Event Reference Synchronization (ERS): An Event-Based IoT Clock Synchronization

Shashi Prabh
SEAS, Ahmedabad University
Ahmedabad, India
shashi.prabh@ahduni.edu.in

***Abstract*—This paper addresses clock synchronization in networks of wirelessly connected nodes, such as, Internet of Things (IoT) and Industrial Internet of Things (IIoT) networks. Synchronizing wirelessly networked sensor nodes in large deployments poses some unique challenges. Though the Precision Time Protocol (PTP), defined in the IEEE 1588 standard, is becoming the dominant clock synchronization method, its requirements of transparent bridges and specialized network interface cards are at odds with the low-cost devices, whereas, low-cost limited-capability sensor nodes are a practical necessity in large-scale deployments. This paper addresses this gap. It presents Event Reference Synchronization (ERS), an asynchronous low-overhead clock synchronization method for IoT and IIoT networks consisting of predominantly low-cost devices. ERS is capable of providing secure time reference service. Evaluation of ERS on a proof-of-concept testbed was carried out where all its components were implemented on a low-cost IoT platform. ERS achieved 85 microsecond accuracy despite using software timestamping.**

I. Introduction

Clock accuracy is important in Internet of Things (IoT) as well as in Industrial IoT (IIoT)[1]. Networked sensor nodes in an industrial plant facilitate fault detection, fault prediction and real-time optimization of plant operations. Accurate timestamping of sensor readings is necessary for making correct causal inference needed. Some networking protocols and applications also require clock synchronization. Further, battery-powered IoT nodes are duty-cycled to conserve energy. Measurement data on a low-cost device showed that the accuracy of a 50 % duty-cycled node fell to 470 ppm and that of a 20 % duty-cycled device fell to 2580 ppm against the baseline of 45 ppm in 100 % duty-cycled node (Fig. 1). Since duty-cycled nodes tend to loose accuracy when sleeping, it is imperative that the nodes be resynchronized after wake up. An IoT ecosystem can also be distributed across multiple geographical areas. Therefore, synchronizing clocks to an unambiguous external time reference such as GMT is needed.

Network Time Protocol (NTP), developed to synchronize computers connected over the Internet [1], achieves accuracy usually in the range of 5 ms - 100 ms [2]. Precision Time Protocol (PTP) was developed to fulfill sub-microsecond accuracy needed in industrial control applications which NTP could not meet [3]. Global Navigation Satellite Systems (GNSS) based clock synchronization can achieve 10 ns accuracy [4] but

[1]For the sake of brevity, the term IoT is used in this paper to refer to both IoT and IIoT.

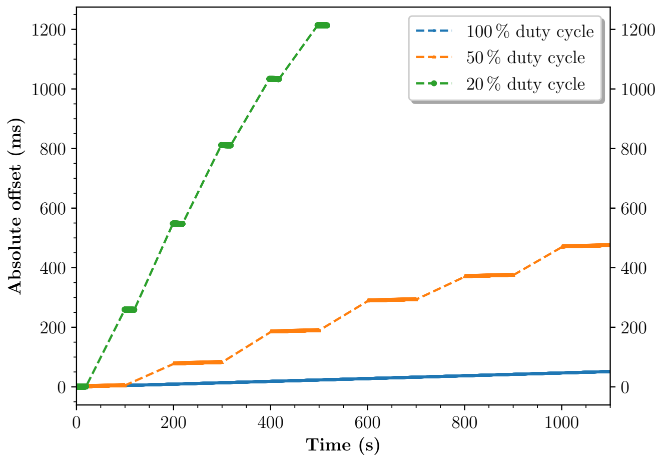

Fig. 1: Accuracy without clock synchronization in duty-cycled nodes. The offset grows fast in the absence of clock synchronization.

putting GPS receiver on every device is not practical. Besides cost, receiving the signal from the satellites requires line of sight visibility which is a limitation for indoor deployments. PTP fills this gap and is gaining wider acceptance. However, achieving the stated accuracy using PTP requires specialized hardware and transparent bridges which would be unreasonable to expect from the low-cost IoT devices and also large-scale deployments, e.g., on factory floors. A more practical approach taken in this work, is to use available high accuracy clock synchronization (e.g., PTP) in a few of the nodes to synchronize their clocks to an external time source, and use a lightweight clock synchronization protocol to synchronize the rest of the clocks with the externally synchronized ones.

