Accurately Initializing Real Time Clocks to Provide Synchronized Time in Sensor Networks

Hessam Mohammadmoradi
University of Houston
hmoradi@cs.uh.edu

Omprakash Gnawali
University of Houston
gnawali@cs.uh.edu

Alex Szalay
Johns Hopkins University
szalay@jhu.edu

Abstract—**Time synchronization is an essential service in many sensor network applications. Several time synchronization protocols have been proposed to synchronize the clocks on sensor nodes. All the synchronization protocols bring energy and complexity overhead to the network. There are many WSN applications which timestamp their samples with second level granularity. We claim real-time clocks can satisfy timing requirements of such applications. In this work, real time clocks are utilized to build a time-synchronized sensor network. We propose an accurate, robust, scalable and energy efficient approach to synchronize sensor network by accurately initializing RTCs on sensor nodes. Our experiments show that accurate initialization of real time clocks allows us to build a time-synchronized sensor network without any energy or complexity overhead of running sophisticated online time synchronization protocol.**

I. Introduction

Sensor networks have a diverse range of applications. Environmental monitoring [1], home climate monitoring [2], structural health monitoring [3], and sniper localization [4] are examples of sensor network applications that have been explored in the past.

To utilize the sensor readings in the context of the specific application, oftentimes, we need to know the time at which the reading was obtained. In some applications, the sensor data needs to be analyzed across time to understand the temporal trends. In some applications, the sensor data needs to be analyzed across space. For example, in an ecology monitoring application, we may need to not only know the sensor readings from different places but also the ability to compare readings from different times. These types of temporal and spatial analyses require that the sensor readings are timestamped with a globally consistent and accurate time.

Many sensor network applications (e.g., soil sensing in agriculture; see table II for other examples) do not require microsecond level time synchronization accuracy. In such applications, high accuracy and precision time synchronization protocols (e.g., RBS [5], FTSP [6], Glossy [7], PulseSync [8]) may be an overkill. Our proposal is to accurately and precisely program the Real Time Clocks(RTC) and use them to provide timing to these applications instead of online time synchronization protocols.

Utilizing RTCs to provide synchronized time to wireless sensor network requires solving one main technical challenge. The RTCs operate at a precision of one second. However, we need to scalably synchronize the RTCs on the nodes as accurately as possible, ideally with any-to-any pairwise error of less than one ms. In this paper, we investigate several approaches to accurately initialize the RTCs on the motes. Our results indicate that RTCs can be initialized with less than 40 microseconds (200 microseconds in the worst case) offset across the node pairs.

Our solution consists of a mechanism to accurately initialize the network of real-time clocks that provide wall clock time to the motes. Each mote in the network has an RTC with its own independent power source as part of the node. During the initialization of the mote (e.g., when it is programmed), we synchronize the RTC to a reference time using a set of wired or wireless synchronization techniques. The key technical insight in our approach is using fine-grained measurements and delays to achieve a sub-millisecond precision despite starting with the clock with a one-second precision.

In this work, we make these contributions: (1) Design of wired and wireless mechanisms to initialize the RTCs in a sensor network and achieve pairwise errors in the order of 40 μs. (2) Comparison of the RTC initialization approaches using extensive experiments on multiple hardware platforms.

II. Related Work

The most challenging problem with RTCs is drift over time. Recently researchers proposed a compact, lower-power and lower-cost circuit which accurately (0.01 ppm) compensates local clock's drift over time [10]. In addition software approaches [11],[12] have been proposed which model clock drifts and also estimate clock offset from reference time. While these efforts help maintain timing accuracy over time, our work focuses on initializing the clocks accurately in the first place.

Researchers have also explored the idea of using real time clocks in time synchronization. The HARMONIA [13] system relies on an accurate real time clock for timekeeping at a coarse granularity (one second) while the high-frequency micro-controller clock is used to synchronize RTCs. In our work, we focus on accurately initializing the RTCs.

