Simulation and Performance Analysis of the IEEE1588 PTP with Kalman Filtering in Multi-hop Wireless Sensor Networks

Article *in* Journal of Networks · December 2014

DOI: 10.4304/jnw.9.12.3445-3453

CITATIONS
2

READS
100

7 authors, including:

Xuewu Dai
Northumbria University
52 PUBLICATIONS **348** CITATIONS
SEE PROFILE

Wuxiong Zhang
Shanghai Institute of Microsystem And Information Technology
8 PUBLICATIONS **38** CITATIONS
SEE PROFILE

Yang Yang
General Electric
144 PUBLICATIONS **2,595** CITATIONS
SEE PROFILE

TS2: A Realistic IEEE1588 Time Synchronization Simulator for Mobile Wireless Sensor Networks

Yiwen Huang[1], Taihua Li[2], Xuewu Dai[3,1], Haowen Wang[4], Yang Yang[4]

Abstract

This paper presents the development of a simulator, named TS2 (Time Synchronization Simulator), for realistically simulating and studying the IEEE 1588 Precise Time Protocol (PTP)'s performance in IEEE 802.15.4 (TI CC2420 chip)-based mobile Wireless Sensor Networks (WSNs). The PTP has advantages of achieving high time precision at low costs. It was designed for wired Ethernet with requirements of symmetric communication paths and accurate time stamping, which, however, is difficult for the low cost WSN to meet as WSNs suffer more from overwhelming transmission delay jitters. An analytic solution to the PTP's performance in WSNs is not possible and it is of importance to evaluate the performance by realistic simulation. Based on the open source OMNeT++ simulation engine, a realistic simulator is developed to simulate the PTP in IEEE 802.15.4 networks. The main contributions and benefits of the developed simulator are: (1) Reality and fidelity. The WSN node's various layers (including radio channels and TI CC2420 chip), drifting oscillator clocks and PTP protocols are simulated realistically. The drifting clock is simulated at an adjustable and higher resolution. (2) Support to both hardware and software time-stamping, and the time-stamping uncertainties by using a separated modular time-stamping module. It also has other features (such as extendibility and code-reusability, mobile WSN nodes, scalability for multi-node, multi-hop simulation). Finally, to demonstrate the simulator's application to evaluating a PTP-based clock correction algorithm, a direct servo clock adjustment algorithm (i.e. a P controller) for a TI CC2420-based WSN is simulated and its performance is analyzed.

Keywords

Time Synchronization, IEEE 1588, OMNeT++ Simulation, Wireless Sensor Networks, IEEE 802.15.4

1 Introduction

In many distributed real-time systems, such as trains, cars, industrial automation and manufacturing, it is crucial to maintain precision time in distributed nodes for accurate time stamping of events, better coordination of data transmission and fine-tuned duty cycles to reduce power consumption. Time synchronization is the process that adjusts each distributed node clock so that the nodes' clock drifts are limited to a sufficiently small range. Since nowadays most communication networks (e.g., Internet, Ethernet, WiFi and Zigbee, etc.) are packet exchange networks, the trend in time synchronization is to using time-stamped packet exchange techniques to estimate the clock drifts so that the accuracy of the clock can be improved.

1.1 Time Synchronization in WSNs

In recent years, various packet-exchange time synchronization protocols have been proposed. A typical example is the Network Time Protocol (NTP) [1] which has been widely used in the Internet. However, the accuracy of NTP is within tens of milliseconds over the public Internet and few milliseconds in wired LANs at ideal conditions. As a result, it cannot be used in applications requiring high time accuracy, such as Time Division Multiple Access (TDMA) scheduling, Time-of-Flight (TOF) localization and industrial field networks [2], etc. In order to achieve better time accuracy in industrial Ethernet, the IEEE 1588 Precise Time Protocol (PTP) [3] is proposed with a target accuracy level of microsecond，or even sub-microsecond, which is much higher than the time synchronization accuracy based on NTP.

For WSNs, many time synchronization protocols have been proposed in recent years due to the dramatic increase of the application of WSNs. Some typical time synchronization protocols for WSNs are Reference Broadcast Synchronization (RBS) [4], Flooding Time

1 School of Electronic and Information Engineering, Southwest University, Chongqing, China, 400715
2 School of Physical Science and Technology, Southwest University, Chongqing, China, 400715
3 Faculty of Engineering and Environment, Northumbria University, Newcastle upon Tyne, UK, NE1 8ST
4 Shanghai Research Center for Wireless Communications (WiCO), Shanghai, China, 200335

Corresponding Author: Dr. T. Li, E**mail**: catalyst@swu.edu.cn

synchronization Protocol (FTSP) [5], Time Synchronization Protocol for Sensor Networks (TPSN) [6], Time-diffusion synchronization protocol (TDSP) [7] and local synchronization protocol [8], etc.

On the other hand, due to the low cost requirement and limited resources, WSN node's clocks are generally implemented through cheap crystal oscillator with some kind of interrupt mechanism. The frequency of a cheap crystal oscillator is greatly affected by manufacturing errors and temperature variation. And the low cost CPU with limited RAM/ROM memory also result in a longer delay in processing interrupts with larger variances [9]. For example, the Mica2Dot platform has only 128K bytes of program memory, 4K bytes of data memory and a maximum radio bandwidth of 38.4 Kbps. Moreover, since wireless nodes are powered by battery and have limited energy, less computing resources and low communication capacity, a time synchronization protocol on the low cost WSN platforms must therefore have low computation, memory and bandwidth overhead. The main challenges in WSN time synchronization is how to achieve high-precision time synchronization while requiring less overhead and consuming, lower communication bandwidth and CPU sources.

The resource-constrained environment of various WSNs platforms motivated us to study a light-weight design that could be implemented in a simple manner. Compare to most existing time synchronization protocols, one of promising features of the PTP is its less demanding on communication bandwidth and computation costs, while keeping the similar level of time accuracy [10]. With small communication bandwidth, the PTP occupies relatively fewer network resources. The PTP has simple operation, small computational and memory requirement, lower hardware requirements, and is easy to maintain. As a result, a well optimization on energy efficiency and accuracy can be achieved by using the PTP in WSNs.

However, the IEEE 1588 is originally proposed for wired Ethernet and it has three assumptions [3]. The degree to which these assumptions hold true determines the accuracy of the clock synchronization. Namely, these three assumptions are:

(1) The exchange of time-stamped packets happens over a short period of time so that the slave's clock offset and skew can be considered constant over that period. Equivalently, this assumption is an assumption on the oscillator stability and accuracy, which requires high quality oscillators.

(2) Symmetric communication paths. It requires the transit time of a message going from the master to a slave is equal to the transit time of a message going from the slave to the master.

(3) Accurate time stamping. It is assumed that both the master and slave can accurately measure the time they send or receive a message.

In wired Ethernet, it is possible to meet these requirements, as high-quality hardware and sufficient bandwidth are achievable in a wired network. For the low-cost WSNs nodes, it is a big challenge to satisfy these assumptions. An example is the Mica2Dots that were reported to have a measured relative skew of about 3000PPM with significant fluctuations [11]. It is much higher than the PTP's requirement of constant offset and skew.

Due to its advantages and challenges motioned above, the PTP is more suitable for communication bandwidth and energy constrained WSNs, in particularly, comparing to the traditional time synchronization protocols (e.g., RBS, FTSP, etc.). However, PTP's synchronization precision is significantly affected by transmission delay jitters and timestamp accuracy. Due to the complexity of wireless communication, such as the sharing media, severer channel fading, hidden nodes, longer bakeoffs and more packet retransmission caused by collision and packet loss, an analytic solution to PTP's performance in WSNs is very difficult and it is of importance to evaluate PTP's performance by realistic simulation.

1.2 Discrete Event Simulator

Discrete Event Simulator (DES) is the state-of-the-art tools for network and protocol simulation. In this approach, the behavior of a complex system (e.g. a network and a protocol, etc.) is represented as an ordered sequence of well-defined events in time (such as packet transmission and receiving, interrupts and so on). This definition feature makes the DES a good choice for simulating time synchronization protocols in various networks [12]. Some examples of famous DESs in network simulation are the commercial OPNET [13] and open source NS2/3 [14] and OMNeT++ [15].

Although there have been some DESs for simulating time synchronization, such as Simsync [16], x-simulator [17], Castalia with clocks [18] and the simulator presented in [19]. Both Simsync and x-simulator simulated the time synchronization only with detailed modeling of communication networks (i.e., MAC, IP layers). Liu etc. [19] simulate the PTP over IP networks on the basis of INET framework with an abstract model of radio channel. However, the industrial WSNs applications usually work in a complicated electromagnetic environment and require a detailed radio channel model to have a realistic simulation of the MAC delays. WSNs have huge numbers of nodes, and there are interferences and conflicts among multi-point communication. It has been shown that the simulation of physical layer has significant impact on the accuracy of

WSN simulation [20]. The simulator MiXiM [21] does have a good radio channel model to achieve highly accurate simulation of physical and MAC layers. A recent realistic simulator of industrial WSNs and sensor networks is WIDAGATE, which is developed to simulate the wireless data collection network for aero engine testing [22]. There has never been a simulator combining the PTP simulation with realistic radio, MAC layer and oscillator clock simulation.

In order to study the effects of timestamp uncertainties, communication and interrupt process delays and jitters on the PTP synchronization accuracy, realistic simulations of the drifting crystal oscillator clocks, the wireless channel, RF hardware and communication stacks are required. The simulator should be able to simulate these events from transmission power, channel attenuation, signal-to-noise-interference-ratio (SNIR), Bit Error Rate (BER), MAC scheduling, CPU process latency to time jitter and delays. As an OMNeT++-based open source simulator, MiXiM is one of few simulators that simulate radio channel and RF signal at high resolution [21]. The *nic* module in MiXiM consists of an MAC layer module (*mac*) and a physical layer module (*phy*). The *phy* mimics the physical layer by two models: the *Analogue* Model for radio channel and the *Decider* for the signal processing in the RF frontend of wireless nodes. For a more detailed description of the MiXiM simulation concepts and its application to wireless sensor network simulation see [21].

