory if small local skews are desired, and it gives little information about transient or probabilistic faults. In practice, the problem is mainly tackled in a pragmatic manner. In wireless sensor networks, for example, people may rely on MAC-layer protocols that handle message retransmissions, or they simply accept decreasing synchronization quality in the presence of high message loss rates.

The challenging problem of finding fault-tolerant clock synchronization mechanisms for multi-hop networks merits further attention for various reasons. First, studying fault-tolerant clock synchronization may lead to valuable (theoretical) insights: The problem asks for error detection and error handling techniques that do not interfere with the ability of algorithms to achieve a high precision of clock values, while at the same time the synchronization offered might be helpful in doing so. It is important to understand to what degree synchronization can be maintained given specific failure models and connectivity constraints. There are many interesting open problems that can be addressed: If nodes are faulty with certain probabilities, how dense do carefully crafted graphs have to be to permit reliable synchronization? What degree of resilience to Byzantine faults can be guaranteed on a given graph? In general, which trade-offs exist between achievable accuracy of timing information and resilience to faults? Moreover, since access to a consistent, accurate time is a basic service and nodes in real-world networks are often not fully connected, this topic has a significant practical relevance.

6 Energy Efficiency vs. Accuracy

When developing applications for wireless sensor networks, energy efficiency is an important issue. Sensor nodes should be able to operate unattended for many years, even when running on small batteries. In order to meet the requirements of the application, as much energy as possible has to be conserved by operating the microcontroller and the radio chip in the power-save mode whenever possible. One approach to reduce the energy consumption of sensor nodes is to arrange periodic rendezvous schemes: Neighboring nodes wake up for a short moment to exchange messages and immediately return to sleep afterwards. The better the clock synchronization is between neighboring nodes, the more energy is saved by shortening the necessary guard intervals before and after the designated rendezvous point. However, accurately synchronized clocks can only be achieved by exchanging periodic synchronization messages, which also requires energy.

Furthermore, the synchronization error will grow at least with the square-root of the distance to the reference node due to the jitter in the message delay [16]. To reduce this effect, one can try to reduce the jitter itself. By means of MAC layer timestamping, it can be cut down at best to the hardware clock granularity $1/f$, where f denotes the frequency at which the clock operates. Thus, reducing the jitter means increasing the time resolution, which in turn requires that the hardware clock operates at a higher frequency. The power consumption P of the clock circuits is given by $P \sim C_L V^2 f$, where C_L is the load capacitance, V is the supply voltage, and f is the clock frequency [3]. Therefore, the power consumption will increase linearly with the clock frequency. Taking into account that synchronization quality decreases (at least) linearly when reducing the frequency of message exchange, it might be best to balance the guard time and the time it takes to switch on the radio and send or receive a message, and maximize the time between messages with these parameters fixed.

Another option to abate clock skews is to circumvent the problem that the jitter and thus the inaccuracy of the clock values increases over (long) paths in the network by utilizing the Global Positioning System (GPS). GPS satellites orbiting the earth periodically send their position information together with the current timestamp of their atomic clock down to earth. If a sensor node is equipped with a GPS receiver, it can obtain the high precision timing information included in the GPS messages. This way, the problem of network-wide multi-hop clock synchronization reduces to the single-hop case. Furthermore, external time synchronization using GPS eliminates the need for exchanging synchronization messages over the sensor network. However, this is only a viable solution for applications where a line of sight to the GPS satellites is available.

State-of-the-art GPS devices provide an accuracy within a few nanoseconds. This is a significant improvement compared to the accuracy of common sensor node platforms (e.g., the Mica2 motes), which is in the microsecond range. However, the improved timing accuracy of GPS-based time synchronization solutions comes at the cost of an increased hardware complexity and higher energy consumption. In particular, in order to achieve maximal precision, a hardware clock operating at a sufficiently high frequency to provide the necessary time resolution is mandatory. Even though the size, cost, and energy consumption of modern GPS receivers is continually decreasing, it is often not feasible to equip every node in the network with a GPS receiver. Instead, only one or a few nodes, the so called reference nodes, are equipped with a GPS receiver and synchronized to UTC. Traditional time synchronization protocols for wireless sensor networks must then be used to synchronize the rest of the network to the reference nodes.

It remains an open problem to build a small, cheap, and low-power sensor node that is able to synchronize its clock to the accurate time pulses provided by the GPS satellites. Moreover, it is unknown if the effects considered above determine the minimum energy consumption of sensor nodes. If this is indeed the case, protocol implementations are required that provide an optimal trade-off between energy efficiency and hardware costs.

7 New Applications

Another interesting direction for future future research is finding new applications that put the theoretical knowledge in this field into practice. In light of the ongoing progress of recent years, two ideas for prospective applications came to our mind, both of which particularly aim at exploiting the better understanding of local skew that has been gained in the past few years.

Our first suggestion is to use clock synchronization principles to achieve other forms of coordination, e.g., coordination of movement or formation. There are various coordination tasks where clock synchronization mechanisms may be employed: Maybe some robots intend to move in a line, always maintaining identical distances, or a traffic jam can be avoided if distances between cars are balanced. Other goals might be the coordination of helicopters or, more generally, any kind of swarm trying to maintain a certain formation based only on local distance information. In all these cases the agents may experience *drifts*, because they are not able or willing to control their speeds perfectly, or subject to disturbance by other forces. Furthermore, their information on their neighbors' positions could be outdated and/or inaccurate, which is exactly the effect of *message delays* in clock synchronization. In summary, the underlying model of a coordination problem might be quite similar to the model presented in Section 3, raising the hope that results from clock synchronization could be applicable to such problems.