This paper presents Event Reference Synchronization (ERS), a clock synchronization protocol that leverages the broadcast nature of wireless transmissions. Though several broadcast-based clock synchronization algorithms have been proposed in the literature (see Section II for a review), the novelty of ERS is that each clock synchronization has the overhead of only one broadcast message and this message can synchronize all the nodes in the transmitter's neighborhood. Further, ERS is asynchronous, that is, the interval between an event and the related clock synchronization broadcast is not constrained. A proof-of-concept implementation of ERS involving a testbed of low-cost (approx. \$7) ESP32 development boards was done. Measurement data show that ERS can achieve 85 μs accuracy on these low-cost devices over an IEEE 802.11 WLAN and using software timestamping. This accuracy is one order of magnitude better than the achievable accuracy of NTP over an IEEE 802.11 WLAN [5]. ERS exceeds the 1 ms accuracy requirement for event sequencing in IIoT applications [5], [6]. Further, the demonstrated accuracy level of ERS meets the requirements of several other industrial applications [7].

The rest of this paper is organized as follows: A discussion of related work is presented in Sec. II. A background on clock synchronization is presented in Sec. III followed by the details of ERS in Sec. IV. A proof-of-concept ERS implementation design, details ad data are presented in Sec. V. Sec. VI presents the conclusion.

II. Related Work

Algorithmic studies of using broadcast for clock synchronization dates back to eighties [8], [9]. Several broadcast-based synchronization protocols have also been proposed in the literature [10]–[14]. Reference Broadcast Synchronization (RBS) [10] was primarily designed for internal clock synchronization in wireless sensor networks, although with overheads it can be extended for external clock synchronization. In RBS some node sends physical layer broadcasts to neighbors. Neighbors record the system clock time when they receive the broadcast which they then exchange with their own neighbors. This allows the nodes to compute the offset and drift rate of their neighbors. This information is then used to convert local event timestamps to the receiver's time at the time of transmitting them to the neighbors. RBS suffers from high message exchange overhead since one clock synchronization entails $O(k)$ message exchanges where k is the number of neighboring nodes. In contrast, one broadcast message synchronizes all the clocks in a neighborhood in ERS. The authors reported the precision of RBS as 29.61 μs for the 99th percentile under negligible traffic load and 48.61 μs for the 99th percentile at a traffic load of 6.5 Mbps. However, a subsequent work reported the precision of RBS between a pair of nodes to be 93 μs [15]. In a theoretical work, Noh et al. build upon RBS where nodes synchronize by listening to the message exchanges among neighbors [11]. Cheng et al. [13] further improved the work of Noh et al. by reducing the message exchanges. Ren et al. developed a mechanism to use periodic beacon broadcasts to emulate a PLL algorithm on sensor nodes [12].

Timing-Sync Protocol for Sensor Networks (TPSN), also developed for wireless sensor networks, uses two-way message exchange discussed in Sec. III. A level discovery procedure arranges clocks in a tree hierarchy where all the clocks are synchronized to a root clock. Using timestamping at Medium Access Control (MAC) layer, TPSN achieved precision of 45.2 μs over 1 hop and 73.6 μs over 5 hops [15]. Flooding Time Synchronization Protocol (FTSP) establishes a tree hierarchy in a wireless sensor network where flooding originating from the root is used to synchronize the nodes [16].

Several clock synchronization protocols based on PTP's `sync` and `follow_up` message sequence (Sec. III) have been reported. Mahmood et al. present clock synchronization approach based on this idea [17]. They implemented the system on off-the-shelf Access Point (AP) and WLAN cards. The AP sends these messages and the stations estimate one-way delay using `delay_req` and `delay_resp` messages. Their system achieved an accuracy of 23.6 μs when using dedicated packets for sending `sync` and `follow_up` and an accuracy of 37.4 ms when using beacons for the same. Reference Broadcast Infrastructure Synchronization (RBIS) is another work that also emulates the PTP's `sync` and `follow_up` message sequence [14] and has been tested on general purpose desktop. RBIS is based on master-slave paradigm and relies on two WiFi APs operating in infrastructure mode, one of which is synchronized using PTP. One AP sends the `sync` message and the other sends `follow_up` message. These messages are locked in frequency to avoid ambiguity and due to this fact, scaling RBIS to large networks with multiple APs seems difficult. Another clock synchronization protocol in this category but for IoT mesh networks is Mesh Time-synchronization Protocol (MTP) [18]. In MTP, the gateway is synchronized to external time reference which then broadcasts the timing to the other nodes in the mesh. Measurement data on a hardware implementation of MTP showed its accuracy to be in 0.5-3.2 ms range for various settings.

Mani et al. present a clock synchronization system, called SPoT, for IoT where a lightweight client synchronizes local clock to a reference server [19]. SPoT was shown to offer an accuracy improvement of one order of magnitude over SNTP [20] and (17 times) and MQ Telemetry Transport, MQTT [21] (22 times). The 15 ms accuracy achieved by SPoT, however, is two orders of magnitude lower than the accuracy of the proposed ERS. A comparison of the clock synchronization systems discussed here is presented in Table I. Extensive list of clock synchronization literature can be found in survey papers [22] and [23].