Time synchronization process based on the time packet exchange involves two aspects: the drifting local clock and the time packet exchange protocol. In order to study time synchronization, the biased local clock needs to be modeled, and the time packet exchange protocol must be implemented. A good overview of clock modelling and simulation is provided in [23]. A typical oscillator clock consists of a crystal oscillator and a chain of adders (counters) triggered by the oscillator and the clock drifting is usually modelled with simple white noise or Brownian motion. An Allan variance-based clock model is developed for DES in [24], in which the adder's behavior, error sources, event scheduling and its interaction with time synchronization are analyzed.

Since the PTP protocol indeed is a series of events of packet exchange, the simulation of PTP is straightforward in DESs. The challenges are the interaction between clock and synchronization, i.e. servo clock and clock correction. The concept of a time-discrete Proportional-Integral (PI) control was proposed in [25] and a Proportional-Integral-Derivative (PID) controller for an adder-based clock was proposed in [24]. Paper [26] shows that proper parameterization of the PI controller is the key for keeping the servo clock precision. A control servo clock based on the simple skew model was simulated in [17] and, recently, a Kalman filter was proposed for clock servo design [27].

In this paper, we present the development of a realistic simulator for evaluating PTP's performance in the scenario of mobile wireless sensor networks. Since the packet-exchange-based time synchronization involves two aspects: the inaccurate local clock and the time packet exchange protocol, our main attention is paid to: (1) Modelling and simulating WSN node's inaccurate oscillator clock and the PTP protocol, including interrupt processing, time-stamping and clock correction. (2) Interfacing our simulator with the OMNeT++/MiXiM framework to have a realistic time synchronization simulation. The proposed simulator is developed on the open source OMNeT++ simulation platform [15] and its MiXiM package. The OMNeT++ is chosen as the simulation platform, because it is an open source network simulator with a generic, flexible and modular architecture and many simulation packages available for various networks and protocols. Furthermore, the MiXiM allows us to develop an advanced simulator to achieve realistic delay and jitter simulation, which are the key factors affecting the performance of the PTP protocol, we make use of MiXiM to simulate the radio channel (physical layer) and MAC layer in WSNs. It also supports the simulation of the movement of WSN nodes with various motion modes.

The main contributions and significance of the developed simulator are two-fold:

(1) An improved simulation model of WSN node's drifting oscillator clocks and timestamp uncertainties. It enables the developed clock model to be updated faster so that the model is closer to the real oscillator clock. Most existing time synchronization simulators assume the oscillator clocks update interval τ_0 (i.e. resolution) is the same as the time synchronization interval ΔT, which puts to a limit on the resolution of the clock model. Our simulator decouples τ_0 from ΔT and enables a higher situation resolution of the oscillator clocks.

(2) A realistic, modular, extendable, and reusable time synchronization simulation platform. The proposed simulator simulates the WSN node's various layers (including radio channels), delays, jitters, oscillator clocks, and node movements with good fidelity and reality. In particular, the proposed simulator achieves realistic delay and jitter simulation of IEEE 802.15.4 (by TI CC2420), which are the key factors affecting the performance of the PTP protocol. The proposed simulator can be easily extended for various networks, for example, the mobile networks such as Vehicle-to-Vehicle, UAV (Unmanned Aerial Vehicle) networks. The scalability for multi-node, multi-hop simulation is another benefit of our simulator. To demonstrate the application of the proposed simulator, the simulation results of the PTP with an attenuated clock correction algorithm in an IEEE 802.15.4-based mobile Vehicle-to-Vehicle network is presented, which evaluates the performance of our proposed simulator and its

significance.

The rest of this paper is organized as follows: Section 2 presents the requirement analysis, concept and architecture of the proposed time synchronization simulator. Section 3 studies the modeling of drifting oscillator clocks which are widely adopted by low-cost WSN nodes. Modelling of the IEEE 1588 Precision Time Protocol is presented in Section 4. Section 5 presents the OMNeT++-based implementation of the proposed simulator with emphasis on the structure of master/slave nodes, and Section 6 presents implementation of drifting clocks, time stamping, the PTP packet exchange, offset/skew estimation and clock correction. Some simulation results are examined in Section 7 followed by Section 8 finishing this paper with summary and future works.

2 Concept and Simulation Framework

The goal of the proposed simulator is a simulation framework capable of realistically modeling and simulating WSNs oscillator clocks at higher resolution and the IEEE 1588 PTP standard in various network topologies and mobility environments. This section presents the requirement analysis and the architecture of the proposed simulator.

master and slave nodes are equal. However, this assumption of symmetry may not be met in WSNs due to wireless channel sharing, possible collision, backoff and retransmission. It is particularly true if soft time stamping is adopted. Other factors, such as WSN node's CPU processing delays during interruption and task scheduling, also have adverse impacts on the time synchronization. The mobility of the nodes also presents a challenge to the simulation framework. In order to build a realistic simulation of time synchronization, a very detailed (in another words, accurate) model of node's clock, WSN node, wireless channel and networks and in-depth understanding of the inter-effects among related factors are required. Developing such a detailed model from scratch is possible, but it will be a tedious and error prone task. Thanks to many people's contributions to OMNeT++ open-source community, we may re-use some existing modules and source codes to development our simulator.

Various OMNeT-based simulators have been developed in the last decade for WSN simulation. Most of them are for the simulation of MAC and higher layers, and their treatment to the physical layer, hardware and wireless channel are superficial. Although there is some simulator (e.g., MiXiM) that provides the capability of simulating the WSN hardware and wireless channel with detailed model,

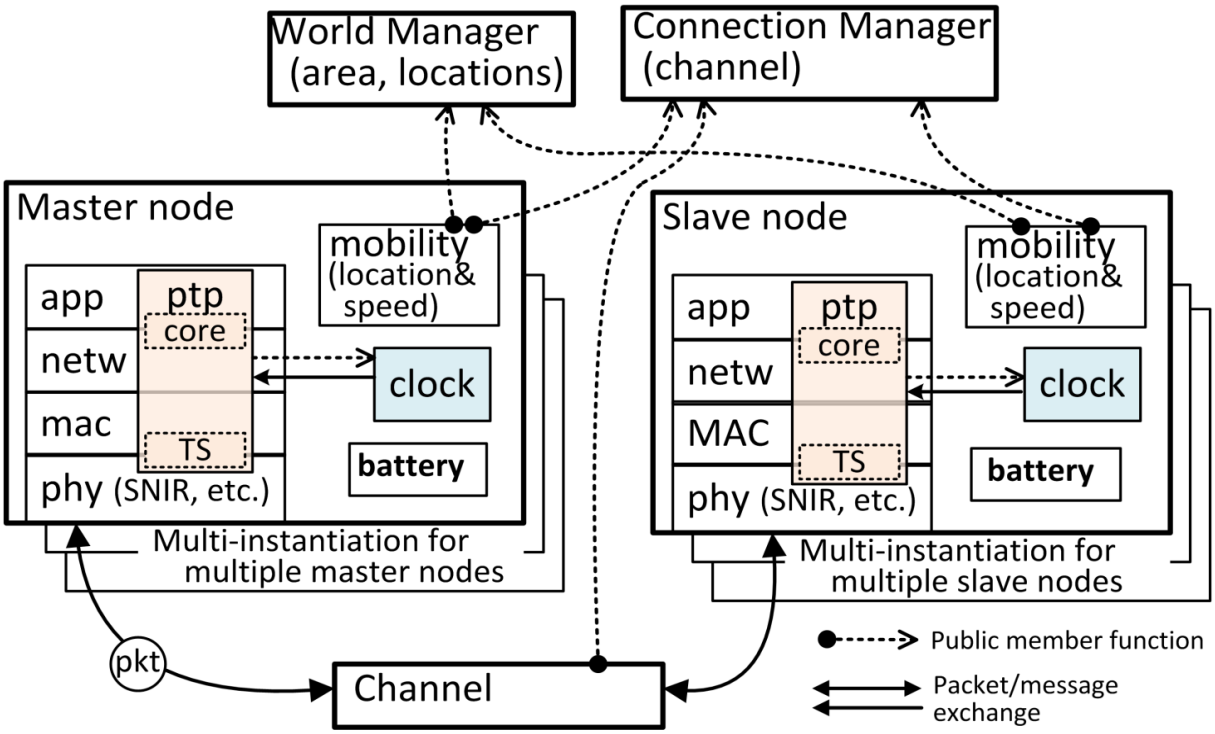

Figure 1 Simulation framework

In contrast to the wired channel in wired networks, the wireless channel in WSNs has complicated impacts on the time synchronization performance. In particular, the IEEE 1588 standard assumes the transmission delays between

there is a lack of detailed model of WSN clock. Since our focus is the detailed model of clocks and realistic simulation of time synchronization, the proposed simulation framework is based on MiXiM for the purpose

of reusing modules. Our efforts are on the development of detailed model of WSN clock and IEEE 1588 PTP protocol and the seamless integration our models into the MiXiM.

The general structure of the proposed simulation framework is shown in Figure 1, which is similar to a standard simulation network in MiXiM. The simulation framework has four components:

(1) Environment model that represents the geographical environment of the WSN network, such as the size of the WSN's deployment area, obstacles that block/attenuate wireless signals. In our simulator, the environment model is defined and controlled by the module *World Manager*.

(2) Mobility and connectivity management. When nodes move, the wireless channel varies and, consequently, the network topology changes. These changes are tracked by the module *Connection Manager*.

(3) Wireless channel that represents the features of the wireless channels and its impacts on packet transmission (e.g. attenuation, propagation delays, interferences and collisions).

