PSync: Visible Light-Based Time Synchronization for Internet of Things (IoT)

XiangFa Guo*, Mobashir Mohammad*, Sudipta Saha*, Mun Choon Chan*, Seth Gilbert*, Derek Leong†
*School of Computing, National University of Singapore, Singapore
†Institute for Infocomm Research, Singapore
{xiangfa, mobashir, sudipta, chanmc, seth.gilbert}@comp.nus.edu.sg, dleong@i2r.a-star.edu.sg

***Abstract*—Time synchronization is an enabling service that allows devices to share a consistent notion of time and thus makes it easier to build efficient and robust collaborative services. However, existing synchronization protocols based on wireless packet transmissions are not energy efficient because powering the radio often consumes a significant fraction of the energy budget. In this paper, we propose *PSync*, a *visible light-based* time synchronization protocol that relies on an LED light source and is highly energy efficient for the receivers. The key novelty in our protocol is the use of a De Bruijn sequence to provide a rough estimate of time using a minimum amount of information. Experiments show that our scheme achieves an average synchronization error of 1.3 timer ticks (32 µs per clock tick). In addition, the additional energy consumed for one round of synchronization based on PSync can be as low as 19% of the energy needed to receive a small packet (1 byte) using IEEE 802.15.4 radio.**

I. Introduction

With the emergence of the Internet of Things (IoT), designing highly energy-efficient protocols suitable for use in a localized or indoor environment such as inside a room or vehicle becomes interesting. In many of these usage scenarios, while the end devices may be highly resource constrained, a resource-rich device may be nearby.

In this paper, we focus on the problem of synchronizing resource-constrained IoT devices. Time synchronization is a key enabling service when there is a need for many devices to coordinate their actions. By ensuring that all the devices share a consistent notion of time, it becomes much easier to build efficient and robust collaborative services.

Typically, synchronization protocols for these wireless networks rely on the fact that every wireless device receives a given packet at (approximately) the same instance of time. Thus, if some device sends a synchronization packet, other devices that receive the packet can synchronize on the instant of packet reception. The energy consumed by these synchronization protocols depends on the cost of radio transmission and reception. For resource-constrained devices, powering the radio can consume a significant fraction of the energy budget. Further, in order to execute the synchronization protocol via packet reception, devices often need to keep the radio on for a period much longer than the actual packet transmission and reception duration, leading to higher energy consumption.

In this paper, we propose *PSync*, a *visible light-based* time synchronization protocol. The main idea in our design is that pulses of light produced by light emitting diodes (LEDs) yield a very efficient mechanism for synchronizing nearby devices.

PSync utilizes an asymmetric design and is motivated by the widespread deployment of LEDs. The use of LEDs is expected to grow due to its energy efficiency [1]. These ambient LED light sources have very fast (nanosecond) switching times and are highly programmable. The power efficiency in the receiver arises from the following factors. First, the photodetectors (PDs) or light sensors in the receivers consume very little power. Second, the sampling of light sensors can be done at a high rate. Therefore, it is possible for a device (receiver) to sleep most of the time and only wake up over short intervals to perform synchronization. In order to synchronize, one or more light sources (simultaneously) broadcast a predetermined bit sequence (where ON is 1 and OFF is 0) and devices synchronize on the transition of a specific pulse.

The key challenge of PSync is the design of the bit sequence. In order to be highly energy efficient, a receiver should be awake only for a very short interval and yet be able to quickly determine the synchronization point. In our design, a receiver first performs low-frequency sampling to estimate the approximate time to the actual synchronization point and subsequently wakes up to perform high-frequency sampling just before the synchronization point.

Our solution to designing the bit sequence is to use a De Bruijn sequence [2], which has the following special property: each possible sequence of N bits (there are 2^N such sequences) can be found in a *unique* position within a De Bruijn sequence of 2^N bits. Thus, we may determine our position in such a sequence by reading only N consecutive bits. While De Bruijn sequences have been applied in genomics, pattern recognition [3], our application of this technique for synchronization is novel.

We have fully implemented our synchronization protocol using an off-the-shelf LED light source controlled by TelosB. Our evaluation shows that PSync can achieve an average synchronization error of 1.3 timer ticks (32 µs per clock tick). Using a light pulse width of 1 ms, one synchronization cycle of PSync consumes around 57 µJ. In comparison, the reception

This work was supported in part by the Agency for Science, Technology and Research (A*STAR), Singapore, under SERC Grant 1224104049.

