Identifying and Verifying Clock Synchronization Protocol Parameters

Sho Fujita
Yokogawa Electric Corporation
Shou.Fujita@jp.yokogawa.com

Kenji Ooishi
Yokogawa Electric Corporation
Kenji.Ooishi@jp.yokogawa.com

Yosuke Ishii
Yokogawa Electric Corporation
Yosuke.Ishii@jp.yokogawa.com

Masato Yamaji
Yokogawa Electric Corporation
Masato.Yamaji@jp.yokogawa.com

ABSTRACT

The Internet of Things(IoT) is a novel paradigm, where things are in the majority position of producers and consumers of traffic. IoT is expected to have a strong impact on our everyday life, i.e., the physical space around us. In the physical space, the notion of time is so universal that the information there is often combined with timestamps. To make them consistent among IoT nodes, applications need a clock synchronization mechanism, which has been an important topic in the research community. Whereas earlier studies focused mainly on improving synchronization accuracy, we focus on improving efficiency to achieve a given synchronization accuracy in a given environment. This viewpoint is important because applications do not always require the best synchronization accuracy and only limited resources are available for IoT nodes. We propose methods to identify and verify appropriate parameters for clock synchronization protocols. Because our methods are not limited to a specific protocol, we introduce a clock synchronization protocol model that generalizes a class of clock synchronization protocols and discuss our methods based on it. Our methods are demonstrated on a trace of time information collected in our testbed.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Communication Networks—*Network Protocols*

General Terms

Performance, Verification

1. INTRODUCTION

The Internet of Things(IoT) is a novel paradigm, which is getting popularity in both academic and industrial communities. In contrast to the current Internet where humans are in the majority position as producers and consumers of traffic, IoT envisions that the position will be taken over by things that interact with each other without human intervention[1]. The things can be either a *smart object*, which is equipped with processing and networking capabilities, or just an object that is externally observed by other smart objects. In latter case, the objects are often discussed in the context of wireless sensor networks(WSNs), which become feasible as radios and sensors decrease their cost. Anyway, IoT is expected to have a strong impact on our everyday life, i.e., the physical space around us.

In the physical space, the notion of time is so universal that information gathered by IoT nodes is often combined with timestamps. To make them consistent among the IoT nodes, applications in IoT need a clock synchronization mechanism. It is also needed by mechanisms coordinating among IoT nodes, such as MAC protocols. Thus, the clock synchronization has been an important topic in the research community. A lot of clock synchronization protocols have been proposed so far[2][3][4]. Their main focus was improving synchronization accuracy. For example, *Flooding Time Synchronization Protocol*(FTSP) has achieved the best per-hop synchronization accuracy of the 1 microsecond range.

However, the best synchronization accuracy is not always needed.

First, requirements for synchronization accuracy depend on objects or phenomena monitored by IoT. The countersniper system proposed by Simon et al. detects muzzle blasts and acoustic shockwaves to locate shooters[5]. It records the timestamps when the muzzle blasts and acoustic shockwaves arrive and calculates the shooter location estimates from them. Although the speed of sound is relatively high, the synchronization accuracy of 1 millisecond is sufficient to get location estimate with an error of 1 foot. The wireless network diagnosis mechanisms need more accurate clock synchronization. To infer interfering links, they need to detect simultaneous transmissions among the nodes[6][7]. According to Cheng et al., the clock synchronization accuracy of 20 microseconds is sufficient, which is the slot size of IEEE 802.11bg specifications[6]. Second, the more accurate clock synchronization is, the more it consumes energy, which is a critical resource in IoT nodes. The situation is complex because synchronization accuracy sometimes contributes to energy saving. For example, synchronized MAC protocols exploit synchronized clocks to reduce energy consumption. They share communication schedule among nodes in advance and make the nodes turning off its radio until they actually transmit or receive frames. To receive the frame even in the case that nodes communication schedule is not shared accurately due to their clock drift, the nodes need to stay in the listen state for an appropriate time. S-MAC protocol[8], which is the popular implementation of the synchronized MAC protocol, stays in the listen state for 0.5 seconds, which is 10^5 longer than typical drift rate.

