

A Light Weight Method for Maintaining Clock Synchronization for Networked Systems

David Salyers, Aaron Striegel, Christian Poellabauer
 Department of Computer Science and Engineering
 University of Notre Dame
 Notre Dame, IN 46656 USA
 Email: {dsalyers, striegel, cpoellab}@nd.edu

Abstract—Maintaining synchronization of clocks between wireless systems is a well known problem of which significant research has been performed. This has led to a variety of methods introduced to maintain clock synchronization. Typically, works are heavy weight in that they require constant communication between systems in order to maintain clock synchronization on the order of microseconds. Unfortunately, for many applications the cost of constant communication to ensure clock synchronization is neither desirable nor acceptable. Additionally, for applications such as link state routing, delay measurements and quality of service measurements, clock synchronization on the order of microseconds is not necessary, and synchronization on the order of milliseconds is sufficient. Thus, in this work we present a light weight technique for correcting for clock drift between systems that will allow for millisecond accuracy during long periods of time while requiring no special hardware nor constant communication between systems. Experimental studies of the measured delay between two systems are performed, showing that with a training period, clocks can remain synchronized within a few milliseconds over long periods of time.

I. INTRODUCTION

Significant work has been performed in clock synchronization between wireless systems [1], [2], [3], [4], where the clocks can achieve a synchronization on the order of microseconds. However, due to clock drift the synchronization between the two systems will soon be lost. Hence, other works (such as [5], [6]) focus on being able to achieve and maintain synchronization between systems. They maintain synchronization by requiring constant communication between the systems to be synchronized, or with the use of specialized hardware.

Critically, these heavy weight synchronization mechanisms may not be desirable or acceptable in many situations. For example, consider a wireless network experiment where a researcher wishes to track the delay of packets as they traverse the network. In this case, the introduction of additional data and/or packets can interfere with the results as bandwidth is consumed in order to maintain synchronization between the clocks. Additionally, microsecond accuracy is not needed, as millisecond accuracy is generally sufficient for many applications (research studying packet delay, QoS applications, ect.) on a wireless network. Thus, in this work we introduce a simple, light weight method to maintain clock

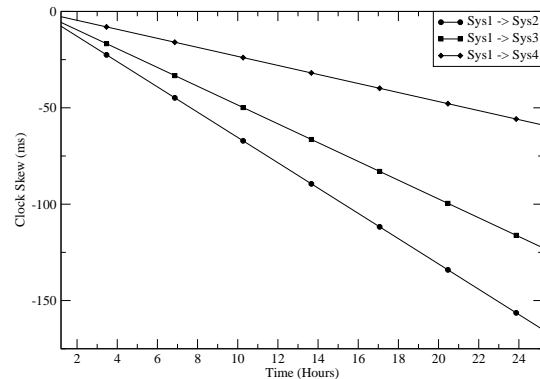


Fig. 1. Graph of clock skew between two systems over time.

synchronization, without requiring additional hardware, on millisecond accuracy over long periods of time only requiring resynchronization after hours have passed.

Typically, clocks in the wireless systems will differ for two reasons. The first reason is clock skew, which is simply the difference between two clocks. While clock skew is relatively easy to correct for, the second reason clocks differ is clock drift. Clock drift refers to the fact that different clocks will count time at slightly different rates, resulting in varying clock skews. While clock skew can be corrected for by synchronization, clock drift can quickly cause the clocks to become unsynchronized.

In order to motivate our work, the following experiment is run. Two identical systems¹, System 1 and System 2, were directly connected via an ethernet crossover cable. Next, the systems were then had their clocks synchronized, using the first system as the ground truth for time. Finally packets were sent from System 1 to System 2 over a period of 24 hours with System 1 recording the time the packet was sent and System 2 recording the time the packet was received at a rate of 200 packets per second. The experiment was then repeated by connecting System 1 with System 3 and System 4, which were also identical. As is seen in Fig. 1, each system has a different clock drift as demonstrated by the varied slopes of

This research was supported in part by the National Science Foundation through the grants CNS03-47392 and CNS05-45899.

¹IBM/Lenovo X41, Fedora Core 6, CPU locked at 1.6 Ghz

the three different pairings of systems.