(4) WSN nodes. There are two types of nodes, namely, master node and slave node.

As shown in Figure 1, a WSN node has the IP node structure with four layers which are mapped to and implemented by various C++ classes (referred to as modules). Namely, they are five protocol layer modules (the application layer *app*, the network layer *net*, the MAC layer *mac*, the physical layer *phy* and the PTP protocol module *ptp*) and some support modules (*clock*, *mobility*, *battery*, etc.). It worth noting that, as a realistic simulator, the *phy* module mimics the TI CC2420 IEEE 802.15.4 chip, including wireless channel sensing procedure, Signal-to-Noise-Interference-Ratio (SNIR) of each packet, etc., which give us a better and much accurate simulation of the packet transmission delays, jitters and the PTP protocol in IEEE 802.15.4 networks. Since the PTP indeed is a cross-layer protocol requiring two main procedures, time-stamping and offset estimation, in our simulator, the PTP protocol is implemented by two modules. The *TS* module is for time-stamping and the *core* module is for offset estimation. The *clock* module simulates the WSN node's oscillator clock and provides local time to other modules within the node. The details of the interface between the *ptp* and *clock* modules will be presented with more details section 5. The *mobility* module stores and updates node's location and speeds and reports to *World* and *Connection Manager*. The *battery* module models the battery's current supply and node's energy consumption to simulate the WSN node's life span. The *World Manager* and *Connection Manager* update node's locations and set up the wireless links among nodes and network topology accordingly.

Compared with most existing simulators, the proposed simulation framework provides the following features: (1) Realistic wireless channel and physical layer simulation. (2) Mobility simulation. The wireless nodes can move in the area at any specified speed and direction. (3) Realistic simulation of drifting oscillator clocks and time-related events (such as interrupts) at high resolution. (4) Implementation the functionality of time synchronization protocol by two modules, *TS* (time-stamping) module and *core* module. By this way, the hardware time-stamping or software time-stamping can be easily simulated by simply adjust the position of the *TS* module. These features make the proposed simulator more realistic for time synchronization of mobile wireless networks.

3 Modeling Temporal Dynamics of Oscillator Clocks

In a WSN node, the clock is a device to represent a certain time interval elapse by counting special event (e.g. the rising/falling edge of an electronic signal), and can be used to measure and indicate time. The special event is generated by a physical entity or electronic signal with stable frequency properties. The physical entity, also called as the clock source, which includes mechanical twisting, ceramic crystals or quartz crystal vibration, atomic oscillation, etc. For WSN nodes, the quartz crystal oscillator is the clock source.

A WSN node usually has multiple clocks which are constructed from hardware and software components. The clock systems of a WSN node consists of (1) an oscillator, controlled by a crystal, ticks at an nominal frequency; (2) a counter, counts the number of ticks produced by the oscillator. Physically, the outputs of a quartz crystal oscillator are only periodic sine or square waves, while, through the process the digital timing circuit (counter), these outputs can be converted to a count value plus one per clock cycle. By storing, comparing and converting to the count value, these tasks, such as the obtaining of time information, the comparison of the order of events, and the conversion of the time scale, etc., can be completed. Generally, the output of an oscillator is called the local physical clock, while the cumulative count value corresponding to the local physical clock is called the local software clock. In accordance with a specific format, the software clock can be stored to visually denote time. Due to it is difficult to directly correct the period and phase of an oscillator, the physical clock is generally not under software control. In general, the clock synchronization is actually achieved by adjusting the clock count value (i.e. software clock). In this paper, if no special notes, the local clocks are all denoted by the local software clocks. For example, on Linux, there are several clocks available. (1)

Jiffy counter, an uncorrected clock used internally to the OS; (2) System clock, a clock corrected by NTP.

3.1 Notations

Unless stated otherwise, the following definitions are applied when using these letter and terms throughput this paper:

- t: Time of an accurate clock. It works as a global reference time representing the actual time.
- ΔT: Synchronization interval (period). Time synchronization is performed in rounds repeatedly and ΔT denotes the interval between two adjacent synchronization rounds.
- n: It represents the n-th round of time synchronization.
- τ_0: Clock update interval representing the resolution of the WSN node's hardware clock. In order to distinguish from ΔT, τ_0 is referred to as physical clock update interval in this paper ($\tau_0 \ll \Delta T$).
- k: It represents the k-th physical clock update.
- t_k: The global reference time at the k-th physical clock update.
- t_1, t_2, t_3 and t_4: Four time stamps as specified by the IEEE1588 standard. t_1 and t_4 are time stamps by the master node and t_2 and t_3 are by the slave node. It is worth noting that, depending on which clock is used to generate the time stamps, t_1, t_2, t_3 and t_4 may not be the time stamps of the global reference time.
- θ: Clock offset (in second) representing the difference of time values between a drifting slave clock and the accurate master clock t.
- γ: Clock skew (in Part Per Million, PPM) representing the normalized frequency deviation from the nominal frequency.
- $\omega_\gamma(k)$: Skew noise representing the random changes of skew. $\omega_\gamma(k)$ is modeled as a Gaussian noise with variance σ_γ^2.
- ϕ: Phase noise of oscillator.
- $\omega_\theta(k)$: Differential phase noise that is defined as the difference between two consecutive phase noises $\omega_\theta(k) = \phi(k+1) - \phi(k)$. $\omega_\theta(k)$ is modeled as a Gaussian noise with variance σ_θ^2.
- ω_η : Time stamping noise representing the uncertainties and random delays caused by interrupts and CPU processing in acquiring time stamps. It is considered as a zero-mean Gaussian random process with variance σ_η^2.

3.2 Model of oscillator clocks

The normal operation of quartz crystal oscillators depends on the mechanical vibrations of a quartz crystal. The frequency of an oscillator is affected by two causes: the external environment and the inherent properties of the crystal itself. The environmentally induced effects are due to the changes in external parameters such as temperature, supply voltage, load change, phase noise and jitter, etc., while the inherent effects are the normal aging of crystal oscillators. The frequency of oscillation is sensitive to almost all environmental variables, in which the change of temperature has a larger effect on the frequency of an oscillator, and is probably the most difficult to handle.

3.2.1 Ideal accurate clocks. An oscillator has the nominal frequency of f_0 being the nominal number of cycles per second in Hertz (Hz). The output voltage of an ideal sinusoidal oscillator can be defined as:

$$V(t) = Vsin(2\pi f_0 t) \tag{1}$$

where t is the time variable represent the global reference time, V is a constant amplitude and the period of the sinusoidal wave is $\tau_0 = 1/f_0$.

Since time is represented by the phase of the sinusoidal wave generated by the oscillator (e.g., a phase of 2π represents a time period of τ_0), the sinusoidal wave is converted into pulse (e.g., one pulse for one cycle) and a counter (or a chain of counters) counts the pulses digitally and adds them up to get traditional time units of seconds, minutes, etc. This process can be modeled by comparing the instant phase $(2\pi f_0 t)$ against 2π.

$$k = \left\lfloor \frac{2\pi f_0 t}{2\pi} \right\rfloor = \lfloor f_0 t \rfloor = \lfloor t/\tau_0 \rfloor \tag{2}$$

where k is an integer indicating the how many cycles has occurred and the floor function operator $\lfloor x \rfloor$ represents the largest integer not greater than x. The floor operator can be understood as k is updated until the integer number of cycles has passed. The event of the counter reaching value k is referred to as k-th counting event.

Let time instant t_k denotes the global reference time at the k-th event, t_k can be seen as the endpoint of a sequence of consecutive intervals, starting from time t_0=0. Given an accurate clock with a constant oscillator frequency equal to the known nominal frequency f_0, the time interval between its two consecutive events is constant. Let τ_0 denote the constant interval for such an accurate clock, t_k is determined by

$$t_k = k \cdot \tau_0 \tag{3}$$

Although a clock's time is commonly indicated as $C(t)$, implying a continuous dependence on the time variable t, eq.(2) shows that only finite increments can actually be achieved. Therefore, we use $C(k)$ to represent the clock's time at the occurrence of k-th event. For such an ideal clock, the clock time is

$$C(k) = t_k = k \cdot \tau_0 \tag{4}$$

3.2.2 Inaccurate drifting clocks. However, in reality, the phase of an oscillator may suffer from random changes and the oscillator frequency may change due to parameter variation (e.g. temperature) and aging. In order to model the oscillator frequency changes, let $\alpha(t)$ denote oscillator frequency change at time t, the actual frequency is a time-varying $f(t)$

$$f(t) = f_0 + \alpha(t) \tag{5}$$

The phase noises of an oscillator are modeled by a random process $\phi(t)$ that represents all the instant phase deviations.

Therefore, the phenomenon of varying oscillator frequency and phase noises are modeled by replacing the constant nominal frequency f_0 with the time-varying frequency $f(t)$. Accordingly, the drifting clock are modelled by modifying the accurate oscillator model (1) into

$$V(t) = Vsin\left(2\pi(\, f_0 t + \int_o^t \alpha(\tau)d\tau\,) + \phi(t)\right) \tag{6}$$

and modifying the counting procedure as shown in (2) to

$$\begin{aligned} k &= \left\lfloor \frac{2\pi(f_0 t + \int_o^t \alpha(\tau)d\tau) + \phi(t)}{2\pi} \right\rfloor \\ &= \left\lfloor f_0 t + \int_0^t \alpha(\tau)\, d\tau + \frac{\phi(t)}{2\pi} \right\rfloor \end{aligned} \tag{7}$$

As a result, the time value of a drifting clock at the occurrence of *k*-th event is

$$\begin{aligned} C(k) &= \left(f_0 t_k + \int_0^{t_k} \alpha(\tau)\, d\tau + \frac{\phi(t_k)}{2\pi}\right) \cdot 1/f_0 \\ &= t_k + \frac{\int_0^{t_k} \alpha(\tau)\, d\tau}{f_0} + \frac{\phi(t_k)}{2\pi f_0} \end{aligned} \tag{8}$$

Equation (8) shows that, the node's clock time $C(k)$ is inaccurate, as $C(k)$ differs from the reference time t_k by a mount of $\frac{\int_0^{t_k} \alpha(\tau)\, d\tau}{f_0} + \frac{\phi(t_k)}{2\pi f_0}$. The inaccuracy is due to the existence of phase noise $\phi(t)$ and the accumulated oscillator frequency variations.