It is important to appropriately set parameters, such as communication and synchronization intervals, to efficiently meet given application requirements in a given environment. Because the protocol parameters depend on the environment where the applications are deployed, we cannot predefine them in the factory. In this paper, we propose methods to identify and verify the parameters of clock synchronization protocols. Because our methods are not limited to specific protocols, we discuss the methods using a protocol model that generalizes a class of clock synchronization protocols. The protocol model represents the predominant clock synchronization protocols[2][3][4]. The first method takes a trace of two clocks' timestamps and statistically identifies appropriate parameters. The second method verifies that the identified protocol parameters meets application requirements in a given environment.

The organization of this paper is as follows. In Section 2, we introduce a protocol model that generalize clock synchronization protocols. Then, we discuss the statistical method to identify appropriate parameters, i.e., communication and synchronization intervals in Section 3. In Section 4, we discuss the verification method that ensure the parameters identified by the statistical method meet synchronization requirements using data on clock rate stability. Then, we consider the future direction of our methods in Section 5, and we conclude this paper in Section 6.

2. PROTOCOL MODEL

We first introduce a protocol model that generalize a class of clock synchronization protocols. In the following sections, our discussion will be based it. We take this approach because ideas to identify and verify protocol parameters, such as communication and synchronization interval, are not limited to specific clock synchronization protocols. The protocol model keeps our discussion general enough to apply to a broad range of clock synchronization protocols.

2.1 Overview

Our protocol model deals with local synchronization between two neighboring nodes: a server and a client. The server has a reference clock or the true clock, and the client tries to synchronize its clock to the server's. Although local synchronization does not meet all application requirements, it is a building block to understand global synchronization of a whole network[9]. We will study the global synchronization in our future works.

We model the underlying clocks before the clock synchronization protocol. Let t be the true clock, which runs at the ideal rate of 1 second per second, and has origin $t = 0$ at some arbitrary instant. In practice, nodes are equipped with imperfect clocks; their rate is different from the ideal one and not even time-varying. On the other hand, their rate is stable enough to be treated as constant for a short time period[10]. Hence, the time scales of the server and client can be expressed as linear functions of the true clock t:

$$C_s(t) = r_s t + C_{s0} \tag{1}$$

$$C_c(t) = r_c t + C_{c0} \tag{2}$$

where r_s and r_c are a constant clock rate of the server and client, respectively. We set $C_{s0} = C_s(0)$ and $C_{c0} = C_c(0)$. Because t is not directly observable, we eliminate t from the above equations.

$$\begin{aligned} C_s(t) &= r_s(\frac{C_c(t) - C_{c0}}{r_c}) + C_{s0} \\ &= (r_s/r_c)C_c(t) + (C_{s0} - C_{c0}(r_s/r_c)) \\ &= r'_s C_c(t) + C'_{s0} \end{aligned} \tag{3}$$

Now that $C_s(t)$ can be expressed as a linear function of $C_c(t)$. We call the coefficients r'_s and C'_{s0} in Equation (3) a *clock relation.* Please note that Equation (3) also holds well only for the short time period, which we will identify in the next section.

In this paper, we regard clock synchronization as maintaining the clock relation; if we know it, the server's timestamp $C_s(t_0)$ can be calculated from the client's timestamp $C_c(t_0)$ measured at the instant t_0. This viewpoint is different but consistent with that of existing studies, which manipulates the clock settings of the client. Because the relation holds for limited periods, our protocol model periodically updates the parameters. The process of our protocol model consists of two phases: *reference point creation* and *clock relation inference.* In the reference point creation phase,

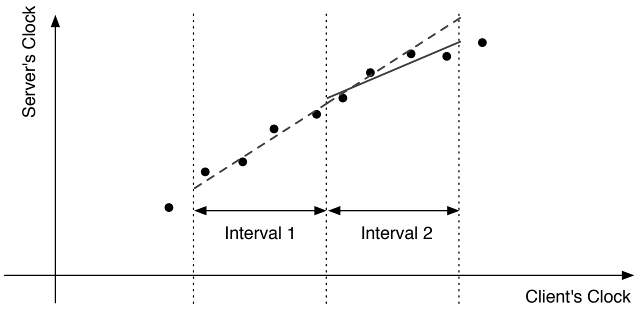


Figure 1: Each point represents a reference point. A clock relation is inferred in each synchronization interval and used in the next synchronization interval.