The results from the above experiment provided the inspiration for our work in that for all three pairs of systems, the change in clock skew is almost a straight line, implying that linear regression can be used to remove the clock drift, and maintain synchronization between systems. While linear regression may not be able to determine an exact drift ratio between the sender and receiver (due to fluctuations in operation of the hardware); it can eliminate the majority of the effect from clock drift.

II. A METHOD FOR ACCOUNTING FOR CLOCK DRIFT IN EXPERIMENTAL DATA

In this section we discuss our method for accounting for clock drift between synchronized systems. The overall method can be separated into the following steps:

- 1) Select a host to be used as the ground truth time.
- 2) Transmit UDP packets from the ground truth system to each of the clients at a slow rate in order to obtain a training set for the linear regression. The length of the training set is directly proportional to the amount of time between synchronizations of the system.
- 3) Perform linear regression to estimate clock drift between the systems.
- 4) Perform a standard clock synchronization (such as [2]) to eliminate the clock skew between the two systems.
- 5) As clock drift can not be estimated perfectly, resynchronization will need to be performed at a periodic basis (although significantly less frequently) in order to maintain millisecond accuracy.

Step 1: Selecting a ground truth system: Given a set of systems, S , to be synchronized, the ground truth system can be any system in the set. However, the system to be used as the ground truth system should have an accurate timer, preferably a TSC (Time Stamp Counter) register. Alternatively, synchronization of the ground truth system with an external source such as GPS or NTP is also acceptable.

Step 2: Run Timing Experiment: In order to obtain the training data between systems, the systems can be connected either via wireless or via Ethernet. Each method has advantages and drawbacks. For wireless, the advantages lie in the fact that all systems can be connected directly to the ground truth for training purposes. However, there are significant drawbacks in wireless transmission. The first major drawback is that the wireless medium is an inherently lossy network. Second, wireless transfers are subject to higher levels of deviation in their delivery time². Finally, it is very important that the wireless network be contention free, without other stations competing for transmission.

By connecting the hosts via Ethernet for the training period, the researcher can minimize the effect of packet loss. However, in order to connect large number of nodes, multiple switches/routers will need to be added to the setup in order to synchronize the hosts at once. This can introduce additional errors in the clock drift estimation as the end-to-end delay

²This can be significantly mitigated by using broadcasts, which are not retried.

between systems will fluctuate more. Alternately, clock drift training can be done in rounds; where each individual host or size n subset of hosts can be connected to the ground truth system.

As each host is connected to the ground truth host a UDP stream will be sent from the ground truth host to the connected host. This UDP stream should be a CBR stream of TSC timestamps (which the receiver will record along with their own timestamps) at a low data rate, in order to minimize the load on the network and the receiving host. Finally, duration of the training period is directly related to how accurate of an estimate is required. For example, in our experimental studies we find that a training period of one hour to provide a close estimation of the clock drift within $0.2\mu s$.

Step 3: Perform Linear Regression: Once the stream has been transmitted for the desired amount of time, linear regression is used to derive an estimate for clock drift m_{1-2} and initial clock skew b_{1-2} can be used in Eq. 1.

$$t_{x_1} = m_{1-2} \times t_{x_2} + b_{1-2} \quad (1)$$

If multiple rounds of tests need to be performed to obtain the required training data from each system, it is important to note that each system does not have to be connected directly with the ground truth system. Once the relationship between two systems is discovered, then each of those systems can then be used in the next round to discover the relationship with another system. This allows for $\log_2(n)$ training sessions. Next, the linear equations for the different hosts can be combined to relate each host to the ground truth. For example consider three systems, x_1, x_2 , and x_3 . Systems x_1 and x_2 have been synchronized via the Eq. 1, and systems x_2 and x_3 have been synchronized via the equation:

$$t_{x_2} = m_{2-3} \times t_{x_3} + b_{2-3} \quad (2)$$

In order to relate x_3 to x_1 we can use substitution to Eq. 2 into Eq. 1 to derive the following:

$$t_{x_1} = m_{1-2} \times (m_{2-3} \times t_{x_3} + b_{2-3}) + b_{1-2} \quad (3)$$

which can be simplified to:

$$t_{x_1} = m_{1-3} \times t_{x_3} + b_{1-3} \quad (4)$$

where $m_{1-3} = m_{1-2} \times m_{2-3}$ and $b_{1-3} = m_{1-2} \times b_{2-3} + b_{1-2}$. The drawback to this method is that the error rates of the different measurements are combined. Thus, if the Eq. 1 resulted in a standard deviation of error of $\pm 0.5ms$, and Eq. 2 had a standard deviation of error of $\pm 0.6ms$, then the expected standard deviation of error between these two measurements in the worst case is $\pm 1.1ms$. Fortunately, experiments show that the worst case is not likely.