TABLE I: A comparison of clock synchronization protocols

	Message protocol	**Accuracy**
NTP	two-way	5 - 100 ms, software timestamping
PTP	two-way	10 - 100 μs, software timestamping. ns, hardware timestamping
ERS	broadcast	85 μs, software timestamping
RBS [10]	broadcast + unicast	48.61 μs (93 μs [15]), hardware timestamping
TPSN [15]	two-way	45.2 μs, hardware timestamping
FTSP [16]	flooding	1.48 μs, hardware timestamping
PTP-based [17]	two-way	23.6 μs
RBIS [14]	synchronous broadcast	3.3 μs, software timestamping, implemented on general purpose PC
MTP [18]	two-way	0.5-3.2 ms, hardware timestamping
SPoT [19]	two-way	15 ms, timestamping not specified

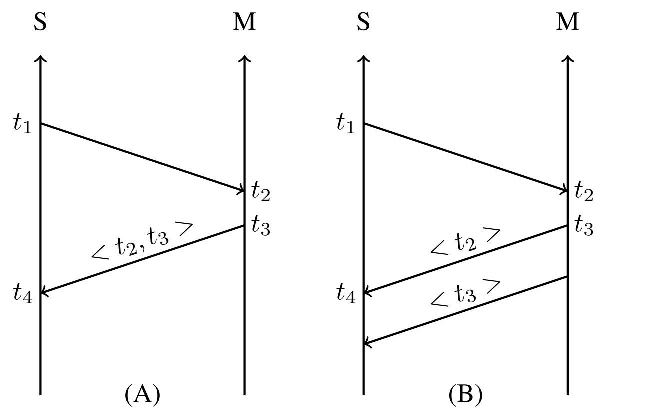

Fig. 2: Synchronization message exchanges

III. Background

If the delay between the sending and receiving a timestamp can be known accurately then clock synchronization is straightforward. Consider synchronizing clock at node S to the clock at node M. Let M send a synchronization message at time t_M for which the receiving timestamp generated at S is t_S. If the delay between the generation of t_M and t_S (according to M's clock) is Δ, then the offset at the receiving clock is $t_M - t_S + \Delta$. This one-way time transfer is used in GPS receivers. Since one-way delay is usually unknown, a commonly used method involves *two-way* message approach where the one-way delay is estimated using message exchanges between master and slave clocks in both directions as shown in Figure 2-A. Let the timestamp of the synchronization request message at the sender be t_1 and that at the receiver be t_2. Further, let the sending and receiving timestamps of the response message be t_3 and t_4. We assume that the offset remains the same during the process. We use δ to denote the offset, and Δ_{req} and Δ_{rsp} to denote the delays incurred by the request and response messages, respectively. The four timestamps are related as follows:

$$t_2 = t_1 + \delta + \Delta_{req} \qquad (1)$$

$$t_4 = t_3 - \delta + \Delta_{rsp}. \qquad (2)$$

Here we get a system of two equations with three unknowns – we can only find out the round-trip delay from these messages but not the one-way delay. Solving for the offset we get:

$$\delta = \frac{t_2 + t_3 - t_1 - t_4}{2} + \frac{\Delta_{rsp} - \Delta_{req}}{2}. \qquad (3)$$

Thus, the accuracy of the estimation of δ depends on the *delay asymmetry* $|\Delta_{rsp} - \Delta_{req}|$. Synchronization protocols, including the one proposed in this paper, try to *minimize* the error in the estimation of δ.

Operating system delays, interrupts, medium access delays, asymmetric routes and queuing delays at switches and routers are main sources of delay asymmetry. The impact of the first three can be eliminated by *hardware timestamping* where recording the timestamp is delayed until the MAC layer passes the message to the physical layer. On the receiving side, the timestamp is also recorded when the message is passed on to the MAC layer. The sending timestamp is then sent in a third message as shown in Fig. 2-B [3], [24].

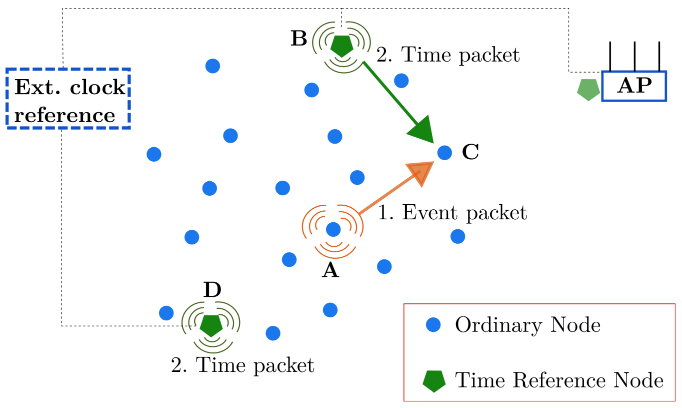

Fig. 3: ERS message sequence at node C. 1. A nearby node (A or the nearby AP) transmits an event packet E. 2. Nodes B and C record the receiving timestamp and the fingerprint identifying E. 3. TRB B broadcasts E's timestamp and fingerprint. 4. Node C computes the offset by comparing the two timestamps.