RTCs are quite popular in embedded systems. We surveyed the current status of RTC initialization techniques in the embedded systems code available publicly. We surveyed 14 different open source software and libraries which initialize RTCs and summarized the results in table I. During the survey of those RTC initialization code, we found that none

TABLE I
SURVEY OF RTC INITIALIZATION CODE. SUMMERY OF [9]

# of Projects	Device	Source of Time	Gap between fetch and initialize	Scalability	Internal/External clk
2	Arduino	From user on serial port	Write immediately	One at a time	External
3	Arduino	compile time	Write immediately	One at a time	External
2	Arduino+Ethernet	NTP	Write immediately	One at a time	External
3	Desktop	System Time	Write immediately	One at a time	Internal
2	Desktop	System Time	Write immediately	One at a time	External
1	Desktop	NTP	Calculate offset time	One at a time	External
1	Raspberry Pi	System Time	Write immediately	One at a time	External

TABLE II
SAMPLE WSN APPLICATIONS UTILIZING LOW GRANULARITY TIMESTAMPING.

Project Name	Granularity	Deployment Length
LifeUnderGround[14]	1 second	6 Months
VigilNet[15]	1 second	Few Months
iMonnit[16]	1 Minute	1 Year
Engagement Analysis[17]	1 Minute	3 Months
Automated Irrigation [18]	1 Second	18 Months

of the studied approaches consider sub second initialization error for initializing RTCs. They fetch the time and write to RTC registers. Another takeaway from the survey is all the mentioned approaches program one RTC at a time. In scenarios with hundreds of nodes, we need to extend those techniques.

Despite the popularity of RTCs in WSNs, a detailed analysis of RTC's initialization process and their applicability in building scalable time synchronized sensor networks have not been investigated before. In our work, we accurately initialize RTCs before deployment and thereby, to some extent, making it unnecessary for the base station to send periodic time updates to the network. The suggested approach, once deployed, does not require network communication and hence can be robust to harsh network environment or reboots and other disruptions.

III. SYSTEM DESIGN

Our design of RTC initialization system consists of (1) the hardware architecture that integrates motes with RTCs with (semi)independent power supply to provide synchronized timing service for the sensor network and (2) a mechanism to scalably and accurately initializate a set of RTCs on the nodes.

A. Timesync Architecture that utilizes RTCs

In this architecture, each mote is equipped with an RTC with an independent or semi-independent power supply. Off-the-shelf RTC packages such as the ChronoDot [19] provide these features. It is also possible to integrate energy storage and RTC chip on to a mote design with minimal effort. Thus, the hardware requirement imposed by this architecture is modest and already fulfilled by many common sensor platforms.

The second component of the architecture is the mechanism that initializes the RTCs. The initialization system consists of a device that keeps the reference time, a bus that connects the reference time device with the nodes, and the commands to program the RTC with the current time. The mechanism is described in detail in the next section.

One concern with this approach is robustness: if the sensor node reboots, it may lose track of time because there is no time synchronization protocol during run time. Two main safeguards exist to address this concern: (1) availability of (semi)independent power supply allows the external RTC to keep time accurately despite sensor node's reboots, crashes, and power outages and (2) leveraging extremely robust, simple, and accurate off-the-shelf RTCs. Another concern is drift over time. As stated in related work section, currently there are low drift off-the-shelf clocks (0.01 ppm) and also simple and effective drift estimation algorithms to reduce drift impact. In addition, our focus is designing timing service for applications that require 1 second or coarser timestamping. Based on our results, RTCs can be used to build reliable time source for such applications.

Finally, the mote uses a simple protocol to query the RTC to obtain the current time. Because the RTC directly provides calendar time, the mote does not maintain mapping of local to reference time. The query is made over I2C or other bus that connects the MCU to the RTC on the device.