3.3 Discrete model of clock

The difference between the clock's time $C(k)$ and global reference time t is referred to as ***offset***, usually measured in second, ms or us, depending on the accuracy of the clock. Let $\theta(k)$ denote the offset of clock $C(k)$ at time t_k, it is

$$\theta(k) = C(k) - t_k \tag{9}$$

As an example, Figure 2 illustrates the accumulation of offsets of two free-running oscillator clocks, namely Clock A and Clock B, where the x-axis represents the reference time t and y-axis represents the clock's offsets. Both free-running clocks have the same average clock accuracy of 10PPM, but subject to different variations due to oscillator quality tolerance and aging. It can be seen that both offsets increase over time and Clock B's offset fluctuations are significant than Clock A's, yielding Clock A is relatively accurate compared to Clock B. It is worth noting that the offsets reach to about 500μs when t approaching to 50s. The offset will keep accumulating and become larger and larger with the time continually lapses. In time-sensitive industrial applications, if the offset increases too quickly and reaches a relatively large value, the clock must be corrected to avoid potential damages to the industrial system. Considering WSN node's working conditions are usually relatively poor, and the quartz crystal oscillator frequency is vulnerable to the external environment, it is very necessary to implement periodic time synchronization in WSNs.

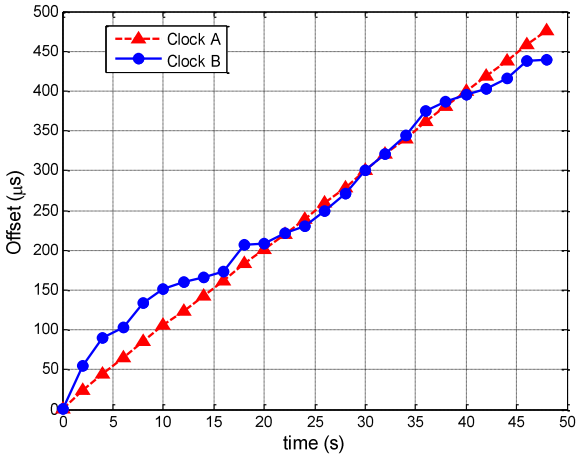

Figure 2 Two examples of free-running oscillator clocks' offsets (red triangle for drifting Clock A and blue dot for drifting Clock B) over an elapsed period of 50 seconds. Clock A is relative more accurate than Clock B.

For an inaccurate drifting clock as described by eq.(8), substituting (8) into (9) yields

$$\theta(k) = \frac{\int_0^{t_k} \alpha(\tau)\, d\tau}{f_0} + \frac{\phi(t_k)}{2\pi f_0} \tag{10}$$

It is worth noting that $1/(2\pi f_0)$ can be regarded as a scaling factor. Without loss of generality, the effects of random phase noise $\frac{\phi(t_k)}{2\pi f_0}$ can be rewritten as a discrete form $\phi(k)$ and the possibility density function (PDF) of $\phi(k)$ have the similar shape as the PDF of its counterpart $\phi(t_k)$.

As shown in eq.(5), $\alpha(t)$ represents the instant change of the oscillator's frequency at time t and $\int_0^{t_k} \alpha(\tau)\, d\tau$ represents the accumulated phase change caused by the frequency variations from t=0 to t=t_k. Usually, frequency change $\alpha(t)$ is a slow process depending on the aging factor, supply voltage and operation temperature of oscillator and $\alpha(t)$ does not change during a short period, and an average frequency change α_k can be associated to each time interval $[t_k, t_{k+1}]$. Thus $\int_0^{t_k} \alpha(\tau)\, d\tau$ can be piecewise discretized as $\sum_{i=0}^{k-1} \alpha(i)\, \tau_0$.

These two treatments above allow a discretized model of offset (10).

$$\theta(k) = \frac{\sum_{i=0}^{k-1} \alpha(i)\, \tau_0}{f_0} + \phi(k) \tag{11}$$

By introducing a term $\gamma(k) = (f(t_k) - f_0)/f_0$ to represent the deviation from the nominal frequency, it follows that $\gamma(k) = \alpha(k)/f_0$ for the period $[t_k, t_{k+1}]$. And a discretized recursive expression of the clock's offset can be obtained from eq.(11):

$$\theta(k) = \sum_{i=0}^{k-1} \gamma(i)\tau_0 + \phi(k) \tag{12}$$

$$\begin{aligned} \theta(k+1) &= \theta(k) + \gamma(k) \cdot \tau_0 \\ &\quad + \phi(k+1) - \phi(k) \end{aligned} \tag{13}$$

Let $\omega_\theta(k) = \phi(k+1) - \phi(k)$ denote the difference of phase noise in two consecutive clock counting events, the offset's transition model can be rewritten as

$$\theta(k+1) = \theta(k) + \gamma(k) \cdot \tau_0 + \omega_\theta(k) \tag{14}$$

Indeed, the normalized frequency deviation $\gamma(k)$ is referred to as ***skew***, which is a dimensionless quality in PPM. For accurate clock, $\gamma(k) = 0$, indicating no frequency changes; and for drifting clock, $\gamma(k) \neq 0$, implying the clock frequency changes by $\gamma(k) \cdot f_0$ in a unit time period. From equation (13), $\gamma(k)$ can also be understood as offset θ's change rate during period $[t_k, t_{k+1}]$. Skew of typical clock crystal in wireless sensor nodes is about in the range of tens PPM to a few hundreds PPM. In some worse cases, it may reach to 3000PPM.

The skew $\gamma(k)$ is usually affected by temperature and other environmental factors, while the skew $\gamma(k)$ is slowly varying, generally, within a small time interval, $\gamma(k)$ can be assumed constant. Various models of clock skew have been proposed and widely used. The simplest one is the constant skew model, in which skew is a constant, i.e., $\gamma(k) = \gamma$. Another model the "simple skew model" (SKM) introduced in [27], where $\gamma(k)$ is modelled as an independent random process. A better model is the so-called auto-regressive (AR) model which describes the skew as a time-varying process in an auto-regressive manner with a small perturbation [28], i.e.

$$\gamma(k+1) = p \cdot \gamma(k) + \omega_\gamma(k) \tag{15}$$

where $\omega_\gamma(k)$ is the noise with zero mean and p is the parameter of the first-order AR model, which is close to 1.

As shown in literature [29-30], both noises ω_θ(k) and ω_γ(k) are two uncorrelated random processes subject to zero-mean Gaussian distribution with variances σ_θ^2 and σ_γ^2 respectively.

Remark 1: Recalling equations (14) and (15), the oscillator clock is updated at a fixed interval of τ_0 and the value of τ_0 is adjustable (by the simulation initialization file omnetpp.ini) at the beginning of simulation. In order to distinguish τ_0 from the time synchronization interval ΔT, τ_0 is also termed as physical clock updates interval. It is worth noting that most existing simulators assume the physical clock updates and time synchronization interval are the same, which reduces the resolution of the simulated oscillator clock. In our simulator, the τ_0 is far less than the clock synchronization interval ΔT. The default value of τ_0 is 10^{-4}s and ΔT is 0.1s. This makes our physical clock updates faster and the proposed clock model is closer to the real oscillator clock.

Remark 2: There is a tradeoff between the clock resolution and simulation speed. Considering some time-sensitive applications (e.g. localization) in which the deviations of synchronized clock in the nanosecond area are of interest, a proper simulation of the crystal oscillator clocks is needed. In order to get an accurate model of reality, it is necessary to simulate the imperfections of the individual crystal clocks at good resolution. Simulating clock at higher resolution is paid at the prices of computation costs. Our simulator allows the end-user freely adjust the clock's update interval (by modifying the parameter *clock.Tcamp* in *omnetpp.ini* file), so that an acceptable tradeoff between the clock resolution and simulation speed can be achieved.

4 Mathematical Model of the PTP

In this section, the IEEE 1588 PTP protocol is formalized. The IEEE 1588 has a Master-Slave (or called leader-follower) clock hierarchy where the objective of the PTP protocol is to keep the biased clock of a ***slave node*** synchronized to an accurate lock of a ***master node*** by exchanging several time-stamped messages. The PTP defines five types of message. They are *Sync*, *Follow_Up*, *Delay_Req*, *Delay_Resp* and *Management* packets and these packets are exchanged in a delay request-response mechanism. A typical sequence of packet exchange is illustrated in Figure 3, where only four packets are required and all four packets are time-stamped. A process of time synchronization is started by the master node sending slave node a *Sync* packet that is stamped with t_1. t_1 is the

master's clock time when the *Sync* transmission starts. A *Follow_Up* packet embedding the time t_1 is needed, if the master node is not able to encapsulate t_1 into *Sync* packet. The *Sync* packet is received by the slave node at t_2 (measured by the slave's inaccurate clock) with a master to slave transmission delay d_{ms}. After receiving the *Sync* packet, the slave node replies with a *Delay_Req* message at slave's time t_3. Usually, there is a delay from t_2 to t_3 due to the packet processing time needed by the slave (such as buffering, interruption, queuing and MAC backoffs). Similarly, the transmission of *Delay_Req* has a slave-to-master transmission delay, denoted by d_{sm}. When receiving the *Delay_Req* at master's time t_4, the master put a timestamp of t_4 to *Delay_Resp*, which is sent back to the slave node at a later time. At the slave side, once all the time stamps of t_1, t_2, t_3 and t_4 are available, as specified by the PTP standard, the offset between the master clock and slave clock is now able to calculate:

$$\begin{cases} t_2 = t_1 + \theta + d_{ms} \\ t_4 = t_3 - \theta + d_{sm} \end{cases} \tag{16}$$