IV. Event Reference Synchronization

Event Reference Synchronization (ERS) relies on the broadcast nature of wireless transmissions to accomplish clock synchronization. In ERS two kinds of nodes are defined: ordinary nodes and Time Reference Broadcaster (TRB) nodes. TRBs are synchronized to an external time source and clocks in ordinary nodes synchronize to TRB clocks. ERS uses transmissions of data and control packets, which we call *event packets*, as time markers. These event packets are heard by ordinary nodes as well as TRB nodes. Though the payload might be encrypted, headers in an event packet are sent in clear. A header contains sufficient information to uniquely identify a particular event packet. The nodes wishing to synchronize their clocks record the timestamp of event packets and cache the timestamp and the unique fingerprint identifying the associated event packet. When a TRB hears an event packet, it also does the same at first and subsequently broadcasts a packet containing the timestamp and fingerprint pair. These broadcasts are referred to as *time packets*. Ordinary nodes use the cached system timestamp associated with event packet and the corresponding reference timestamp contained in the time packet to perform clock synchronization (Fig. 3).

Suppose the system timestamp associated with a particular event packet E at node i is $t_i(E)$ and the timestamp associated with the same event packet at a TRB is $t_r(E)$. Let Δ denote the delay between receiving the event packet and generation of the local timestamp. Lastly, let d_{si} and d_{sr} denote the distances from the node that generated E to the nodes i and TRB, respectively. Thus, the offset at node i for the event packet E is

$$(t_i(E) - \Delta_i(E) + d_{si}/c) - (t_r(E) - \Delta_r(E) + d_{sr}/c)$$

$$= t_i(E) - t_r(E) + (\Delta_r(E) - \Delta_i(E)) + (d_{si} - d_{sr})/c,$$

where c denotes the speed of light. Using MAC layer timestamping the effect of non-deterministic OS delays can be eliminated, making $\Delta_r(E) \approxeq \Delta_i(E)$. Note that the offset is insensitive to non-deterministic medium access delay. The offset then becomes:

$$\approxeq t_i(E) - t_r(E) + (d_{si} - d_{sr})/c.$$

If the locations of the nodes are known, the contribution $(d_{si} - d_{sr})/c$ due to propagation delay asymmetry can be explicitly factored-in. However, the contribution due to distance asymmetry is negligible in practice due to the limitations on the range of IoT nodes and the extant of network deployment which is typically few tens of meters. Clearly, $(d_{si} - d_{sr})/c$ is upper bounded by $\min(R, D)/c$ where R is the maximum range of an individual node and D is the diameter of the network. Large-scale deployments are multi-hop where R would likely be the limiting factor. For example, in deployments with a maximum range of 30 m, the maximum propagation delay asymmetry is $0.1\,\mu$s, well within the accuracy objective of this protocol. Consequently, the offset can be approximated by $t_i(E) - t_r(E)$ if the distances are not known.

Several TRBs can be in the proximity of a given node. This can be advantageous because a node that gets multiple time packets for the same event can use the more robust average offset. Further, in multi-hop networks an event packet can generate several time reference messages as the original packet gets forwarded through the network. Since the event packets' header signature will change each time the packet is forwarded, the time reference from a previous hop will not match and cause ambiguity. In a prototype implementation using IEEE 802.11 which is discussed below, periodic beacons from the same AP are used to synchronize nodes. Since the MAC header signature changes from beacon to beacon, no ambiguity results.

The previous PTP-like approaches in the literature differ in details but the fundamental principle remains the same – the broadcast message or a synchronous follow-up message, as in PTP, provides the time reference(e.g., [14], [17]). ERS differs from these PTP-like approaches in that the time reference is sent asynchronously. Since an event is used as time marker by both ordinary node and TRBs, the need for one-way delay estimation is eliminated, which would not have been the case where the only broadcasts were TRB's time reference broadcasts. Unlike RBS and its derivatives, ERS provides external clock synchronization natively. Further, the frequency correction approaches used in NTP, PTP, RBS and other systems cited earlier can also be used in ERS for syntonization.

V. Implementation

A. Design

In the proof-of-concept implementation ESPRESSIF System's ESP32-WROOM-32D development boards were used. IEEE 802.11, commonly used in IIoT deployments, was used for networking. Time packets were broadcasted as modified IEEE 802.11 beacon frames which are referred below as *time beacon*s. The frame format of all the IEEE 802.11 data frames has a 30 Byte MAC header and a 4 Byte Frame Check Sequence (FCS) shown in Fig. 4. If the `FromDS` bits in the frame control field is not set, `Addr2` contains the source MAC address. Otherwise, it contains either Basic Service Set Identifier (BSSID) or transmitting AP's MAC address. The frame format of all IEEE 802.11 management frames has a 24 Byte MAC header and a 4 Byte FCS shown in Fig. 5. The fields `DA` and `SA` contain the MAC addresses of the destination and the source, respectively. The `Frame control`, `Addr2` or `SA`, `Sequence control` and `FCS` fields together are used as event packet fingerprint, though further optimization of fingerprint size is possible.