B. RTC Synchronization Through Accurate Initialization

The key idea in our approach is to convert the problem of synchronizing a network of RTCs into a problem of initializing the RTCs with accurate time and providing hardware resources and simple design to maximize the probability of uninterrupted timekeeping for rest of the deployment.

Traditionally, the RTCs have been programmed either by the end users or at factories. These initialization rarely emphasize high precision. The RTCs provide time at a granularity of one second. Thus, a user may look at the watch and program the RTCs. One may also write a program that obtains the current UNIX time and program that time to the RTC. We investigated different initialization methods and measured their accuracy and sources of error.

Removing sub-second jitter while programming: Fetching the current time either by GPS or NTP and programming the RTC with that time as quickly as possible could take indeterminate time. As a result, the RTC may be programmed with past time with variable offset. A solution is to introduce appropriate wait between time fetch and RTC initialization (using `wait()` function) or estimate the time when RTC initialization is expected to execute and programming the RTC with that *future* time instead of the current time. This requires modeling the delays in the system, including jitters generated by the `wait()` function.

Programming a set of RTCs at a time: The node with the reference time should program as many RTCs as possible at a time to minimize the initialization time and to minimize the possible difference in time across the nodes. For a small number of nodes, it is possible to build a wired bus to initialize a set of RTCs at a time. For larger number of nodes, we use one-hop wireless communication to provide the reference time to the nodes and initialize all the nodes nearly simultaneously.

IV. Implementation

A. Real Time Clocks

Almost all embedded systems, including sensor networks have internal RTC which can be utilized for time synchronization purposes; but, internal RTCs are subject to high drift and node reboots. We need to equip nodes with external RTCs which have lower drift and more reliable power source. We use ChronoDot [19] as an example of external RTCs in our implementation experiments.

ChronoDot is an accurate real time clock module, based on the DS3231 temperature compensated RTC (TCXO). In normal usage scenario its battery estimated to work for 8 years.

Other RTC solutions may drift minutes per month, especially in extreme temperature ranges but the ChronoDot will drift less than a minute per year. This makes the ChronoDot well suited for time critical applications that cannot be regularly synchronized to an external clock.

B. Wired Initialization of RTCs

We use Arduino, Raspberry Pi, and Laptop to initialize the Chronodot over the wire.

1) Online Initialization via Arduino: Arduino is an open-source computing platform based on a simple micro-controller board, and a development environment for writing software for the board. The Arduino device can program multiple ChronoDots at a time through the I2C bus.

2) Online Initialization via Raspberyy Pi: Raspberry Pi also supports I2C, to communicate with ChronoDots. Raspberry Pi also has Ethernet port, thus it connects to the Internet and obtains NTP (Network Time Protocol) [20] time. Thus, Raspberry Pi is a suitable choice for online initialization of ChronoDots.

Minimize initializing jitters with RealTime OS: In order to provide predictable delays during time estimation and programming the ChronoDot, we installed a real time operating system (Patched Raspbian) on Raspberry Pi. In real time OS, all the kernel code can be preempted. We changed operating system scheduling policy from standard time sharing (SCHEDOTHER) to first in first out one. With unpatched Raspbian operating system, ChronoDot initialization faced hundreds micro-seconds of jitter (200 us) but real time patch reduces the jitter to 40 microseconds.

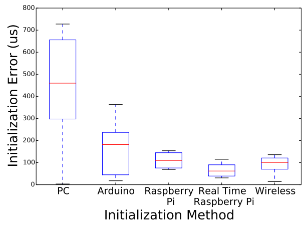


Fig. 1. Online initialization error with different methods

3) Online Initialization via PC: In a lab environment, PC or similar devices are normally used to program the motes. In this method, we use the PC to initialize the ChronoDot attached to the mote during mote programming. Although the seamless programming and initialization the ChronoDot in one shot is appealing, this method could introduce some jitters due to the unpredictability of a general-purpose OS on the laptop and due to the presence of an intermediate component (mote) between the initiator, the PC, and the ChronoDot.