Step 4: Synchronize each system with the ground truth system: Once the clock drift between each system has been determined, the final step is to eliminate the clock skew between the systems. Thus, the systems are synchronized with the ground-truth system. The specific method does not specifically matter, however a method such as [2], is preferable as multiple systems can be synchronized concurrently and the

method does not require the addition of special hardware. Once the clock skew has been eliminated, the b values of the linear expression can be set to 0 in order to indicate an initial clock skew of 0.

If the clock drift is estimated perfectly, then no further synchronization would ever need to be performed. Unfortunately, it is not possible to estimate the clock drift with 100% accuracy. There are many reasons for this, including slight variations in a CPU's clock cycle rate, variations in packet delivery due to OS overhead, ect. In our experiments it is shown that clock drift can be estimated within $0.2\mu s$. This puts a maximum error of $17.200 ms$ between two clocks after a one day period (with experimental results showing significantly better performance than the worst case). Thus, in order to maintain sub-millisecond accuracy the systems will need to be resynchronized. *It is very important to note, that the most costly aspect of this timing method, linear regression, does not need to be performed again.* While the method presented in this paper does not completely remove the need for communication between the systems in order to maintain synchronicity, it does considerably reduce the amount of communication in order to maintain synchronization by allowing for longer periods of time between synchronizations.

III. EXPERIMENTATION

In our experiment, two identical laptops are used (same as used in the initial experiment), both of which had been booted over in the hour prior to the start of the experiment, such that each system was warm. For time synchronization purposes the two systems were connected directly via a cross over cable. Each experiment consisted of 1500 byte UDP packets sent at a rate of 200 pps for the duration of the experiment. The first set of experiments investigates the length of time data needs to be transferred in order to obtain an accurate measurement for time drift.

In order to determine the appropriate run length such that the appropriate function for time synchronization can occur we ran an experiment for five days. The five day set was split into five single day single sets. Then a portion of each day test set is used to develop the linear equation used for the remainder of the data. In Table I and Table II, the average error and standard deviation for each of the training sizes on the rest of the data set is shown. As can be seen, for a 24 hour experiment a training set of at least an hour needs to be used in order to provide an accurate linear estimation of the clock drift. While training set times of less than an hour can provide an accurate estimate, the variation in the error for the different data sets is significantly greater. *It is also important to note that the training only needs to be done once to establish an estimate of the relative drift between the two systems.* Once the relationship has been established the systems only need to be resynchronized when the worst case cumulative error grows beyond the acceptable error bound.

For a shorter experiment length, a shorter training set can be used to provide an accurate measurement of clock drift, as shown in Table III. Here it can be seen that a training set of only one minute is required to obtain a measurement that is

Resync. Time	1 Day Test - Day 1	
	Avg. Err.	Std. Dev.
1h - Train	2.1ms	1.8ms
1h - Train, 1h - Resync	0.1ms	1.0ms
1h - Resync	4.0ms	3.7ms

TABLE IV
STANDARD DEVIATION OF CLOCK SKEW AND AVERAGE ERROR BY
SYNCHRONIZATION METHOD FOR A TEST CASE OF 1 DAY.