TRB sends this fingerprint of an event packet along-with the associated reference timestamp as an *Information Element* (IE). Depending on the type of management frame, the `Frame Body` contains a set of IEs. The format of an IE is a 1 Byte `Element ID` field, followed by a 1 Byte `Length` field containing the value of the size of the rest of the information element in Bytes, followed by one or more fields defined for the particular information element (Fig. 6). Beacons contain the Service Set Identifier (SSID) IE which has `Element ID` defined to be 0 and the `Length` could be up-to 32 followed by up-to 32 characters long human-readable `SSID` name. SSID name in time beacon was configured to `_TIME_`. We note that the same SSID in different TRBs can be used since its purpose is only to recognize a time packet.

In the IEEE 802.11 standard, a number of information Element IDs are either not used or left for vendors' use. Some of these defined in the standard are 17-31, 128-129, 133-136 and 143-173 [25]. In time packets (beacons), after the SSID IE with the name `_TIME_`, a Time Reference IE is added. This IE contains the event packet fingerprint and reference timestamp (Fig. 7). We assigned 25 as `Element ID`, 14 Byte for fingerprint followed by 8 Byte reference timestamp (in POSIX timestamp format: 4 Byte second field and 4 Byte microsecond field).

Though security is not the focus of the work presented in this paper, it should be noted that the design of ERS admits secure synchronization natively. For secure synchronization, TRB can add another Security IE that contains the 22 Byte data of Time Reference IE encrypted by its private key. This, however, would require the availability of key distribution service. The nodes that require secure synchronization can use the encrypted IE while other nodes can still use the unencrypted Time Reference IE.

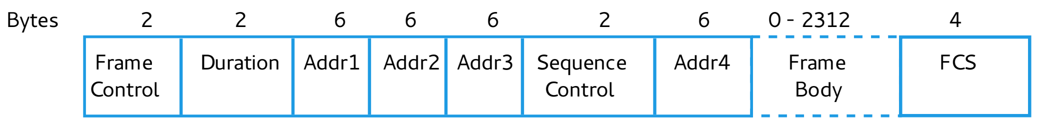

Fig. 4: Generic IEEE 802.11 data frame format.

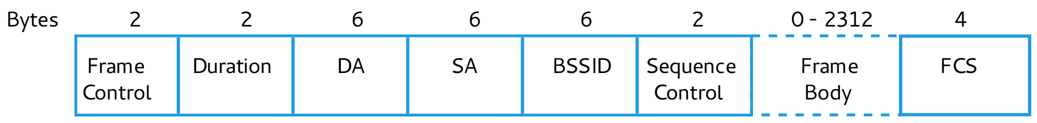

Fig. 5: Generic IEEE 802.11 management frame format.

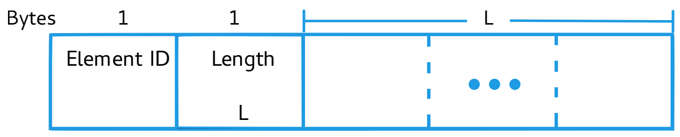

Fig. 6: IEEE 802.11 information element (IE) format.

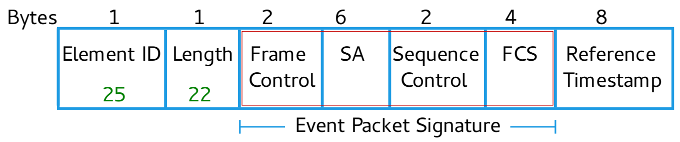

Fig. 7: IE contained in a time beacon.

B. Details

The ESP32-WROOM-32D module has a dual-core 32-bit ESP32-D0WD microcontroller unit, 4 MB external flash memory along-with IEEE 802.11b/g/n, Bluetooth and Bluetooth Low Energy (BLE) modules [26]. It runs on FreeRTOS which is a real-time operating system [27]. Networking is handled by a FreeRTOS task running Lightweight TCP/IP protocol stack. In the default setting, one of the cores handles networking and the other is left for applications. There is support for two low energy sleep modes, called, “light sleep mode” and “deep sleep mode” [28]. In the light sleep mode, CPU, RAM and peripherals are clock-gated, where as in the deep sleep mode, these are not powered at all. In the deep sleep mode, the Real Time Clock (RTC) controller and a low power coprocessor remain functional. However, when the light sleep mode was used, the heap contents were getting corrupted upon waking up.