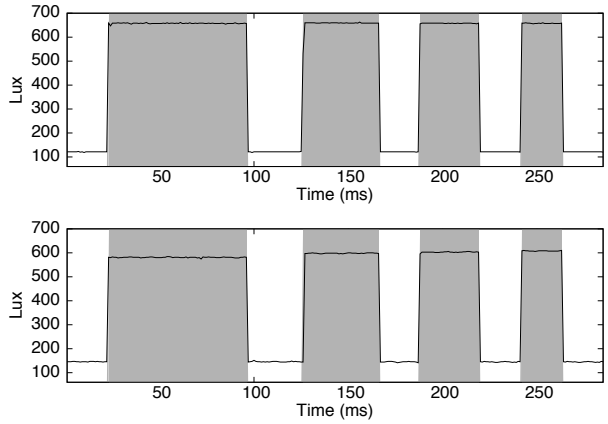


Fig. 1. Light sensor readings from 2 TelosB motes detecting the same light source.

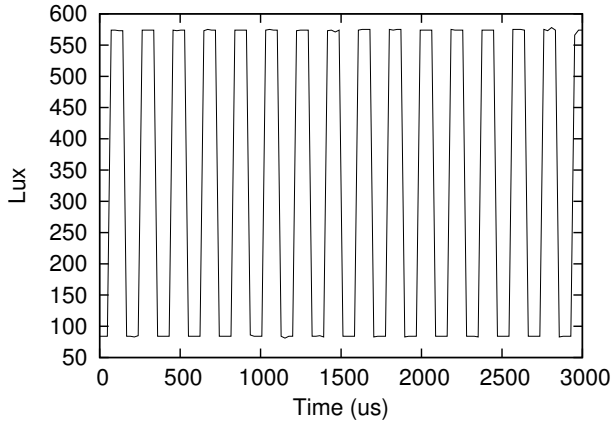


Fig. 2. Light intensity detected by a TelosB mote with blinking rate of 5 kHz.

of a small (1 byte) packet over the IEEE 802.15.4 radio consumes 171 µJ. The energy consumption of PSync can be further reduced by using a smaller pulse width. With a 0.5 ms pulse width, synchronization power consumption reduces to 32 µJ.

This paper is organized as follows. In Section II, we present related work. We motivate the possibility and benefit of light-based synchronization in Section III and present the protocol details in Sections IV through VI. Experiments are presented in Section VII, discussions in Section VIII, and we conclude in Section IX.

II. Related Work

Visible light based communication (VLC): Visible light communication over a short range has been standardized as IEEE 802.15.7. The standard supports data rates of up to 96 Mb/s. Schmid *et al.* [4] demonstrated a prototype system for using low power LEDs that supports low data rate (less than 1 Kbps) over a distance of up to 2 m. Visible light has also been used for localization. In *Epsilon* [5], a receiver employs a light sensor to retrieve the beacon information which includes measured RSS (received signal strength) values. The RSS values are later used to compute the distances to multiple bulbs.

Time synchronization and calibration: For devices with reliable Internet connections and no resource constraint, time synchronization is well understood. It can, for example, be provided by a service such as the network time protocol (NTP) [6]. On resource-constrained wireless networks, a variety of synchronization protocols, such as RBS [7], TPSN [8], FTSP [9], and Glossy [10] have been proposed. These protocols use RF communications among the nodes and the time of packet reception as a reference. Mostly, they can achieve fast synchronization and accuracy on the order of 10 µs or less. However, the nodes need to keep their radios always on during the synchronization process, which can consume significant energy.

A related issue is clock calibration. Calibration is used to compensate for clock drift after synchronization is done and can be used to reduce errors between clock synchronizations, hence reducing the need for more frequent synchronization. The proposed techniques mostly try to use periodic background signals to perform calibration. For example, Li *et al.* [11] use the periodicity of the pulse from FM radio broadcast to calibrate the clocks in different nodes. FLIGHT [12] uses the periodic change in the intensity of the light emitted from indoor fluorescent lamps to achieve the same. Note that even after calibration, more work needs to be done to achieve synchronization. In FLIGHT, protocols such as FTSP is still needed to provide the initial time reference.

Preamble sampling based MAC protocols: In wireless sensor networks, MAC protocols that employ preamble sampling have been proposed, e.g., B-MAC [13], X-MAC [14], CMAC [15], and other related asynchronous MAC protocols. In these protocols, the preambles either just indicate activities (e.g., channel is busy) or MAC addresses; they are not explicitly designed for fine-grained synchronization.

Our contribution: Our proposed protocol PSync is novel in that visible light is used for synchronization rather than communication. The technique is based on a De Bruijn sequence which enables a receiver to quickly obtain a rough estimate of the synchronization point by reading only a small number of bits.

III. Motivation and Basic Measurement Results

In order to test the feasibility of visible light-based time synchronization, we conducted the experiments described below.

For a radio-based synchronization protocol [10], the basic assumption is that the SFD pins of the radio fall exactly at the same time for all the devices on receiving the synchronization packet. This provides all the devices a common synchronous event. For visible light-based time synchronization to be feasible, we need an analogous synchronous event observable by all the devices simultaneously. The rise and fall time for an LED is on the order of nanoseconds [1], and hence, LEDs are ideal candidates — as long as the rise and fall observed by the sensors is sharp.