Then the actual master-slave offset can be calculated from (16):

$$\theta = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} + \frac{d_{sm} - d_{ms}}{2} \tag{17}$$

However, in many applications, it is not possible to measure the actual transmission delays due to the high investment on additional instruments. Therefore, in the IEEE 1588 standard, the offset is estimated as

$$\theta_M = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{18}$$

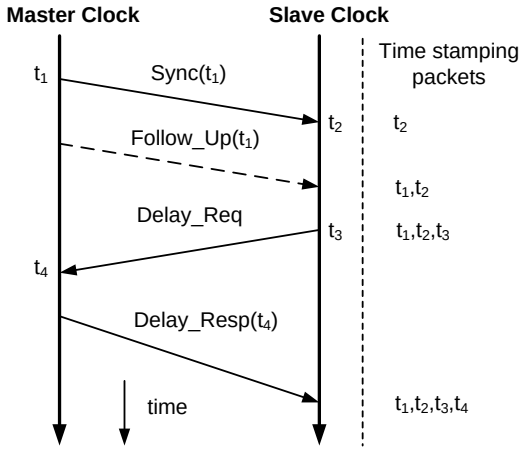

Figure 3 Time stamping and packet exchanges in PTP

Comparing θ_M and θ, one can see that the estimated offset θ_M may not be the same as the actual offset θ. Indeed, they differ by an amount of $(d_{sm} - d_{ms})/2$.

In the ideal condition of symmetric transmission delays between master and slave, i.e., $d_{sm} = d_{ms}$, which is one requirement of the IEEE 1588 standards, the error term $(d_{sm} - d_{ms})/2$ results in zero and there is no offset estimation errors.

However, in real WSNs, due to wireless media sharing, collision, retransmission and other factors, it is highly likely that the transmission delays between nodes are not symmetric, i.e., $d_{sm} \neq d_{ms}$. As a result, the offset determined by the PTP has an error

$$\frac{(d_{sm} - d_{ms})}{2} \tag{19}$$

In computer networks, the variation of transmission delays is described as delay jitter and it is important to study the delay jitters on the performance of time synchronization. One main objective of the developed simulator is to simulate the transmission delay asymmetry in mobile wireless networks realistically so that we can have a better understanding to what extent the performance of time synchronization are affected by the jitters of transmission delays.

5 Simulator Implementation – Node Structure

Note that this paper's focus is the simulation of node's clock and time synchronization, rather than the simulation of MAC and physical layers of wireless sensor nodes. As the MiXiM simulator has implemented these general modules (such as wireless channel, *World Manager*, *Connection Manager*, *mobility*, *battery*, MAC and physical layers, etc.), it is selected and reused as a starting point for developing our simulator. In this section, our main attention will be paid to the implementation of the clock model and the PTP. More specifically, two modules (*clock* and *ptp*) are discussed with details. For more information of other modules, the reader is referred to references [15, 21]. Moreover, three PTP type's packets are defined in the proposed simulator, i.e., SYNC, DREQ and DRES for the PTP's *Sync*, *Delay_Req* and *Delay_Resp* packets, respectively. In our simulator, we assume the master clock is accurate enough and there is no need to use the *Follow_Up* messages. Since the choice of a proper module structure is the key to achieving the aims of reusability and extendibility, the node's structure design is first presented and followed by the implementation of master and slave nodes.

5.1 Node's structure design

Recalling Figure 1, our simulator has two types of nodes, master node (named as *mnode* in our simulator) and slave node (named as *snode* in our simulator). Both are located in

the deployment area defined by the *World Manager* module and these nodes are connected through a wireless channel which is managed by the *Connection Manager* module. The simulator supports the simultaneous simulation of several slave nodes in one network (i.e., *snode*[0], *snode*[1], …).

Figure 4 illustrates the key simulation components of a WSN node, which are constructed according to the IP structure with three additional modules, namely, *Clock*, *ptpCore* and *ptpTmSmp*. The *Clock* module simulates WSN nodes' oscillator clocks and the clock correction realistically. The *Clock* module provides two interfaces to other modules in the node. One interface is an OMNeT++ gate which is used to simulate CPU's timer (counter) interrupt. Every time when the timer fires (i.e., the counter reaches its threshold), a message will be sent to the *ptpCore* module so that the *ptpCore* module is able to carry out periodic operations based on the time of *Clock* module, such as starting a new process of time synchronization at every one minute according to the clock time.

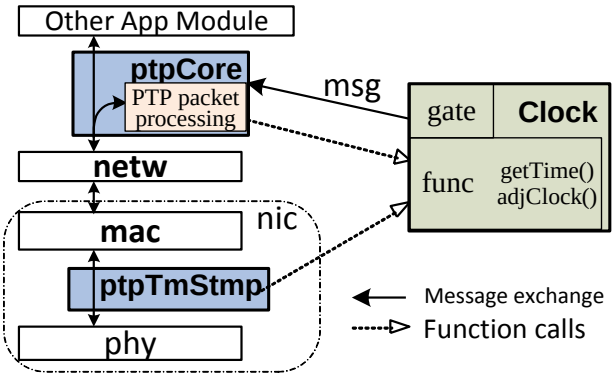

Figure 4 General structure of a WSN node

Another interface of the *Clock* module is its public member functions. By calling these interface functions, like *getTime*() and *adjClock*(), other modules acquire the time value of the *Clock* module or do clock correction easily. This is a straightforward way to simulate the processes of time acquisition and clock correction in WSN nodes.

Remark 3: It is worth pointing out that, in practice, a WSN node's operation is based on its inaccurate oscillator clock. Therefore, in the simulation, the PTP protocol should refer to the *Clock*'s drifting local time, rather than the actual global reference time provided by OMNeT++'s API simTime(), when it measures the time or schedules time synchronization events. The *Clock* module provides the drifting hardware clock's current time to other modules within the same node by using the interface function getTime(). Calling getTime() at an arbitrary time t will acquire the returned drifting local time as

$$t_k + \theta(k) + \gamma(k) \cdot (t - t_k) \tag{20}$$

where t_k represents the time of the most recent clock update and $(t - t_k)$ is the time elapsed since then.

Remark 4: The application module can also schedule an event in local clock time (rather than the OMNeT++'s global reference time) by invoking the method scheduleAtClockTime(). The *Clock* module then calculates the time left for the events during each clock update procedure and schedules the desired event by invoking the OMNeT++ API scheduleAt(). When the scheduled event fires, a message will be sent back to the caller module via the gate connection.

Due to the fact that the PTP indeed is a cross-layer protocol with time stamping performed at hardware layer for high synchronization accuracy, the functionalities of PTP are divided into two modules, namely, *ptpCore* and *ptpTmStmp*. The *ptpTmStmp* is for the PTP's time stamping process only and the *ptpCore* is responsible for all other operations (such as scheduling PTP packet exchanges, processing PTP packets and estimating clock offset and skew). The *ptpTmStmp* module is put between the *mac* layer and the *phy* layer to simulate the hardware time stamping. In our simulator, when the *phy* module receives a packet from *mac* layer, the packet transmission starts immediately. At this time, a time stamp is given to the packet to be transmitted indicating the starting time of packet transmission. When the *phy* layer finishes receiving a packet from wireless channel, it sends a packet up to the *mac* layer. At this time instance, a time stamp is given to the received packet to indicate the receiving time. Please note that all these time stamps are generated from the time value of the *Clock* module by calling *Clock* module's interface function *getTime*().

These packets go through the *mac* and network layer and reach the *ptpCore* module. In the *ptpCore* module, only the packets for PTP protocol will be processed and those packets from other application protocols are forwarded to the corresponding module. The *ptpTmStmp* module are the same for both master and slave nodes, and the *ptpCore* module in master and slave nodes are implement by different classes.

The advantages of the proposed WSN node's structure are: (1) Simulation reality. The WSN node's various layers, inaccurate oscillator clock and PTP protocols are simulated accurately and realistically. (2) Extendibility and reusability. The proposed structures are compatible with most existing simulators, e.g., MiXiM, so that we can easily integrate our PTP module into other simulators. For example, various application scenarios with different traffic loads and different MAC protocols can be easily simulated by using different layer modules or adjusting their parameters. (3) Scalability. Simulation of multiple masters and multiple slave nodes are supported, as well as the multi-hop time synchronization in a large scale WSN network.

5.2 Implementation of master node

One implementation of the master node (named *mnode*) is shown in Figure 5. The *mnode* module consists of several modules: *nic* (Network Interface Card) is a compound module of *phy* (physical), *mac* layers and *ptpTmStmp* module. The *phy* and *mac* layers are implemented by *CSMA802154* and *PhyLayerBattery* module, respectively, to simulate the TI CC2420 (IEEE 802.15.4) chip. The *netw* module is for IP network layer and is implemented by a basic IP module and the *appl* module is to simulate general applications with burst data transmission.

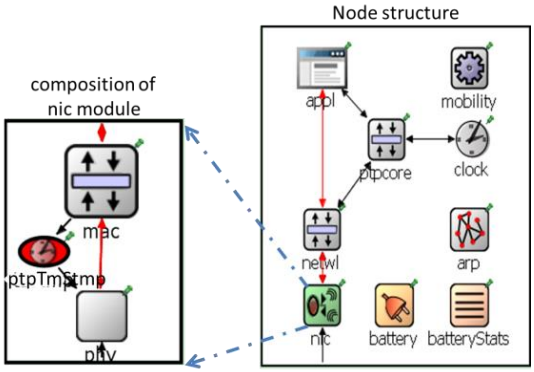

Figure 5 Implementation of Master node with accurate clock, hardware time stamping and TI CC2420 chip (IEEE 802.15.4 communication hardware).