C. Wireless Initialization of RTCs

1) Wireless Online Initialization of RTCs: Wired initialization does not scale to a large number of nodes. Wireless initialization can initialize a large number of RTCs at the same time. The key idea here is to attach RTCs to sensor nodes and put nodes in transmission range of base station. The base station sends the initialization command and reference time via single message. Once each node receives the reference time, it initializes its RTC. Note that this time configuration protocol is expected to run just once during factory initialization or staging lab before the deployment thereby avoiding any communication overhead during deployment-time.

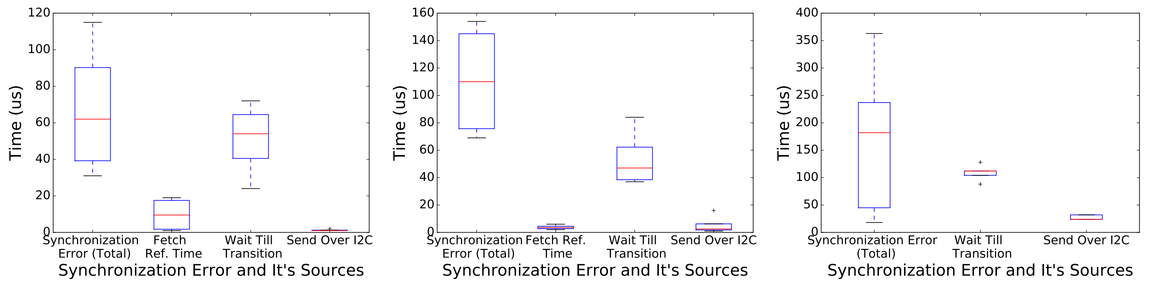
2) Wireless Offline Initialization of RTCs: In this method, the motes first obtain the RTC time through I2C interface. They then report this time to the base station. The base station uses fine-grained instrumentation to record accurate reception time at sub-millisecond precision. The base station then can use multiple timestamps to create a regression model to obtain sub-millisecond time on the mote. If the jitters on the I2C on the mote RTC interface is small and the recording of reception timestamps is accurate when the base station receives the RTC time, then the overall error of this technique can be low. We perform experiments to understand the impact of these contributors to the overall accuracy of the system.

V. Evaluation

1) Measurement approach: In this section, we summarize the performance results from our experiments.

A. Initialization Accuracy

One of our goals is to accurately initialize the RTCs with the various approaches to understand the most effective method to initialize RTCs. We now describe our experiment setup to study the initialization accuracy.



(a) RaspberryPi (Real Time Raspbian 4.1) (b) Raspberry Pi (Raspbian 4.1) (c) Arduino

Fig. 2. Jitter Source Analysis During Online Initialization

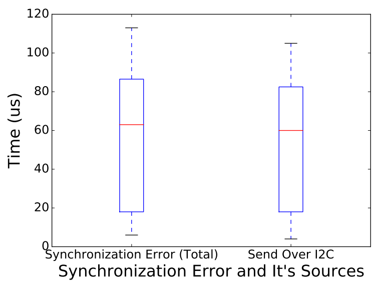


Fig. 3. Error with jitter-free fetching of reference time.

We programmed all the Chornodots and set their alarm register to a specific time. We connected their alarm pin (SQW) to logic analyzer and waited for the alarm to raise the output pins. The time between when the pin from one ChronoDot is raised and when the pin from the second ChronoDot is raised is the synchronization error for that pair of ChronoDots. Our goal is to measure time difference between all those ChronoDots by measuring time difference between the rising edge of each signal. We sampled data with 1 M (10^6) samples per second with a multiple-channel logic analyzer and measured time difference between rising edge of output signals of Chronodots.

1) Measured Error: Figure 1 shows the histogram of measured errors for the four methods of online initialization. Initialization with PC has largest initialization error and real time OS on Raspberry Pi has most accurate results.