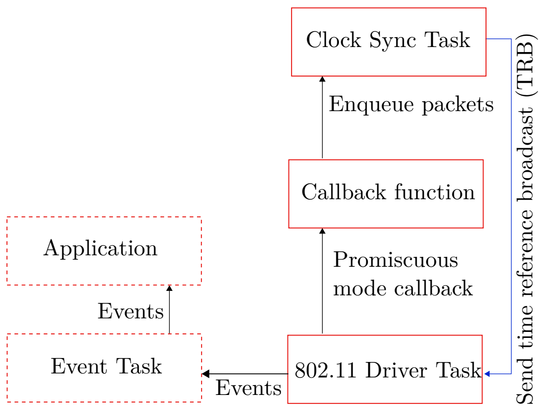

Fig. 8: Block diagram of the implementation.

The block diagram of the implementation is shown in Fig. 8. In the normal mode the 802.11 driver sends events to an event task loop which is then handled by application using callback functions. However, in promiscuous mode, the driver task directly calls the registered callback function for the packets that match the filtering criteria. In our tests, the hardware timestamp on ESP was found to be unreliable. Therefore the timestamp of event messages is recorded in the callback function. In addition to the RTC timer that uses 32.768 kHz oscillator, ESP32 also has a high accuracy timer that derives time from an 80 MHz clock source [28]. Measurement of the frequency stability of both the RTC and the high-resolution timer gave a difference of 7.52 ppm. However, since the high-resolution timer is not available in the sleep modes, RTC was used in the measurements. Upon recording the timestamp, the callback function enqueues the packet to be processed by the *Clock Synchronization Task* (CST). The functionality of CST differs on ordinary nodes and TRBs. In ordinary nodes, CST matches the event packet signature sent in time beacons to that stored at the node. Upon a match, it computes the offset and sets the system clock. In TRBs, CST forms (raw) time beacon frame and calls the 802.11 driver API for its transmission.

C. Measurement Data

The measurement setup consisted of four ESPRESSIF ESP32-WROOM-32 boards. Three ESP32 boards were used as ordinary nodes and one ESP32 board was used as the TRB. The entire TRB functionality was implemented on the ESP32 board. Beacons from APs located in neighboring buildings were used as event packets. A TPLINK Access Point (AP) running on Atheros QCA9533 SoC in our control served as the external time source to the TRB node. The beacon interval of this AP was kept at the default 102.4 ms. The TRB was caching 10 event packets and associated timestamps at a time and then was broacasting the time packets for each of these in an infinite loop. Logs from the ESP32 nodes were sent through UART (Universal Asynchronous Receiver-Transmitter) to a

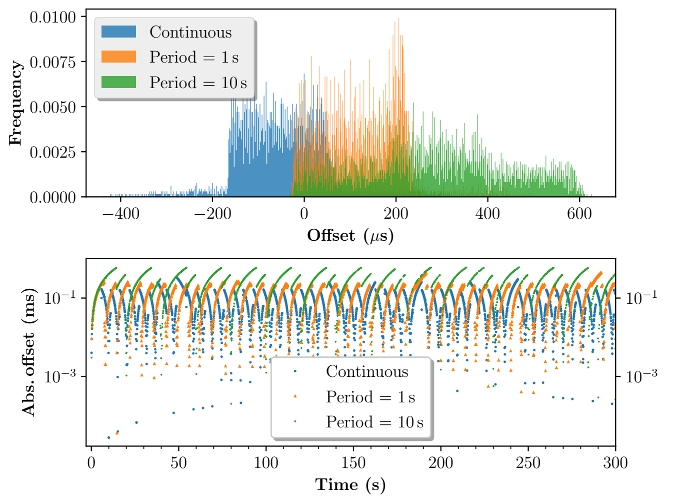

Fig. 9: Accuracy of ERS as a function of synchronization frequency.

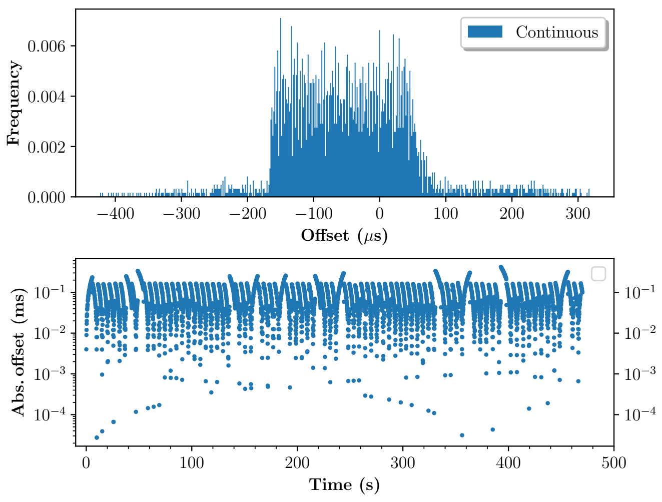

Fig. 10: Relative error between a pair of synchronized nodes.

laptop. All the beacons sent by the TPLINK AP that the nodes received (along-with the 8 Byte beacon timestamp that is sent in the `Frame body` of 802.11 beacon frames) and the associated local timestamp were sent by all the nodes over UART for analysis.