(a) Master clock module: Since the master node is the node to which the slave nodes attempt to synchronize, the clock of the master node is assumed accurate. Let t denote the *global reference time* and $M(t)$ denote the clock of the master node at time t, that is

$$M(t) = t \tag{21}$$

Note that, in the OMNeT++ platform, the simulation time works as the global reference time t. Thus t is always equal to the output value of OMNeT++ API function simTime() that returns the simulation time immediately. Therefore, $M(t)$ is simply implemented by calling the API simTime().

(b) Time stamping module: This module is implemented by a simple module *PtpTimeStmp*. The task of time stamping is completed in its member function *handleMessage*(). When a packet arrives at the module either from the higher *mac* layer or from the lower *phy* layer, *PtpTimeStmp::handleMessage*() is invoked and it first checks if this packet is a PTP protocol packet. If not, it forwards this packet to corresponding output gate without changing the packet. If yes, it acquires a time stamp by calling interface function *getTime*() of the node's *Clock* module and attaches the time stamp to the packet's filed of *tsTX* for transmitting packet or *tsRX* for receiving packet, accordingly. More specifically, time stamps t_1 and t_3 in Figure 3 are attached to *tsTX* as transmission time stamps and t_2 and t_4 are attached to *tsRX* as receiving time stamps. Then the time stamped PTP packet is sent down to *phy* module for transmission or sent up to *mac* module for receiving, respectively. The OMNeT++ API function *getEncapsulate*() is used to check the packet type and set the time stamp recursively.

(c) Master ptpCore module: This module is implemented by a simple module *ptpMaster*. The main tasks of *ptpMaster* are: (1) It schedules the time synchronization timer at an interval of ΔT to kick off the PTP packet exchange. When the timer fires (e.g. every ΔT=10 seconds) for a new round of time synchronization, *ptpMaster* broadcasts a SYNC packet. (2) When it receives a DREQ packet from a slave node, it replies a DRES packet carrying the receiving time stamp of the DREQ packet.

5.3 Slave node implementation

Similarly, the slave node (named *snode*) has the same structure as the master node. The only differences are the *Clock* module and the *ptpCore* module. The *Clock* module is implemented by *Clock.ned* that models an inaccurate WSN oscillator clock, while as the *Clock* module in master node is an accurate clock (i.e. equal to simTime()) representing the global reference time. The *ptpCore* module in slave node is implemented by a compound module *PtpSlave.ned* that manages the time-stamped packet exchange with master node according to the IEEE 1588 protocol. Once the packet exchange finishes, the slave clock's offset is estimated and the *Clock* module's interface function *adjClock*() is invoked to correct the clock's offset and skew. Details of the slave node's *Clock* and *PtpSlave* modules will be described next.

6 Simulation of Drifting Clocks and the PTP

This section presents the implementation of the slave node's *Clock* module and *PtpSlave* module. The *Clock* module mimics the drifting oscillator clock in WSN nodes and the *PtpSlave* module implements the functions of the slave node specified by the IEEE 1588 protocol.

6.1 Implementation of Slave Clock module

The *Clock* module in a slave node is responsible for simulating the inaccurate drifting slave clock. Different from the master node's *Clock* module that always has zero offset and skew, the slave's *Clock* is a drifting clock with time-varying non-zero offset and skew as modelled in equations (14) and (15). And the slave's clock offset and skew need regularly.

In the implementation, the *Clock* module schedules a clock update timer which fires at every τ_0 second. Each time when the timer fires, clock offset $\theta(k)$ and skew $\gamma(k)$ is updated according to the model (14) and (15).

6.2 Simulating the uncertainties in time stamping

The *Clock* module provides a public member function *getTime*() for other modules to acquire the clock's present time $S(t)$ at any time t. Note that t may not necessarily equal to t_k.When the slave clock's present time $S(t)$ is required at an arbitrary time t, $S(t)$ is calculated by

$$S(t) = S(k) + \gamma(k) \cdot (t - t_k) \tag{22}$$

and returned to the caller. t_k is the time when the clock is last updated and *S*(*k*) is the slave clock's time at last clock update. The last term $\gamma(k) \cdot (t - t_k)$ represents the additional offset occurred during the period from last clock update at t_k to present time t.

However, the time stamping cannot be exactly at the same time as the transmitting/receiving events occur, due to delays caused by various reasons, such as hardware circuit's delays, the interruption queue, CPU processing delays, transmission delays, etc. In order to evaluate the performance of PTP in a realistic scenario, we develop a mechanism to simulate these delays and inaccurate time stamps in time stamping process. In fact, time stamping is the processing of reading clock's value and the uncertainties in time sampling work as a measurement noise when reading the value of a clock. Therefore, in our simulator, a random process $\omega_\eta(t)$ is implemented to represent the time stamp uncertainty of the slave clock and the inaccurate timestamps are implemented by adding $\omega_\eta(t)$ to $S(t)$ when a timestamp is required. Therefore, the *Clock* module's *getTime*() function modifies the clock's observed value as

$$S(t) = S(k) + \gamma(k) \cdot (t - t_k) + \omega_\eta(t) \tag{23}$$

which is the returned value of function *getTime*(). Usually, $\omega_\eta(t)$ is considered as a zero-mean Gaussian random noise process, with variance σ_η^2.

6.3 Implementation of PTP protocol

The PTP protocol and time-stamped packet exchange are simulated by *ptpCore* modules in both master and slave nodes. As shown in Figure 5, the master and slave node have the same node structure, but the *ptpCore* module in master node is implemented by the *PtpMaster.ned* while the slave node's *ptpCore* module is implemented by the *PtpSlave.ned*.

PtpMaster module implements the functions of the master node specified by the IEEE 1588 protocol. It triggers off the time synchronization procedure regularly by sending a SYNC packet at the specified time synchronization interval ΔT and replying slave's DREQ packet with a DRES packet.

PtpSlave module implements the functions of the slave node described by the IEEE 1588 protocol. The packet exchange with master node is mainly implemented by the *PtpSlave*'s function *PtpSlave::handleMasterMessage*() which is invoked each time when receiving a PTP packet from the master node.

The working flow of the implementation for the IEEE 1588 protocol between the master and slave node is described as follows:

(1) At *PtpMaster* module, when the time synchronization timer fires, a SYNC packet is created and sent to the slave nodes. Note that, the transmission time stamp t_1 is attached to the SYNC packet by master node's *PtpTmStmp* module.

(2) When a SYNC packet arrives at the slave node's physical layer, a receiving time stamp t_2 is attached to the SYNC packet by the slave node's *PtpTmStmp* module.

(3) The SYNC packet goes through the MAC and IP layer and is detected by the *PtpSlave* module. *PtpSlave* module records the time stamps t_1 and t_2. As specified by the IEEE 1588 protocol, *PtpSlave* waits for a uniformly distributed delay (implanted by a delay timer) before replying the master node with a DREQ packet.

(4) When the DREQ packet arrives at the physical layer and the transmission starts, a time stamp t_3 is attached to the DREQ packet by the slave's *PtpTmStmp* module.

(5) Once received the DREQ packet, master node's *PtpTmStmp* module attach a time stamp t_4 to the packet. Then the *PtpMaster* module replies with a DRES packet back to the slave. The DRES packet carries both the time stamps t_3 and t_4 that was encapsulated in the DREQ packet.

(6) When the DRES packet arrives at the slave node, its time stamps are not required by the PTP standards and the time stamp *rsTX* and *tsRX* are simply ignored by the slave node. Indeed, the time stamps t_3 and t_4 stored in the DRES packet are extracted by the *PtpSlave* module. Then the clock offset is estimated as specified by eq.(18).

(7) The function *estClock*() is invoked to carry out additional post-processing of these time measurements and offset estimates (e.g., skew estimation, averaging, the Kalman filtering, etc.).

(8) Then the *PtpSlave* calls the clock adjustment function *adjClock*() provided by the *Clock* module to correct the slave clock so that the slave clock is synchronized to the master clock as accurate as possible.

It is worth noting that, in the last two steps, the user and/or researcher should rewrite these two functions *estClock*() and *adjClock*() to implement their own specific clock offset/skew estimation and clock correction algorithms.

6.4 Clock Correction

Clock correction mechanism has a significant impact on the synchronization performance. Although the final objective of the proposed simulator is to evaluate and compare the synchronization performances of various clock correction algorithms, the main focus of this paper is simulator development for realistic simulation of oscillator clocks and PTP in mobile WSNs, due to the limited pages. In this section, a simple clock correction is presented and works as a vehicle to demonstrate the features of the proposed simulator and how the developed simulator is used to evaluate the PTP's performance. More advanced clock correction algorithms can be found at [24-26] and [27].

Skew Estimation: In our simulator, besides the standard clock offset estimation as specified in equation (18), the clock skew is estimated as well to improve the performance of clock synchronization. The skew estimate can be obtained by two consecutive synchronization processes:

$$\gamma_{\mathrm{M}}(n) = \frac{\theta_M(n) - \theta_M(\mathrm{n}-1)}{t_1(n) - t_1(n-1)} \tag{24}$$

where $\theta_M(n)$ denotes the offset estimate at present time synchronization, $\theta_M(n-1)$ represents the offset estimate at last time synchronization. Same rule applies to $t_1(n)$ and $t_1(n-1)$. Ideally, if the clock skew is constant between two consecutive time synchronization, the above estimation is accurate.

Clock correction: The *Clock* module also provides an interface function for offset and skew adjustment, which is implemented by the public member function *adjClock*() with input arguments θ_{adj} and γ_{adj}. Here, θ_{adj} and γ_{adj} denote the amount of offset and skew adjustments, respectively. In function *adjClock*(), the slave clock's offset and skew are adjusted as

$$\begin{cases} \theta(k) = \theta(k) + \theta_{adj} \\ \gamma(k) = \gamma(k) + \gamma_{adj} \end{cases} \tag{25}$$

and the adjustments take effects immediately.