the server and client exchange their timestamps and create *reference points*. A reference point is a pair of timestamps measured by both the server and client at the same instant. For example, the reference points measured at instants t_0 and t_1 are $(C_s(t_0), C_c(t_0))$ and $(C_s(t_1), C_c(t_1))$, respectively. In the clock relation inference phase, the clock relation is calculated using linear regression on the reference points.

The inferred clock relations are used as shown in Figure 1. The client and server create reference points at an interval called *communication interval*. The reference points are divided by another interval called *synchronization interval*. A clock relation is calculated from the reference points within a synchronization interval. In Figure 1, each point corresponds to a reference point, whose x and y coordinates are timestamps of the client and server, respectively. The dashed line is created from the reference points in Interval 1 and then used to estimate the server's timestamps in Interval 2.

In the following part of this section, we will explain each of the phases in more detail. And, to prove that our protocol model is general enough, we explain how the existing clock synchronization protocols fit into our protocol model.

2.2 Reference Point Creation

To create a reference point, the client needs to know server's timestamp; the server needs to transmit it to the client through a wireless channel. In general, wireless transmission suffers from delays, which cause errors in reference points. Maróti et al. pointed out the seven sources of delays in wireless transmission[4]:

1. Propagation
2. Transmission/Reception
3. Encoding/Decoding
4. Byte Alignment
5. Send/Receive
6. Access
7. Interrupt Handling

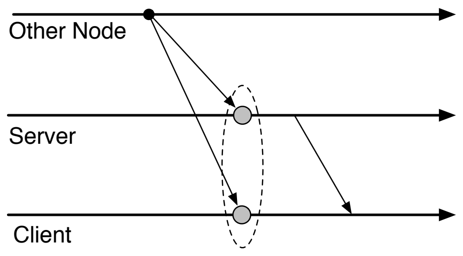


Figure 2: Creation of a reference point in RBS. A reference broadcast triggers the server and client to record timestamps at the same time, and the server's timestamp is delivered to the client later.

The propagation delay is highly deterministic and depends only on the distance. Although we do not always know the distance between the server and client, the propagation delay is often negligible in practical deployment of wireless sensor networks. For example, it is less than 1 microsecond if the distance is less than 300 meters. The transmission/reception, encoding/decoding, and byte alignment delays are deterministic and calculated from information included in a frame; We can normalize the client's timestamp by subtracting them. The send/receive, access, and interrupt handling delays are nondeterministic and major causes of errors in reference point creation. The existing protocols implement mechanisms to reduce the effect of these nondeterministic delays.

2.2.1 Reference Broadcast Synchronization

Reference Broadcast Synchronization(RBS) creates reference points only from reception timestamps[2]. Thus, RBS is often categorized into *receiver-to-receiver synchronization*[11]. To let the server and client receive a frame and record the timestamps of the reception, the third nodes periodically transmit *reference broadcast frames*. The reference broadcast frames do not include any time information; they simply trigger the server and client to record the timestamps of their reception at the same time. The server's timestamp need to be delivered to the client in another frame, as shown in Figure 2.

In RBS, the send and access delays are eliminated from reference point creation because its timestamps are taken for the single frame. Although the original paper does not indicate it, RBS can also eliminate the receive delay by leveraging MAC-level timestamping.

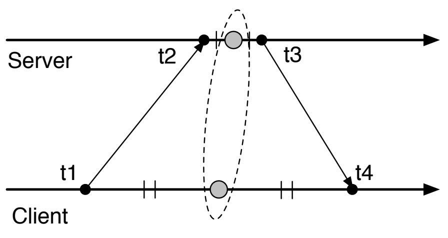


Figure 3: Creation of a reference point in TPSN.