The accuracy of ERS as a function of clock synchronization interval is presented in Fig. 9. Data was collected for continuous synchronization and for synchronization periods 1 s, 10 s and 30 s. The number of samples was 4500 for the continuous synchronization and approx. 5000 for the other cases. Synchronization was triggered using a high priority periodic task and once the offset was adjusted, no further clock synchronization was done until the next trigger. In the case of continuous synchronization, CST initiated next synchronization round as soon as the system clock was set. The average accuracy for continuous synchronization was 84.649 μs and the precision was 67.018 μs. The measurement data are summarized in Table II. The continuous synchronization mode data is skewed towards negative offset due to fast clock rate.

The relative error was measured by comparing the system timestamps on a pair of ordinary nodes. The beacons sent by the TPLINK AP were filtered (in) and the system timestamp offset for the same beacon frames were computed. The data are shown in Fig. 10. The mean error was 110.03 μs (95% CI 107.64-112.43 μs).

These results show that ERS achieves sub-millisecond accuracy with software timestamping, low synchronization frequency and the implementation of the TRB functionality on a low-cost limited-capability device. This accuracy meets the requirements of general IoT applications, causal inferencing of events [29] and monitoring applications in industrial settings [7]. In the following evaluations, the synchronization period was set to 5 s. This setting appears to offer a good balance between accuracy and clock synchronization overhead.

TABLE II: Accuracy vs. Synchronization period

Synchronization interval	***Accuracy***	***95 % CI***
Continuous	84.65 μs	82.67 - 86.61 μs
1 s	125.53 μs	123.17 - 127.90 μs
10 s	286.04 μs	281.47 - 290.61 μs
30 s	590.60 μs	580.49 - 600.90 μs

Accuracy in duty-cycled nodes

One of the main challenges of clock synchronization in IoT is duty-cycling for energy saving since the system time in sleep modes is usually maintained by low frequency crystal oscillators. The performance of ERS on duty-cycled nodes was measured where the duty-cycling settings were kept as 100 %, 50 % (50 s active period followed by 50 s sleep) and 20 % (20 s active period followed by 80 s sleep). As mentioned earlier, the synchronization period was kept at 5 s during the active period. The clocks were free-running during the sleep period.

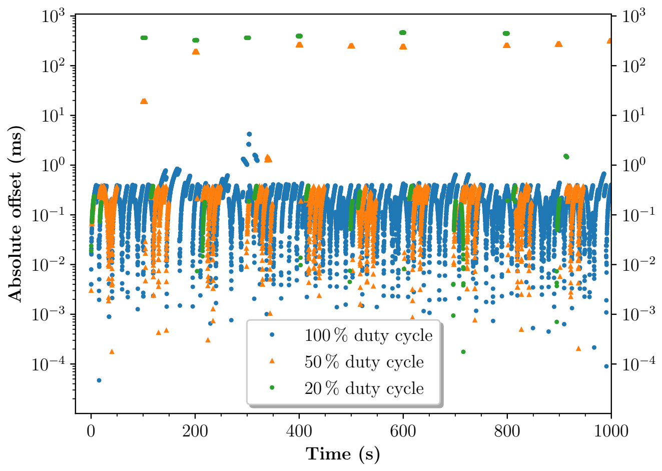

Fig. 11: Accuracy in duty-cycled nodes

Fig. 11 shows the measurement data. Synchronization period of 5 s was sufficient to maintain sub-millisecond accuracy during the active periods. The average accuracy during active period was 217.5 μs (95 % CI 213.1-221.9 μs). During the sleep period, the average accuracy dropped to 462 ms which again improved to sub-millisecond range upon receiving the time beacons during the active period. This post-wakeup but pre-synchronization clock values appear as the outliers (points near the top) in the figure.

VI. Conclusion

ERS provides external clock synchronization for low-cost devices in IoT and IIoT networks. It provides better accuracy than NTP and complements the PTP standard by extending its coverage to capability-limited nodes. In ERS, the event messages that are markers of time, and the time reference messages that contain reference time, *both* are broadcast messages. Therefore, a single time reference broadcast synchronizes all the neighboring nodes that also received the associated event message. This is in contrast with other broadcast-based algorithms discussed earlier, such as RBS and its derivatives, where broadcast is used to eliminate non-deterministic delays but unicast is still used to infer the clock offset. ERS does away with the unicast altogether. Further, being asynchronous ERS can be implemented on low-end devices. Message filtering and various other optimizations on TRB as well as ordinary nodes to further improve the scalability of ERS are interesting future directions.