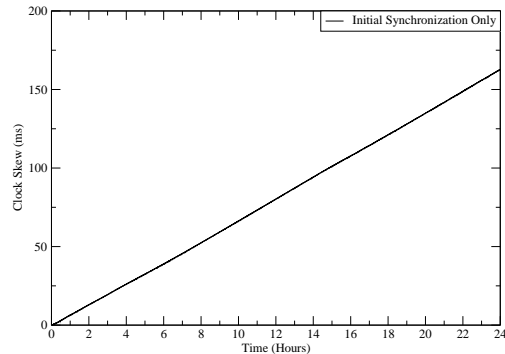


Fig. 2. Clock skew growth with initial synchronization.

on average less than 1ms from the expected value. Training sets significantly longer the one minute do not significantly improve the clock drift estimation. The results in Table III, in addition to the results in Table I and Table II indicate that even with the linear regression of the clock drift, that as more time passes the error and the standard deviation of the error becomes greater. Thus, for the best results the clocks will have to be resynchronized in order to maintain an acceptable margin of error.

In Table IV, the average error and the standard deviation are shown under various synchronization scenarios. By resynchronizing the clocks every hour a sub-millisecond average error is possible in addition to a sub-millisecond standard deviation. Also shown as the last entry in Table IV, is the average error and standard when just resynchronization every one hour is used. In Fig. 2 the clock skew of the TSC register between two identical systems increases to over 150 milliseconds in one day. In Figs. 3 and 4, the clock skew of the TSC register is shown with the following clock drift schemes:

- *1HT*: Two hours of training are used with no resynchronization.
- *1HT-1R*: Thirty minutes of training is used with resynchronization occurring every hour.
- *1R*: No training with a one hour resynchronization period.

The best performing method is *1H - 1R*, where one hour of training is used with resynchronization occurring every hour. This method allows a close estimation of the clock drift between the two systems with the hourly resynchronization limiting the cumulative error from the estimation to only a few tenths of a millisecond. Moreover, training by itself (*1HT*) still has a better overall estimation at the end of the 24 hour

Test Day	1s		5s		10s		30s		1m	
	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.
1	-76.2ms	43.2ms	17.9ms	11.1ms	19.7ms	12.2ms	14.3ms	9.0ms	14.1ms	8.9ms
2	39.1ms	22.6ms	-11.1ms	6.3ms	0.5ms	1.3ms	2.7ms	2.1ms	11.1ms	6.7ms
3	-36.2ms	21.7ms	-5.5ms	4.1ms	4.3ms	3.5ms	-16.9ms	10.6ms	-26.2ms	15.9ms
4	-14.1ms	8.2ms	2.6ms	1.9ms	4.0ms	2.6ms	1.3ms	1.4ms	14.3ms	8.4ms
5	-41.0ms	23.3ms	-3.1ms	2.0ms	1.9ms	1.9ms	177.6ms	102.9ms	0.2ms	1.4ms

TABLE I

STANDARD DEVIATION OF CLOCK SKEW AND AVERAGE ERROR BY TRAINING SET LENGTH FOR A TEST LENGTH OF ONE DAY.

Test Day	5m		10m		30m		1h		2h	
	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.
1	18.6ms	11.4ms	19.5ms	11.9ms	10.2ms	6.4ms	2.1ms	1.8ms	1.1ms	1.3ms
2	2.9ms	2.3ms	0.5ms	1.3ms	-1.7ms	1.3ms	-2.9ms	1.9ms	-2.2ms	1.6ms
3	-6.2ms	4.5ms	-5.6ms	4.1ms	-5.8ms	4.1ms	-5.8ms	4.2ms	-5.7ms	4.2ms
4	2.3ms	1.8ms	2.9ms	1.6ms	1.6ms	1.5ms	1.3ms	1.4ms	1.0ms	1.3ms
5	2.6ms	2.3ms	2.6ms	2.3ms	3.3ms	2.6ms	0.9ms	1.5ms	0.5ms	1.5ms

TABLE II

STANDARD DEVIATION OF CLOCK SKEW AND AVERAGE ERROR BY TRAINING SET LENGTH FOR A TEST LENGTH OF ONE DAY.