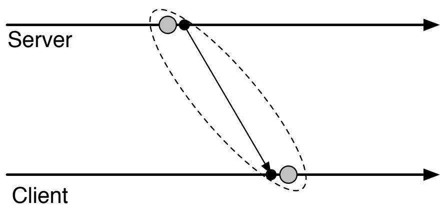


Figure 4: Creation of a reference point in FTSP.

Therefore, the interruption handling delay is the only cause of errors in reference point creation of RBS.

2.2.2 *Timing-sync Protocol for Sensor Networks*

Timing-sync Protocol for Sensor Networks(TPSN) creates a reference point from a 2-way message exchange between the server and client[3]. This is because it is categorized into *sender-to-receiver synchronization*[11]. As shown in Figure 3, the client node transmits a message with the timestamp *t1* at the transmission. The server node replies to the message with a message including the reception timestamp of the last message *t2* and transmission timestamp of this message *t3*. From the four timestamps, the client creates a reference point$(t1+t4, t2+t3)$.

TPSN increases the synchronization accuracy exploiting the software architecture specific to WSN nodes. In general, WSN nodes can control their radio transmission and reception procedures step by step. They can include the timestamp just before transmitting the bytes that will contain it and record the timestamp just before receiving the bytes. Therefore, it can eliminate the send/receive and access delays. Moreover, because of its 2-way message exchange, TPSN can measure propagation time and eliminate its affect, and reduct timestamp errors exploiting an average of two timestamps. such as t1 and t4; t2 and t3.

2.2.3 *Flooding Time Synchronization Protocol*

FTSP exploits MAC level timestamping and eliminates nondeterministic delay from its synchronization transaction[4]. FTSP is different from TPSN because it adopts 1-way message exchange; only the server node transmits the frames.

FTSP eliminates its timestamp errors to 1.4 microsecond in average, 4.2 micro second at maximum by compensating bit alignment time and exploiting multiple measurement to create a timestamp.

2.3 Clock Relation Inference

In the clock relation inference phase, the client node infers the clock relation between itself and the server. Because Equation (3) captures a relation that holds temporally, the client updates the clock relation periodically. Although each model expressed by Equation (3) is too naive to capture clock drifts, the continuously generated models as a whole can capture the clock drift. Our protocol model applies linear regression on the reference points to get the clock relation.

2.3.1 *TPSN*

On the clock relation inference, TPSN poses a limitation on the underlying clock model; it does not consider the relative rate r'_s but only the offset C_s. As stated in the last subsection, TPSN creates a reference points from the average of two timestamps, and thus reduces the time stamp errors included in the reference points. However, it exploits only 1 reference point to synchronize the clocks once and thus it cannot infer the skew. If clock's inherent accuracy is 40ppm, which is picked up from [3], the clock error may increase by 40 microseconds per second at maximum. When applications have stringent requirements on the synchronization accuracy, TPSN must adopt very short communication and synchronization intervals.

2.3.2 *RBS and FTSP*

Both RBS and FTSP fit our protocol model well; they apply linear regression to multiple reference points to calculate both r'_s and C'_1. Compared to TPSN, RBS and FTSP can eliminate timestamp errors from multiple reference points and thus provide more accurate synchronization.

3. PARAMETER IDENTIFICATION

In this section, we propose a method to identify protocol parameters, i.e., communication intervals, which is denoted by I_c and synchronization intervals, which is denoted by I_s. In each of synchronization intervals, there are (I_s/I_c) reference points on average. To identify the protocol parameters, we need to consider not only the application requirements but also the environment where the application is deployed. This is because clock synchronization protocols depend on their underlying clocks, which are affected by environmental factors, such as temperature and atmosphere pressure.

3.1 Error Analysis

We first formalize synchronization errors. A clock relation, which is expressed by Equation 3, is inferred from reference points in a synchronization interval. The clock relation is used to calculate timestamps on the server's time scale until the next clock relation is inferred. In Figure 1, the dashed and solid lines illustrate the clock relation calculated from the reference points in Interval 1 and Interval 2, respectively. The synchronization error E_s in Inteval2 is the difference in y-coordinate between the dashed line and each of reference points because the timestamps on the server's time scale are estimated using the clock relation calculated in Interval 1.