Acknowledgment

This work was supported by Ahmedabad University grant number URBSEASE20A6/SUG/20-21/03-SP-08.23.

References

[1] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[2] http://www.ntp.org/ntpfaq/NTP-s-algo.htm, Retrieved Jan 2021.

[3] "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–300, 2008.

[4] D. W. Allan and M. A. Weiss, "Accurate time and frequency transfer during common-view of a GPS satellite," in *34th Annual Frequency Control Symposium*. IEEE, 1980, pp. 334–346.

[5] D. Anand, D. Sharma, Y. Li-Baboud, and J. Moyne, "EDA performance and clock synchronization over a wireless network: Analysis, experimentation and application to semiconductor manufacturing." ISPCS 2009 International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, 2009.

[6] G. S. Antonova et al., "Standard profile for use of ieee std 1588-2008 precision time protocol (ptp) in power system applications: Ieee pes psrc working group h7/sub c7 members and guests," in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*, 2012, pp. 1–6.

[7] "IEEE Standard Profile for Use of IEEE 1588 Precision Time Protocol in Power System Applications," *IEEE Std C37.238-2017 (Revision of IEEE Std C37.238-2011)*, pp. 1–42, June 2017.

[8] J. Y. Halpern, B. Simons, R. Strong, and D. Dolev, "Fault-tolerant clock synchronization," in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ser. PODC '84. New York, NY, USA: Association for Computing Machinery, 1984, p. 89102. [Online]. Available: https://doi.org/10.1145/800222.806739

[9] J. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1 – 36, 1988.

[10] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 147163, Dec 2002. [Online]. Available: https://doi.org/10.1145/844128.844143

[11] K. Noh, E. Serpedin, and K. Qaraqe, "A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization," *IEEE Transactions on Wireless Communications*, vol. 7, no. 9, pp. 3318–3322, Sep. 2008.

[12] F. Ren, C. Lin, and F. Liu, "Self-correcting time synchronization using reference broadcast in wireless sensor network," *IEEE Wireless Communications*, vol. 15, no. 4, pp. 79–85, Aug 2008.

[13] K. Cheng, K. Lui, Y. Wu, and V. Tam, "A distributed multihop time synchronization protocol for wireless sensor networks using pairwise broadcast synchronization," *IEEE Transactions on Wireless Communications*, vol. 8, no. 4, pp. 1764–1772, April 2009.

[14] G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Implementation and evaluation of the reference broadcast infrastructure synchronization protocol," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 801–811, June 2015.

[15] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 138149.

[16] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 3949. [Online]. Available: https://doi.org/10.1145/1031495.1031501

[17] A. Mahmood, G. Gaderer, and P. Loschmidt, "Software support for clock synchronization over IEEE 802.11 wireless LAN with open source drivers," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Sep. 2010, pp. 61–66.

[18] T. Beke, E. Dijk, T. Ozcelebi, and R. Verhoeven, "Time synchronization in IoT mesh networks," in *2020 International Symposium on Networks, Computers and Communications (ISNCC)*, Oct 2020, pp. 1–8.

[19] S. Mani, R. Durairajan, P. Barford, and J. Sommers, "An architecture for iot clock synchronization," in *Proceedings of the 8th International Conference on the Internet of Things*, ser. IOT '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3277593.3277606

[20] D. Mills, "Simple network time protocol (SNTP) Version 4 for IPv4, IPv6 and OSI," Internet Engineering Task Force, RFC 4330, Jan 2006.

[21] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT Version 5.0," OASIS Standard, Tech. Rep., 2019.

[22] M. Lvesque and D. Tipper, "A survey of clock synchronization over packet-switched networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2926–2947, Fourthquarter 2016.

[23] A. Mahmood, R. Exel, H. Trsek, and T. Sauter, "Clock synchronization over IEEE 802.11 - A survey of methodologies and protocols," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 907–922, April 2017.

[24] J. C. Eidson, *Measurement, control, and communication using IEEE 1588*. Springer Science & Business Media, 2006.

[25] "IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, March 2012.

[26] "ESP32-WROOM-32D and ESP32-WROOM-32D data sheet," Espressif Systems, Tech. Rep. 2.1, 2021.

[27] "FreeRTOS." [Online]. Available: https://www.freertos.org/

[28] "ESP32 technical reference manual," Espressif Systems, Tech. Rep. Version 4.3, 2020.

[29] D. M. Anand, J. G. Fletcher, Y. Li-Baboud, J. Amelot, and J. Moyne, "Using clock accuracy to guide model synthesis in distributed systems: An application in power grid control," in *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Sep. 2010, pp. 7–12.