2) Source of Error: Among all the techniques, the most accurate approach is using Raspberry Pi booted with real time OS. We measured a delay of about 40 microseconds in initialization process. We can divide initialization process to: Step 1: Fetching reference time (NTP time) Step 2: Delay till transition from second t to second t+1 Step 3: Send reference time to ChronoDot via I2C

Figure 2 shows delay and jitter values for the three steps. To measure synchronization error, before starting of each step, we raised a signal and immediately after finishing lowered it back. Using logic analyzer, we measured the total time it takes for each step to finish and from that we calculated jitter values for each step. Large jitters will result in large errors in initialization.

The main source of jitter is the `wait()` function. Other two steps have less then 10 microseconds jitter. The problem with `wait()` function is, we use OS-provided sleep functionality which is based on CPU's internal clocks and its accuracy is depends to hardware architecture.

We also measured synchronization errors using general-purpose operating system instead of a real time one. As seen in figures (2a) and (2b), using real time operating system and application improves accuracy by almost 200 %. We also did the same analysis for Arduino and results are shown on figure 2c.

B. Error with jitter-free fetching of reference time

The major source of error was fetch time jitter. Figure 3 shows sources of error in the case of jitter-free fetch time. Based on figure 3,the main source of initialization error is the I2C jitter. The approach also illustrate the best case accuracy: initiating RTC programming using GPIO prevents some errors and we can achieve almost 10 microseconds synchronization error among nodes.

C. Offline Wireless Initialization Performance

In this experiment a network of 10 bacon motes, each equipped with unsynchronized ChronoDots, is deployed in transmission range of a base station. Each mote, after a random back off time, sends its local RTC time value to base station. Using GPIO pins, we measured average time it takes for motes to capture synchronized time from attached ChronoDot via I2C interface and also time for transmitting message to base station. The base station, then uses the technique described as wireless offline initialization (Section IV-C2) to keep track of mapping between RTC time on the individual motes and the wall clock time at sub-ms precision. Figure 4 shows experiment's results. We find that offline initialization incurs less than 20 μs jitter. Thus, offline approach is more deterministic compared to the online initialization techniques.

D. Scalability

The wired initialization approach has limited scalability but the wireless approach can handle large sets of nodes. We equipped 10 Bacon motes with ChronoDots. We configured a base station with access to the reference time. The base

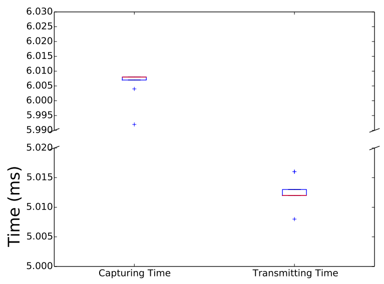


Fig. 4. Offline Initialization Performance

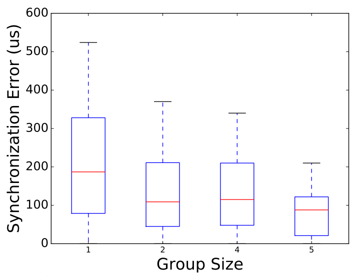


Fig. 5. Multiple Round Wireless Initialization

station periodically broadcasts synchronization messages to the network. Each receiving mote initializes its own RTC and sends an acknowledgment to the base station. The base station continues to broadcast the initialization messages to the network till it receives acknowledgment from all the nodes. We emulate a real world situation in which not all the motes receive the initialization messages in one round. We conducted four sets of experiment and in each experiment we changed the number of initialization rounds. On each round, a specific number of nodes receive synchronization message and we call them a group. As an example, whenever we have two rounds of initialization, there are two groups, each with five nodes.

Figure 5 illustrates pairwise synchronization errors on different experiments. We learn from figure 5 that the number of synchronization rounds have direct impact on synchronization error. In other words, more rounds of synchronization increases time difference between sensors but even in case of 10 rounds of synchronizations, error will not exceed a few hundreds of microseconds.