The synchronization error E_s can be divided into the *reference point error* E_{rp} and the *clock relation error* E_{cl}, which correspond to the phases of the protocol model we explained in Section 2.

$$E_s = E_{rp} + E_{cl} \tag{4}$$

The reference point error E_{rp} is a error in the reference point creation phase. In Figure 1, it is the difference in y-coordinate between each of the lines and the reference points from which the line is calculated. The clock relation error E_{cl} is a error in the clock relation inference phase. In Figure 1, it is the difference in slope between the dashed and solid lines. By separating the synchronization error into the errors corresponding to the protocol model phases, we can understand the error causes more deeply as we will see shortly.

3.2 Trace Analysis

To evaluate the proposed identification method, we apply the method to an eight day long trace of time information in a testbed. The testbed is composed of three nodes in an air-conditioned room. The nodes have a AMD LX800 processor running at 500 MHz with 256 MB DDR memory. They are embedded with an IEEE 802.11 network card with Atheros AR5414 run by ath5k device driver. One of the nodes is configured to be an access point, and transmits a beacon frame once per 100 milliseconds. The other nodes are configured to be monitoring nodes instead to be stations associating to the access point. This is because we need to disable the native clock synchronization mechanism of IEEE 802.11 (TSF). The two monitoring nodes receive the beacon frames and record the timestamps of their reception. Then, they exchange the timestamps and create reference points from them in a way like RBS [1].

[1]Actually, we needed a pre-process that filters reference points that included errors that are larger than 20000 microseconds. We believe the errors are caused by the operating system that records timestamps in software interruptions but not hardware interruptions. This problem should be eliminated if we reimplement the part of the operating system.

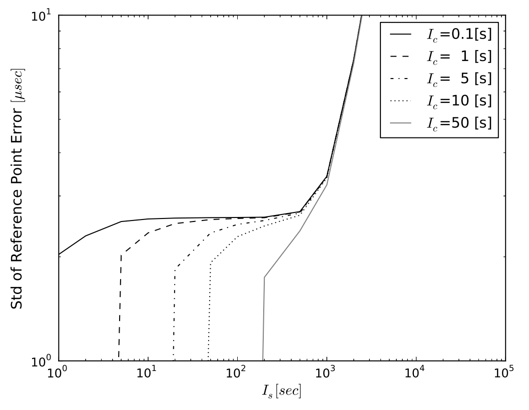


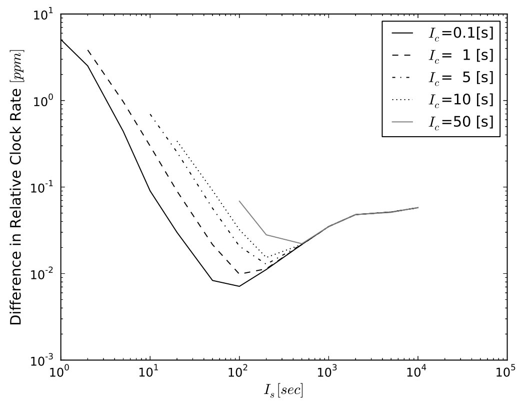
Figure 5: Errors in reference point creation.

3.2.1 Reference Point Error

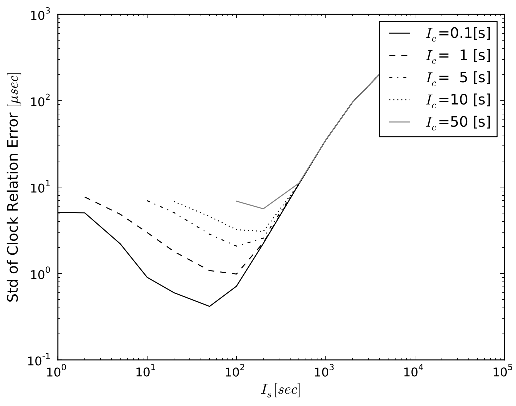
The standard deviation of the reference point errors calculated from the trace are shown in Figure 5. With relatively small I_s for each of communication intervals I_c, the reference point errors are quite small or often equal to 0. If the number of reference points (I_s/I_c) is small the inferred linear function suffers from the reference point errors; the true and inferred clock relations deviate. Especially if (I_s/I_c) is equal to 2, the function always runs exactly on the two reference points; the reference point errors are 0. As (I_s/I_c) increases, the inferred clock relation should get closer to the true one. With synchronization intervals up to around $100[s]$, reference point errors get close to and stay around $2.6[\mu s]$. This errors should be caused by nondeterministic interrupt handling delays. With synchronization intervals larger than $100[s]$, reference point errors drastically increase. This is because Equation (3) does not hold well in that large synchronization intervals.