The clock correction algorithm is implemented by a subfunction *calcClkCorrection*() of the slave's *ptpCore* module. Workings as a controller, *clacClkCorrection*() first calculates the control output (i.e., θ_{adj} and γ_{adj}) from the offset estimate given by the PTP and then returns. With the returned values of offset/skew adjustment, *ptpCore* module invokes the *Clock* module's interface function *adjClock*() to apply the offset/skew adjustment. It is worth noting that adjusting skew is equivalent to the procedure of adjusting frequency.

In the simulator, two offset and skew adjustment algorithms are implemented for demonstration. One is the direct correction method that uses the estimated offset $\theta_M(n)$ and $\gamma_M(n)$ straightforward as the adjustment amounts:

$$\begin{cases} \theta_{adj} = -\theta_M(n) \\ \gamma_{adj} = -\gamma_M(n) \end{cases} \tag{26}$$

Ideally, if the estimates of $\theta_M(n)$ and $\gamma_M(n)$ are accurate and good enough, the direct correction should be able to work well.

However, in real WSNs, the offset and skew estimation suffer from various noises and perturbations. As a result, the over-correction or under-correction may occur and the time synchronization performance degrades.

An improved clock correction algorithm is the attenuated correction method, in which the correction amounts area fractions of the offset's and skew's estimates:

$$\begin{cases} \theta_{adj} = -\alpha \cdot \theta_M(n) \\ \gamma_{adj} = -\beta \cdot \gamma_M(n) \end{cases} \tag{27}$$

where α and β are the attenuation coefficients taking values among (0, 1). In this method, the clock is corrected by a relatively attenuated amount of the offset and skew estimates. Over-correction can be avoided at the costs of taking a relatively longer time to synchronize the slave clock to master clock. As one can see, the direct correction is a special case of the attenuated correction when α=1 and β=1.

The clock correction algorithm in eq.(27) indeed is a proportional controller and there are have been many clock correction algorithms in the literature, such as the PID controller [24], PI controller [26] and Kalman filter [27]. These algorithms can be implemented individually and integrated into our simulator easily. It is worth noting that the values of θ_{adj} and γ_{adj} are determined by the time synchronization algorithm, not by *Clock* module. The *adjClock*() function only modifies the present clock's offset and skew by the amounts of input arguments. The end-user have freedom to modify the subfunction *calcClkCorrection*() to implement their own clock correction algorithms.

7 Simulation Results

In order to verify the clock model and study the impacts of parameter variations (e.g. clock stability, time stamping uncertainty) on the time synchronization performance, extensive simulations have been done and this section presents these results. In the simulation, the initial values of slave clock's offset and skew are set to 0μs and 10PPM, respectively. The time synchronization interval ΔT is 0.1s and the physical clock update interval τ_0 is 10^{-4}s, which is much smaller than synchronization interval ΔT. The configurations of the simulations are summarized in Table 1 as below. One typical drifting clock presented in [11] (i.e. Clock A mentioned in section 3.3) is simulated. The skew

of the clock is subject to a random perturbation with standard variance σ_γ=10^{-9} and its phase noise is of standard variance σ_θ =10^{-7}. In order to study the effect of the time-stamping uncertainty $\omega_\eta(t)$ on the time synchronization accuracy, various uncertainties of timestamps are achieved by varying the standard variance σ_η of the random process $\omega_\eta(t)$. In different runs of simulation, σ_η varies from 10^{-8} to 10^{-2}. A smaller σ_η corresponds to accurate hardware timestamp, while the larger σ_η represents fluctuating software timestamp.

Table 1 Simulation configurations

Symbol	value	Unit
ΔT	0.1	second
τ_0	1×10^{-4}	second
σ_θ	10^{-7}	second
σ_γ	10^{-9}	PPM
σ_η	[10^{-8}, 2×10^{-8}, 4×10^{-8}, 10^{-7},…10^{-2}]	second
α β	[0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.2,0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]	
Node's movement speed	0.2	m/s

These simulation configurations are set in a textual initialization file omnetpp.ini and the end-user can change the values of these parameters directly either in the OMNeT++'s GUI window or in any text editor without specific code changing and re-compiling. For example, values of α and β are set by **.*clock.alpha*=0.01 and **.clock.beta=0.4 respectively.

7.1 *Impacts of software time stamping and transmission asymmetry*

The impacts of transmission delays in the IEEE 802.15.4 MAC layer are first studied. Here we only consider the delays caused by the MAC protocols. Please note that, the default position of the time stamping module *ptpTsStmp* is between the MAC layer and physical layer to implement hardware time stamping. In hardware time stamping, the delays in MAC layer is not included in the transmission delays. In this simulation of studying MAC delays' impacts on time synchronization, the *ptpTmStmp* module is moved between the MAC layer and network layer. This is co-called software time stamping, because, in practice, such a time stamping procedure is carried out by software codes rather than hardware. As a result, the MAC delays (e.g., backoff, collision and retransmission) are included in the packet transmission delays and affect the performance of time synchronization.

The top plot of Figure 6 shows the master-to-slave delays d_{ms} and slave-to-master delays d_{sm}, where it can be seen the delays fluctuate largely among 5ms and 13ms.These fluctuation are mainly due to the random and exponential backoffs adopted in the IEEE 802.15.4 CSMA protocol. The bottom plot of Figure 6 shows the difference between d_{ms} and d_{sm} . It can be seen that the transmission delays are not symmetric, i.e. $d_{ms}(n) \neq d_{sm}(n)$, which work as the main source of offset estimation errors (see equation (19)) in the IEEE 1588 PTP.

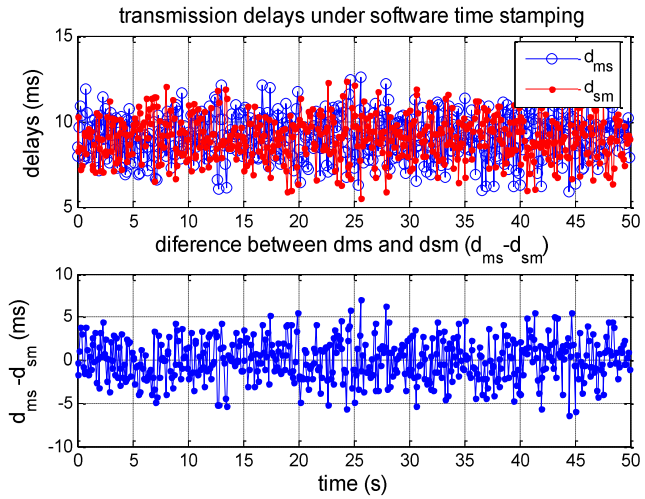

Figure 6 Transmission delays under software time stamping.

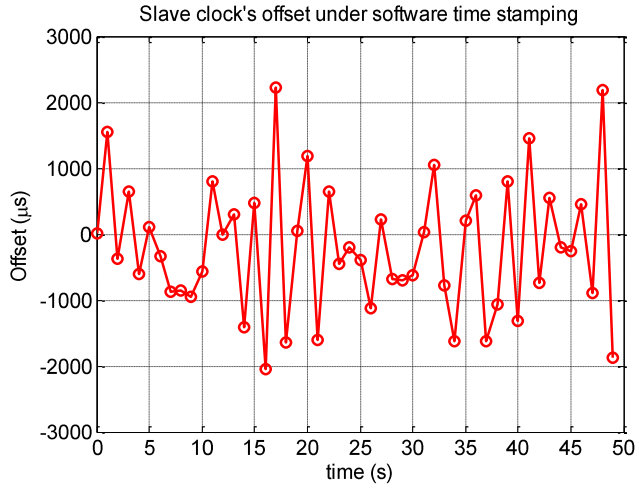

Figure 7 Slave clock's offset subject to asymmetric IEEE 802.15.4 MAC delays

The performances of the IEEE 1588 PTP subject to asymmetric MAC delays and software time stamping are depicted in Figure 7, where the offsets of the slave clock under software time stamping are presented. For the illustration purpose, only the results of Clock A are presented. It can be seen that the offset of the slave clock may be as large as 2ms, which is similar to the performance of NTP protocol. In another words, the software time

stamping degrades the performance of the PTP significantly. This verifies that hardware time stamping is critical and necessary for the PTP to achieve its designed target accuracy of micro-level.

7.2 Comparison of different clock correction algorithms

The performances of the IEEE 1588 PTP time synchronization under different clock correction algorithms are presented in Figure 8 and Figure 9. The performances of the direct correction (α=β=1) and attenuated correction (α=0.4 and β=0.03) are compared in terms of the slave clock's offsets and skews. Recalling the offsets of free-running Clock A and Clock B as shown in Figure 2, it can be seen in Figure 8 that the offsets of the corrected clock are much smaller than the offsets of free-running clock. Although the offset of free-running clock may achieve 500μs in 50 seconds, by applying the time synchronization protocol, it is always bounded between ±5 μs.

Furthermore, Figure 8 and Figure 9 demonstrate the performance improvement of the attenuated clock correction scheme. In Figure 8, the direct correction algorithm results in the offset vary among ±5μs, while the attenuated correction algorithm has a much smaller offset fluctuation of ±1 μs. As shown in Figure 9, the improvement on skew is more obvious. After about 10 seconds, the skews of the clock synchronized by the attenuated correction algorithm converge steadily to 0, while the skews of the direct correction algorithm fluctuate obviously. The clock offsets benefit from the improvements on clock skew of attenuated correction scheme.