The alternative approach, offline initialization, has even better scaling characteristics. All the sensor motes can send their local time to the base station and using just two records per mote, base station can build a linear model for mapping the mote RTC time and the reference time. Thus scaling of this approach is limited only by the state on the base station. Most often, the base station is assumed to have large memory and computational resources.

VI. Conclusions

Differnet WSN applications have different timstamping accuracy requirement. We focus on applications that can live with some errors and coarse timestamping but desire extreme simplicity and reliability over deployments. In this paper, we investigated applicability of using reliable RTCs to build time synchronized sensor networks without using costly and complicated time synchronization protocols. We designed wired and wireless approaches to initialize RTCs and evaluated their performance via extensive testbed experiments and the results show high levels of accuracy and robustness.

VII. Acknowledgement

This work was supported in part by the National Science Foundation under grant no. IIS-1111507. We also thank Doug Carlson, Nire Rattner and Andreas Terzis for their support in this work.

References

[1] L. M. Oliveira and J. J. Rodrigues, "Wireless sensor networks: a survey on environmental monitoring," *Journal of communications*, vol. 6, no. 2, pp. 143–151, 2011.

[2] M. Kohli and R. Tiwari, "Wireless sensor networks: Applications and impact," *IUP Journal of Information Technology*, vol. 10, no. 1, p. 56, 2014.

[3] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Wireless sensor networks for structural health monitoring," ser. SenSys '06. New York, NY, USA: ACM, 2006, pp. 427–428.

[4] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 1–12.

[5] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 147–163, 2002.

[6] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 39–49.

[7] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *IPSN'11*.

[8] C. Lenzen, P. Sommer, and R. Wattenhofer, "Pulsesync: An efficient and scalable clock synchronization protocol," *TON*, vol. 23, no. 3, 2015.

[9] H. Mohammadmoradi and O. Gnawali, "State of the Art RTC Initialization Approaches," University of Houston, Department of Computer Science, Tech. Rep., 10 2016.

[10] M. Buevich, N. Rajagopal, and A. Rowe, "Hardware assisted clock synchronization for real-time sensor networks," in *RTSS'13*.

[11] H. Kim, X. Ma, and B. Hamilton, "Tracking low-precision clocks with time-varying drifts using kalman filtering," *Networking*, vol. 20, no. 1, 2012.

[12] M. Leng and Y.-C. Wu, "Low-complexity maximum-likelihood estimator for clock synchronization of wireless sensor nodes under exponential delays," *Signal Processing*, vol. 59, no. 10, 2011.

[13] J. Koo, R. K. Panta, S. Bagchi, and L. Montestruque, "A tale of two synchronizing clocks," in *ENSS'09*. ACM, 2009, pp. 239–252.

[14] "life under ground," http://lifeunderyourfeet.org/en/, accessed: 2016-04-13.

[15] J. A. Stankovic, A. D. Wood, and T. He, "Realistic applications for wireless sensor networks," in *Theoretical Aspects of Distributed Computing in Sensor Networks*. Springer, 2011, pp. 835–863.

[16] D. Basu, G. Moretti, G. S. Gupta, and S. Marsland, "Wireless sensor network based smart home: Sensor selection, deployment and monitoring," in *SAS'13*.

[17] H. Mohammadmoradi, O. Gnawali, D. Moss, R. Boelzle, and G. Wang, "The impact of user engagement in the effectiveness of energy saving programs," in *Proceedings of the IPSN'16*. IEEE, 2016.

[18] J. Gutirrez, J. F. Villa-Medina, A. Nieto-Garibay, and M. . Porta-Gndara, "Automated irrigation system using a wireless sensor network and gprs module," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 1, pp. 166–176, Jan 2014.

[19] "Chronodot key features," http://goo.gl/UdUIX5, accessed: 2016-04-13.

[20] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," Tech. Rep., 2010.