3.2.2 Clock Relation Error

In Figure 6(a), differences in clock rates between successive synchronization intervals are shown. The curve in Figure 6(a) is similar to *allan deviation* presented in [10]. The allan deviation expresses a stability characteristics of one clock whereas our graph expresses a stability characteristics of a pair of clocks communicating through a nondeterministic channel. Moreover, only the two timestamps at edges of intervals are used to calculate clock rates in allan deviation whereas our model exploits as many reference points as possible. In Figure 6(a), the differences in clock rates drop sharply with a slope of less than -1 because the number of reference points (I_s/I_c) refines statistical inference of clock rates. This coincides with the reason why the reference point errors are small in Figure 6(a). With synchronization interval I_s around $100[s]$, the differences climb steadily



(a) Standard deviation of rate difference between successive synchronization interval.



(b) Errors in clock relation inference, which is multiplication of rate difference by synchronization interval.

Figure 6:

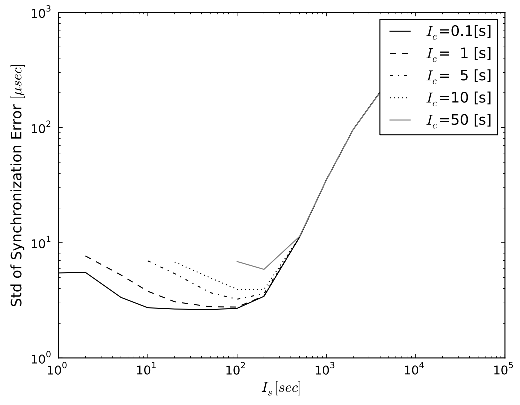


Figure 7: Synchronization stability, which is addition of errors in reference point creation and clock relation inference.

for the same reason as the reference point errors.

The clock relation error E_{cl} is calculated by multiplying the difference in clock rates between successive synchronization intervals by synchronization interval I_s. The clock relation errors are shown in Figure 6(b).

3.2.3 Synchronization Error

Finally, the protocol parameter I_s and I_c can be identified from the synchronization errors, which are calculated by adding the reference point errors and clock relation errors and shown in Figure 7. The synchronization errors express the statistical stability of clock synchronization in the testbed environment. For a given synchronization accuracy, we can identify both I_s and I_c. For each of I_c, I_s that achieves the most accurate synchronization is identified and we call it I_s*. Because larger I_s provides more efficient clock synchronization, it is useless to pick up I_s that is smaller than I_s*. Therefore, we first select I_c that sufficiently achieves the required accuracy and then I_s between I_s* and the upper bound that meets the requirement.

4. PARAMETER VERIFICATION

In the last section, we identify protocol parameters I_s and I_c for a trace collected in an environment. The protocol parameters are proved to statistically work well for the environment, but not to work in any environments. In this section, we verify that the protocol parameters meet the synchronization requirement in a bottom-up approach.

To verify the protocol parameters, we clearly define not only application requirements but also the environment where the application is deployed. For the application requirements, we consider only the clock synchronization accuracy A_s. We verify that the equation

$$E_s < A_s \tag{5}$$

holds for the environment. For the environment, there are some environmental factors that cause the underlying clocks to drift, such as temperature, atmosphere pressure, voltage and so on. In this paper, we consider only the temperature because it is the dominant factor of clock drifts.

To be concrete, we explain the verification method using some practical application requirement and environment.