Test Hour	30s		1m		5m		10m		30m	
	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.	Avg. Err.	Std. Dev.
1	0.3ms	5.2ms	0.3ms	5.3ms	0.5ms	5.4ms	0.6ms	5.7ms	0.4ms	7.3ms
2	0.2ms	5.2ms	0.5ms	5.2ms	0.2ms	5.2ms	0.1ms	5.2ms	0.1ms	5.1ms
3	-0.4ms	5.0ms	-0.8ms	5.0ms	0.0ms	5.1ms	0.0ms	5.1ms	0.0ms	5.1ms
4	0.0ms	5.1ms	0.5ms	5.2ms	0.0ms	5.1ms	0.0ms	5.1ms	0.0ms	5.1ms
5	7.0ms	8.5ms	0.0ms	5.2ms	0.0ms	5.2ms	0.0ms	5.2ms	0.0ms	5.1ms

TABLE III

STANDARD DEVIATION OF CLOCK SKEW AND AVERAGE ERROR BY TRAINING SET LENGTH FOR A TEST LENGTH OF ONE HOUR.

period than when synchronization is used every hour. In order for resynchronization alone to give similar performance similar to $1H - 1R$, resynchronization would need to be performed every two minutes introducing significant overhead.

Limitations of using Linear Regression: In order to determine the stability of the drift estimation, a linear regression was performed on every sub-set of one hour of the five day test set. It was found that the standard deviation of the drift measurement was less than 0.1 microseconds with a difference of 0.2 microseconds between the minimum drift and maximum drift. There are two primary factors to this deviation. First, processors are not exact in their clock rates, and they can fluctuate according to temperature and other environmental factors. The second source of error which causes training sets to be longer is that the end-to-end delay between systems can fluctuate on the order of a few microseconds during each transmission. Thus, many samples are needed to find the average case, such that the minor fluctuations in end-to-end delay do not significantly affect the overall result. Fortunately, even with these errors we show a 25X increase in accuracy from the standard clock drift of roughly $5\mu s$ per second which occurred normally.

IV. RELATED WORK

In [6], Pásztor and Veitch demonstrate the effectiveness of using the TSC (Time Stamp Counter) for accurate measurement of time which is found on most modern processors. It

is typically a 64bit register that indicates the number of clock cycles that have passed since system start-up. The TSC register has two main benefits, namely stability and low read overhead (10 - 20 μs). The main shortcoming with their work is that the cycle rate of the processor needs to be accurately measured, which they achieve by adding DAG cards to their hosts. Our method does not require an accurate measurement of a systems cycle rate, as differences from the expected cycle rate and the actual clock rate will be eliminated by the linear regression formula.

Other works focus on the synchronization of clocks between various systems. In [2], Maróti et al. propose a method of clock synchronization between receivers in a wireless network utilizing broadcasts. The method is different in that the work does not synchronize the receivers with the sending node, but the receivers are cooperatively synchronized. In [3], PalChaudhuri, Saha, and Johnson present a method to synchronize clocks in a multi-hop wireless network. Notably, each of the two works could be substituted for the synchronization step of our protocol.

In [4], Ganeriwal, Kumar, and Srivastava present a method for accurately synchronizing the clocks of motes in a wireless ad-hoc sensor network. They show their method to achieve synchronization within $20\mu s$ on average and $50\mu s$ in the worst case. Iwanicki, van Steen, and Voulgaris address scalability in the context of a large-scale distributed system through a gossip-based scheme, Gossiping Time Protocol (GTP) in [7].

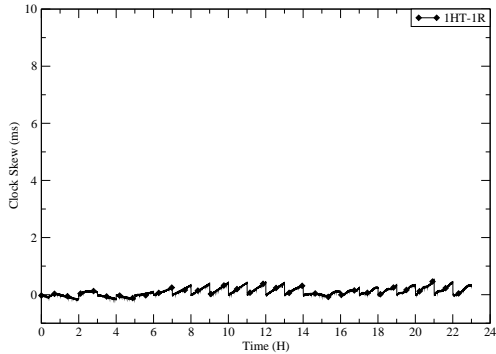


Fig. 4. Clock skew with clock drift correction, hourly resynchronization.

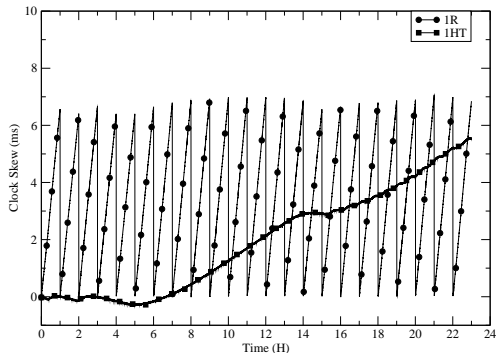


Fig. 3. Clock skew growths with resynchronization or clock drift correction.