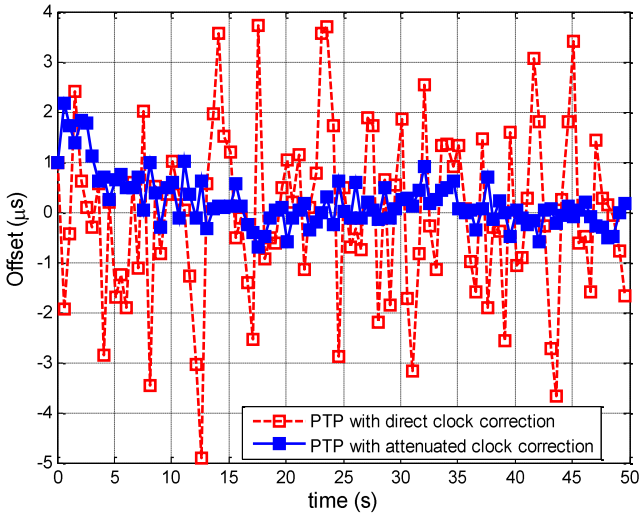

Figure 8 Offsets of Clock A under the direct correction and attenuated correction schemes

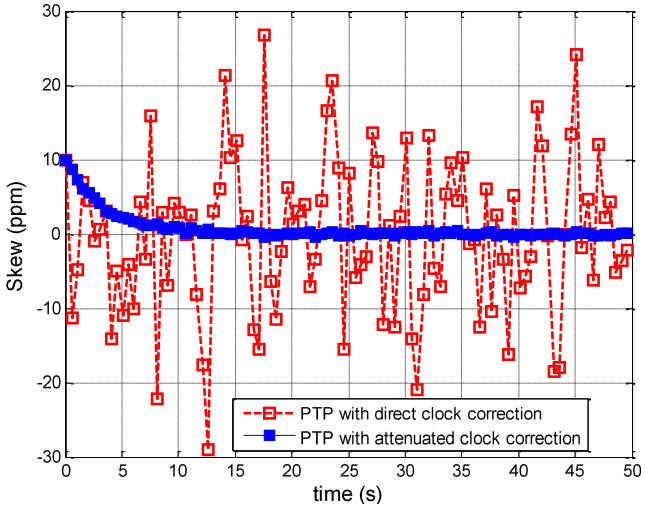

Figure 9 Skews of Clock A under the direct correction and attenuated correction schemes

Moreover, the proposed attenuated correction scheme shows a feature of convergence. As shown in Figure 8 and Figure 9, at the first several rounds of time synchronization, the offsets and skews of attenuated correction decays steady and converge to the minimal values after about 10 seconds. Then the offsets and skews are bounded in small ranges.

8 Conclusion and Future Work

In this paper, an open source simulator **TS2** is developed for the IEEE 1588 PTP time synchronization in mobile IEEE 802.15.4 WSNs. The concept and structure of the developed simulator are discussed followed by the development of a realistic model of drifting oscillator clocks with better and adjustable resolution. The detailed implementation of the drifting clock, time stamping, radio channel (physical layer), the IEEE 802.15.4 layer (i.e., the TI CC2420 chip) and the IEEE 1588 PTP are presented. Some simulation results are presented to demonstrate the features of the simulator. In particular, the impacts on the PTP performance caused by time-stamping uncertainties and the complicated MAC protocols are examined and analyzed by simulation. The simulation results show that the simple but effective algorithm of attenuated clock correction gives better performance than the direct correction approach. Moreover, the simulator has strong compatibility, extendibility and scalability. With minor changes, the simulator can be easily extended to simulate the PTP over various wireless/wired communication networks with different topologies, such as WiFi, Ethernet, Zigbee, WiMAX, MANET and Ad-Hoc networks.

The developed TS2 simulator provides an efficient and visual approach to analyze and evaluate the process and performance of the PTP time synchronization in WSNs. In the future, more advanced clock correction algorithms (e.g.

Kalman filter, PI controller, etc.) will be evaluated on this simulation platform. It is straightforward to extend the simulator to simulate multi-hop time synchronization. The multi-hop simulation will give a more practical and detailed study on the PTP time synchronization performance in real networks.

Funding

This work was partially supported by the National Natural Science Foundation of China (NSFC) under the grants 61101135 and 61231009, the Fundamental Research Funds for the Central Universities (XDJK2014C167, XDJK2012C065), the 863 Project under the grant 2014AA01A707, and Science and Technology Commission of Shanghai Municipality under the grant 13XD1403400.

Reference

1 Mills DL. "Internet time synchronization: the network time protocol." *IEEE Transactions on Communications* 10 (1991): 1482-1493.

2 Gaderer G, S. Thilo, and F. Ring et al. "A novel, wireless sensor/actuator network for the factory floor." In *Proc. of IEEE Sensors Conference* (2010): 940-945. .

3 IEEE Instrumentation and Measurement Society. *IEEE Standard 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. New York: The Institute of Electrical and Electronics Engineers, Inc. 2008.

4 Elson J, L. Girod and D. Estrin . "Fine-grained network time synchronization using reference broadcasts." In *Proc. of the 5th symposium on Operating Systems Design and Implementation (OSDI'02) and ACM SIGOPS Operating Systems Review*36 (2002): 147–163.

5 Maróti M, B. Kusy and G. Simon et al. "The flooding time synchronization protocol." In *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems*, ACM, (2004): 39–49.

6 Han T, B. Li and L. Xu. "A universal fault diagnostic expert system based on Bayesian network." In *Proc. of 2008 IEEE International Conference on Computer Science and Software Engineering*, 1 (2008): 260–263.

[7] Su W and Akyildiz I. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, vol.13, no.2, pp.384–397, 2005

[8] Lin YW, Hsu CC, Wang JS, et al. A local synchronization protocol for low-power data transmission in wireless sensor networks. *Simulation*, vol.88 no.12, pp.1419–1437, 2012

[9] Phillips J and Kundert K. Noise in mixers, oscillators, samplers, and logic an introduction to cyclostationary noise. In *Proc. of the IEEE Custom Integrated Circuits Conference, CICC'2000*, pp.431–438, 2000.

[10] Ganeriwal S, Kumar R and Srivastava MB. Timing-sync protocol for sensor networks. In *Proc. of the 1st International Conference on Embedded Networked Sensor Systems*, ACM, pp.138–149, 2003.

[11] Abubakari H and Sastry S. IEEE 1588 style synchronization over wireless link. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS)*, pp.127–130, 2008.

[12] Antoine-Santoni T, Santucci J, De Gentili E, et al. Discrete event modeling and simulation of wireless sensor network performance. *Simulation*, vol.84, no.2-3, pp.103–121, 2008.

[13] Kucuk K, Bandirmali N, Kavak A, et al. A modified sectoral sweeper-based localization estimation and its implementation in a multi-hop wireless sensor networking environment by using OPNET. *Simulation*, vol.89, no.6, pp.746–761, 2013.

[14] NS 3 official website. http://www.nsnam.org/documents.html

[15] OMNeT++ release 4.1. http://www.omnetpp.org

[16] Xu C, Zhao L, Xu Y, et al. Time synchronization simulator and its application. In *1st IEEE Conference on Industrial Electronics and Applications*, pp.1–6, 2006 .

[17] Giorgi G and Narduzzi C. Modeling and simulation analysis of PTP clock servo. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS'2007)*, pp.155–161, 2007.

[18] Ferrari F, Meier A, and Thiele L. Accurate clock models for simulating wireless sensor networks. In *Proc. of the 3rd International ICST Conference on Simulation Tools and Techniques SIMUTools'10*, ICST, Brussels, Belgium, pp.21:1–21:4, 2010.

[19] Liu Y and Yang C. OMNeT++ based modeling and simulation of the IEEE 1588 PTP clock. In *Proc. of 2011 IEEE International Conference on Electrical and Control Engineering (ICECE)*, pp.4602–4605, 2011.

[20] Hamida EB, Chelius G, and Gorce JM. Impact of the physical layer modeling on the accuracy and scalability of wireless network simulation. *Simulation*, vol.85, no.9, pp.574–588, 2009.

[21] Köpke A, Swigulski M, Wessel K, et al. Simulating wireless and mobile networks in OMNeT++ – the MiXiM vision. In *Proc. of the 1st International Workshop on OMNeT++ (SIMUTools 2008)*, Brussels, Belgium, 2008.

[22] Dai X, Mitchell JE, Yang Y, et al. Development and validation of a simulator for wireless data acquisition in gas turbine engine testing. *IET Wireless Sensor Systems*, vol.3, no.3, pp.183–192, 2013.

[23] Gaderer G, Nagy A, Loschmidt P, et al. A novel, high resolution oscillator model for DES systems. *2008 IEEE International Frequency Control Symposium*, pp.178-183, 2008.

[24] Ring F, Nagy A, Gaderer G, et al. Clock Synchronization Simulation for Wireless Sensor Networks. In *Proc. of the IEEE Sensors Conference 2010*, pp.2022-2026, 2010.

[25] Eidson JC. Measurement, control, and communication using IEEE 1588. *Springer*, 2006.

[26] Exel R and Ring F. Improved clock synchronization accuracy through optimized servo parameterization. *2013 International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, pp.65-70, 2013.

[27] Giorgi G and Narduzzi C. Performance analysis of kalman-filter-based clock synchronization in IEEE 1588 networks. *IEEE Transactions on Instrumentation and Measurement*, vol.60, no.8, pp.2902–2909, 2011.

[28] Hamilton BR, Ma X, Zhao Q, et al. ACES: Adaptive clock estimation and synchronization using Kalman filtering. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking (MobiCom'08)*, ACM,pp.152–162, 2008.

[29] Rutman J and Walls F. Characterization of frequency stability in precision frequency sources. In *Proc. of the IEEE*, vol.79, no.7, pp.952–960, 1991.

[30] Yang Z, Pan J, and Cai L. Adaptive clock skew estimation with interactive multi-model Kalman filters for sensor networks. In *IEEE International Conference on Communications (ICC)*, pp.1-5, 2010.