The application requires some synchronization accuracy, say, $A_s = 50[\mu s]$ and we pick up a synchronization interval $I_s = 200[s]$ using the proposed identification method. Assuming that the clock relation error is dominant for $I_s \approx 200[s]$, we must keep the difference in relative clock rate between successive synchronization

Table 1: Clock rate deviation from the true clock.

temp. [°C]	clock 0 [ppm]	clock 1 [ppm]	clock 2 [ppm]	clock 3 [ppm]	clock 4 [ppm]
85	2.19	-4.80	-4.01	-8.60	-2.80
80	0.11	-6.20	-6.01	-9.21	-4.92
75	-1.74	-7.20	-7.63	-10.36	-6.65
70	-2.31	-8.93	-9.03	-11.45	-7.23
65	-2.91	-10.08	-10.42	-11.64	-7.64
60	-2.79	-9.82	-11.34	-11.50	-7.71
55	-2.68	-9.42	-11.42	-10.99	-7.20
50	-2.30	-9.05	-10.95	-10.32	-6.37
45	-1.75	-7.98	-10.01	-9.48	-5.15
40	-1.12	-6.64	-9.22	-8.18	-3.92
35	-0.42	-5.48	-8.11	-7.13	-2.98
30	0.56	-4.25	-6.62	-5.84	-1.61
25	1.66	-2.82	-4.73	-4.35	0.07
20	2.64	-1.86	-3.01	-3.01	1.34
15	3.98	-0.61	-1.61	-1.68	2.72
10	4.43	0.55	-0.37	-0.43	3.88
5	4.93	1.68	0.54	0.61	4.57
0	5.43	2.23	1.40	1.10	4.88
-5	5.74	2.39	1.60	1.14	4.92
-10	5.98	2.21	1.66	1.33	5.03

intervals below

$$A_s/I_s = 50[\mu s]/200[s] = 0.25[ppm]. \quad (6)$$

Therefore, required synchronization accuracy can be achieved, if differences in relative clock rate per minute are below

$$0.25[ppm]/(200/60)[min] = 0.075[ppm/min]. \quad (7)$$

We next define the environment that affects the rates of the underlying clocks. In Table 1, rate deviation for five different clocks are shown, which is provided by the IEEE 802.11 network vendor. The absolute values of the rate deviation are large, up to 12 ppm, which cause an error up to 12 microseconds per second. However, our protocol model does not use the naive rate but the relative rate, and thus can tolerate this large deviations. To estimate the worst-case changes, we consider the severest environment in practice, where temperature cycles between the upper and lower bound of the guaranteed operating range of the clocks, that is 85 and -10 [°C], respectively. In the left side of Table 2, the clock rate differences between successive temperature measurement points are shown. The light and dark gray cells indicate the fastest and slowest changes at each of the temperature ranges. In the environment, the time take for temperature to moves from a temperature measurement point to another is $(60[min] \times 12[hour])/19 = 37.89[min]$. In the right side of Table 2, the maximum rate difference is calculated by subtracting the values in the light gray cells by those in the dark gray. Then, they are divided by 37.89[min] and we get the maximum rate differences per minute, which are all below 0.075[ppm]; the protocol parameters meet the application requirement.

Table 2: Differences in relative clock rate between successive measurement points of temperature.