As with other synchronization protocols, the synchronization does not specifically account for clock drift and only corrects clock skew.

The most closely related work to ours is the work in [5]. In [5], Aweya et al. investigate the prospect of using a linear process model in order to determine clock drift. The core premise of this paper is that standard linear regression by itself is not enough to correct for clock drift. Thus, a Phase Locked Loop is used in order to correct the receivers time according to time stamps received from the sender. This is done on a packet-by-packet basis between the sender and receiver by utilizing Discounted Least-Squares and Direct Smoothing in order to maintain synchronization. Using simulations they show the effectiveness of their protocol at maintaining synchronization under a variety of initial clock skews and error rates. Our work is different in that we present experimental studies demonstrating the effectiveness of linear regression on actual hardware. Also, we find that simple linear regression is indeed sufficient for maintaining millisecond synchronization between systems over large periods of time.

In [8], Veitch, Babu, and Pásztor, continuously monitor the time stamps between systems and use a sliding window protocol in order to maintain clock synchronization between the systems. This work is also similar to all traffic based measurement protocols in that large periods of abnormally

high delay may substantially affect the adjustment of the receiver's clock, which is not desirable for research purposes.

In [9], Tulone presents a method for maintaining clock synchronization between wireless motes during periods of time when clock synchronization protocols cannot be run during a brief period of time. The author's method allows for error bounds during these periods of non-synchronization to be divided in half. Our method is similar as we are concerned with the periods of time between clock synchronization. While Tulone's work will give a close estimate of the true clock during short periods of non-synchronization, during larger periods significant error will still be present. Another practical problem in [9] is that it will be difficult to ensure that each device has one clock counting faster than the true time and one clock counting slower.

V. CONCLUSIONS

In conclusion, this work introduced a simple method to account for the various amounts of clock drift between systems. While, there have been many other works that correct for clock drift, they typically require the addition of special hardware and/or consistent communication between the hosts. On the other hand, our method does not require the addition on any specialized hardware nor does it require constant communication between systems to maintain synchronization.

While the linear regression is expensive and time consuming to perform, in only needs to be performed once provided the CPU cycle rate stays stable. Additionally, once the clock drift estimate between two systems is derived, that synchronization will only need to be performed once every couple of hours in order to maintain millisecond accuracy. This is a significant improvement over simple synchronization techniques which would need to be performed every few minutes in order to maintain a similar level of synchronization.

REFERENCES

- [1] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, pp. 281–323, May 2005.
- [2] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of SenSys 2004*, New York, NY, USA, 2004, pp. 39–49, ACM Press.
- [3] S. PalChaudhuri, A. K. Saha, and D. B. Johnson, "Adaptive clock synchronization in sensor networks," in *Proceedings of IPSN 2004*, New York, NY, USA, 2004, pp. 340–348, ACM Press.
- [4] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of SenSys 2003*, New York, NY, USA, 2003, pp. 138–149, ACM Press.
- [5] J. Aweya, D. Y. Montuno, M. Ouellette, and K. Felske, "Clock synchronization using a linear process model," *International Journal of Network Management*, vol. 16, no. 1, pp. 3–28, 2006.
- [6] A. Pásztor and D. Veitch, "PC based precision timing without GPS," in *Proceedings of SIGMETRICS 2002*, New York, NY, USA, 2002, pp. 1–10, ACM Press.
- [7] K. Iwanicki, M. van Steen, and S. Voulgaris, "Gossip-based clock synchronization for large decentralized systems," in *Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems and Services (SelfMan 2006)*, Dublin, Ireland, June 2006, pp. 28–42.
- [8] D. Veitch, S. Babu, and A. Pásztor, "Robust synchronization of software clocks across the Internet," in *Proceedings of IMC 2004*, New York, NY, USA, 2004, pp. 219–232, ACM Press.
- [9] D. Tulone, "A resource-efficient time estimation for wireless sensor networks," in *Proceedings of DIALM-POMC 2004*, New York, NY, USA, 2004, pp. 52–59, ACM Press.