The second idea refers to chip design. Traditionally, synchronous circuits are controlled by a clock signal dissipated from a single source to all logical gates by means of a clock tree. This clock signal is used to determine when it is safe to advance to the next computational step by guaranteeing sufficient time between clock pulses for gates to switch states and signals to propagate. Therefore, a *local* synchronization guarantee between directly connected gates is needed. Even if the clock tree is well designed, we will observe a stretch in the distance compared to the communication graph of the logical gates. In the worst case, this stretch could be in the order of the diameter. Certainly, the stretch will grow with increasing chip size also in practice because different paths of computations are joined at some point in the chip logic. As a result, weak synchronization guarantees limit the frequency at which a chip can safely operate. We propose to improve this situation by means of a distributed clock generation scheme. Hopefully, by equipping a chip with, e.g., a grid of time sources, running a clock synchronization algorithm with strong local skew bounds on this grid, and dissipating the clock signal only locally by means of clock trees enables the construction of large chips without incurring a decline in the operating frequency. This scheme would come at the expense of additional clocks and chip logic for the synchronization algorithm, which could be compensated for by increasing the area of the chip.

We hope that the presented open problems have stimulated the reader's interest in distributed clock synchronization as a vivid and evolving research area.

References

1. M. Ben-Or, D. Dolev, and E. N. Hoch. Fast Self-Stabilizing Byzantine Tolerant Digital Clock Synchronization. In *Proc. 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 385–394, 2008.
2. S. Biaz and J. Lundelius Welch. Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Information Processing Letters*, 80(3):151–157, 2001.
3. A. P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power Digital CMOS Design. *IEEE Journal of Solid State Circuits*, 27(4):473–484, 1992.
4. D. Dolev, J. Y. Halpern, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching Approximate Agreement in the Presence of Faults. *Journal of the ACM*, 33(3):499–516, 1986.
5. S. Dolev. Possible and Impossible Self-stabilizing Digital Clock Synchronization in General Graphs. *Journal of Real-Time Systems*, 12(1):95–107, 1997.
6. S. Dolev and J. Lundelius Welch. Wait-free Clock Synchronization. *Algorithmica*, 18(4):486–511, 1997.
7. J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization Using Reference Broadcasts. *ACM SIGOPS Operating Systems Review*, 36:147–163, 2002.
8. R. Fan and N. Lynch. Gradient Clock Synchronization. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 320–327, 2004.
9. M. Függer, U. Schmid, G. Fuchs, and G. Kempf. Fault-Tolerant Distributed Clock Generation in VLSI Systems-on-Chip. In *Proc. 6th European Dependable Computing Conference (EDCC-6)*, pages 87–96, 2006.
10. S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-Sync Protocol for Sensor Networks. In *Proc. 1st ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 138–149, 2003.
11. S. T. Huang and T. J. Liu. Four-state Stabilizing Phase Clock for Unidirectional Rings of Odd Size. *Information Processing Letters*, 65(6):325–329, 1998.
12. B. Korte, D. Rautenbach, and J. Vygen. BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip. *Proceedings of the IEEE*, 95(3):555–572, 2007.
13. F. Kuhn, T. Locher, and R. Oshman. Gradient Clock Synchronization in Dynamic Networks. In *Proc. 21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 270–279, 2009.

14. C. Lenzen, T. Locher, and R. Wattenhofer. Clock Synchronization with Bounded Global and Local Skew. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 500–510, 2008.
15. C. Lenzen, T. Locher, and R. Wattenhofer. Tight Bounds for Clock Synchronization. In *Proc. 28rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 46–55, 2009.
16. C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal Clock Synchronization in Networks. In *Proc. 7th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2009.
17. T. Locher. *Foundations of Aggregation and Synchronization in Distributed Systems.* PhD thesis, ETH Zurich, 2009.
18. T. Locher and R. Wattenhofer. Oblivious Gradient Clock Synchronization. In *Proc. 20th International Symposium on Distributed Computing (DISC)*, pages 520–533, 2006.
19. J. Lundelius Welch and N. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2/3):190–204, 1984.
20. M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *Proc. 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 39–49, 2004.
21. D. Mills. Internet Time Synchronization: the Network Time Protocol. *IEEE Transactions on Communications*, 39:1482–1493, 1991.
22. R. Ostrovsky and B. Patt-Shamir. Optimal and Efficient Clock Synchronization under Drifting Clocks. In *Proc. 18th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 400–414, 1999.
23. S. PalChaudhuri, A. K. Saha, and D. B. Johnson. Adaptive Clock Synchronization in Sensor Networks. In *Proc. 3rd ACM/IEEE International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 340–348, 2004.
24. M. Papatriantafilou and P. Tsigas. On Self-stabilizing Wait-free Clock Synchronization. *Parallel Processing Letters*, 7(3):321–328, 1997.
25. B. Patt-Shamir and S. Rajsbaum. A Theory of Clock Synchronization. In *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 810–819, 1994.
26. P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Proc. 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 37–48, 2009.
27. T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, 1987.
28. K. Sun, P. Ning, and C. Wang. *Secure Localization and Time Synchronization for Wireless Sensor and Ad Hoc Networks*, chapter Secure and Resilient Time Synchronization in Wireless Sensor Networks. Springer US, 2007.