temp range [°C]	clock 0 [ppm]	clock 1 [ppm]	clock 2 [ppm]	clock 3 [ppm]	clock 4 [ppm]	max diff [ppm]	per min [ppm]
85 to 80	2.08	1.40	2.00	0.61	2.12	1.51	0.040
80 to 75	1.85	1.00	1.62	1.15	1.73	0.85	0.022
75 to 70	0.57	1.73	1.40	1.09	0.58	1.16	0.031
70 to 65	0.60	1.15	1.39	0.19	0.41	1.20	0.032
-65 to 60	-0.12	-0.26	0.92	-0.14	0.07	1.18	0.031
-60 to 55	-0.11	-0.40	0.08	-0.51	-0.51	0.59	0.016
-55 to 50	-0.38	-0.37	-0.47	-0.67	-0.83	0.46	0.012
-50 to 45	-0.55	-1.07	-0.94	-0.84	-1.22	0.67	0.018
-45 to 40	-0.63	-1.34	-0.79	-1.30	-1.23	0.71	0.019
-40 to 35	-0.70	-1.16	-1.11	-1.05	-0.94	0.46	0.012
-35 to 30	-0.98	-1.23	-1.49	-1.29	-1.37	0.51	0.013
-30 to 25	-1.10	-1.43	-1.89	-1.49	-1.68	0.79	0.021
-25 to 20	-0.98	-0.96	-1.72	-1.34	-1.27	0.76	0.020
-20 to 15	-1.34	-1.25	-1.40	-1.33	-1.38	0.15	0.004
-15 to 10	-0.45	-1.16	-1.24	-1.25	-1.16	0.80	0.021
-10 to 5	-0.50	-1.13	-0.91	-1.04	-0.69	0.63	0.017
-5 to 0	-0.50	-0.55	-0.86	-0.49	-0.31	0.55	0.015
-0 to -5	-0.31	-0.16	-0.20	-0.04	-0.04	0.27	0.007
-5 to -10	-0.24	0.18	-0.06	-0.19	-0.11	0.42	0.011

5. DISCUSSION

In this paper, we verify the protocol parameters identified by the statistical method using the data on the clock rate deviation as shown in Table 1. We fortunately got the data from an IEEE 802.11 network vendor, but it is not the case; the rate(frequency) stability of the clocks is not usually described on their specifications. A typical specification of a clock describes only the upper bound of the clock rate deviation, which only ensures it does not go apart from the true clock faster than the given clock rate. It is useful for protocols, such as TPSN, because they rely directly on the clock rates. However, when we consider protocols that use relative clock rate, which is r'_s in a clock relation, it is not so important because the deviation of the two clock rates can be compensated. As the more protocols adopt the relative clock rate, the specification of clock rate stability will be more helpful. The stability on other conditions, such as atmosphere pressure and voltage, will be also helpful to extend the verification method.

6. CONCLUSION

In this paper, we propose methods that identify and verify appropriate parameters of clock synchronization protocols for given synchronization accuracy and environment. Our methods help implementors of IoT systems meet synchronization accuracy required by applications as efficiently as possible. It also verifies the parameters assuming some clock characteristics and environment where the applications are deployed. Although the applicability of the verification is currently limited, we believe we can improve it as environments are defined clearly and more detailed rate characteristics is included in the clock specifications.

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 54(15):2787–2805, October 2010.

[2] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *The 5th Symposium on Operating Systems Design and Implementation*, pages 147–163, December 2002.

[3] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *The First ACM Conference on Embedded Networked Sensor Systems(SenSys)*, pages 138–149, November 2003.

[4] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi. The flooding time synchronization protocol. In *The 2nd International Conference on Embedded Networked Sensor Systems(SenSys)*, pages 39–49, November 2004.

[5] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and D. Frampton. Sensor network-based countersniper system. In *The 2nd International Conference on Embedded Networked Sensor Systems(SenSys)*, pages 1–12, November 2004.

[6] Y.-C. Cheng, J. Bellardo, and P. Bënko. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *the 2006 conference on Applications, technologies, architectures, and protocols for computer communications(SIGCOMM)*, pages 39–50, October 2006.

[7] N. Ahmed, U. Ismail, and K. Papagiannaki. Online estimation of rf interference. In *the 2008 ACM CoNEXT Conference*, December 2008.

[8] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *The Conference on Computer Communications(INFOCOM)*, pages 1567–1576, June 2002.

[9] C. Lenzen, P. Sommer, and R. Wattenhofer. Optimal clock synchronization in networks. In *The 7th International Conference on Embedded Networked Sensor Systems(SenSys)*, pages 225–238, Novemnber 2009.

[10] D. Veitch, S. Babu, and A. Pàsztor. Robust synchronization of software clocks across the internet. In *the 4th ACM SIGCOMM conference on Internet measurement(IMC)*, pages 219–232, October 2004.

[11] B. Sundararaman, U. Buy, and A. D. Kshemkalyani. Clock synchronization for wireless sensor networks: A survey. *Ad Hoc Networks*, 3(3):281–323, May 2005.