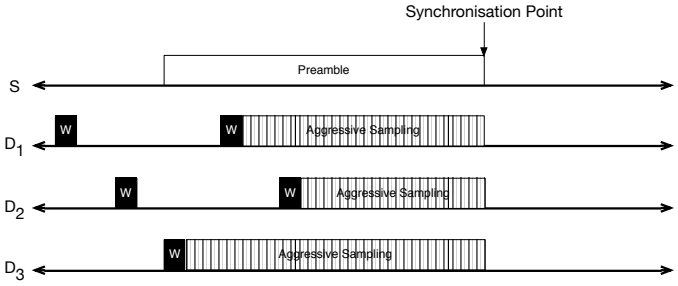


Fig. 3. A naive preamble allowing duty cycling devices to accurately synchronize at the falling edge.

TABLE I
NOTATION

Symbol	Definition
τ	time for transmitting one bit in the preamble
2^ν	length of the De Bruijn sequence
ν	length of the unique subsequence
κ	number of bits for verification

To validate this observation, we deployed two TelosB devices and programmed them to sample light at a rate of 1 kHz, with a light source blinking at different rates. From the results shown in Fig. 1, we can see that the edges are very sharp and the devices can clearly detect these rise and fall events with a synchronization error bounded by the sampling rate.

The next question is whether we can easily program the transmitter to blink at a very high rate and whether the receiver can detect the changes in light intensity. Fig. 2 shows the intensity detected by a light sensor for a blinking rate of 5 kHz. The result shows that despite being a fairly resource-constrained device, the TelosB can generate and receive a 5 kHz light pulse clearly. In fact, we were able to achieve such transmissions and receptions without artifact for up to 10 kHz with the TelosB devices.

IV. Design of Preamble Sequence

This section presents the core mechanism of our visible light-based time synchronization protocol. For a device to be synchronized, the minimal requirement is to have a light sensor. The other basic requirement is the presence of a programmable source of light that can initiate the process of synchronization. The basic idea of our synchronization protocol is to trigger the LED to emit light, and then use the falling edge of the light signal to synchronize all the devices in its luminance range.

A. Preamble

In wireless communications, a preamble is typically used to synchronize the transmission timing between two or more devices. Our approach is inspired by extremely simple and successful asynchronous LPL-based MAC protocols like B-MAC [16], Wise-MAC [17] and X-MAC [14] which make use of preambles to synchronize packet reception.

1) Naive Preamble: We begin with a simple strategy. The transmitter (e.g., the LED in the house lighting system) emits a preamble of duration T, where T is slightly longer than the wake-up interval of the duty cycling devices. Devices with light sensors periodically wake up and sense. On detecting an ongoing preamble transmission, they aggressively sample the light sensor until they observe the falling edge of the light signal, which is used as the synchronization point. This simple strategy depicted in Fig. 3 is obviously not energy efficient for the resource-constrained devices being synchronized.

2) Slotted Preamble with Data Encoding: A better strategy, which allows greater duty cycling among devices, uses a slotted preamble with data encoded into each slot indicating the number of slots remaining until the synchronization point (i.e., the falling edge). One such simple binary encoding could use tuples **00** and **01** to represent 0 and 1 respectively, and use **11** as the delimiter between slots. In this way, a receiver can unambiguously detect a complete slot in the preamble.

In this scenario, "*6 slots to go*" can be encoded into the seventh slot from the end as **(01)(01)(00)**. Each slot can be separated using a special **11** delimiter. With each slot length considered fixed, the devices can sample a slot and quickly go to sleep until the synchronization point.

This slotted preamble strategy with data encoding is illustrated in Fig. 4. Even though this strategy leads to more duty cycling, it also leads to a significant increase in the preamble length.

3) Energy Efficient Preamble: The challenge lies in smartly designing a preamble that saves energy for the light emitters and, more importantly, for the light receivers. Moreover, the scheme should be robust enough to prevent bad synchronization due to random fluctuations in the light signal caused by background noise or physical obstructions. An energy-efficient preamble strategy should meet the following requirements:

1) The preamble should be short and be able to inform the receivers about the synchronization point well in advance. This eliminates the prolonged continuous aggressive sampling at the receivers.
2) The receivers should sample at a low rate, and for a very short duration, during the preamble. At the same time, they should be able to quickly and accurately infer the synchronization point and go back to sleep until the synchronization point.
3) The preamble should have error-correcting properties that allow the devices to detect false positives and false negatives.

Our efficient preamble makes use of a De Bruijn sequence. We first explain the basics of the sequence and then briefly discuss an efficient algorithm to generate and decode the sequence. Later we discuss how our scheme exploits this preamble to achieve precise synchronization on the order of microseconds.

B. De Bruijn Sequences

De Bruijn sequences are periodic sequences in which every possible ν-tuple over a finite alphabet appears exactly once in a given period. Computing the position of a particular ν-tuple

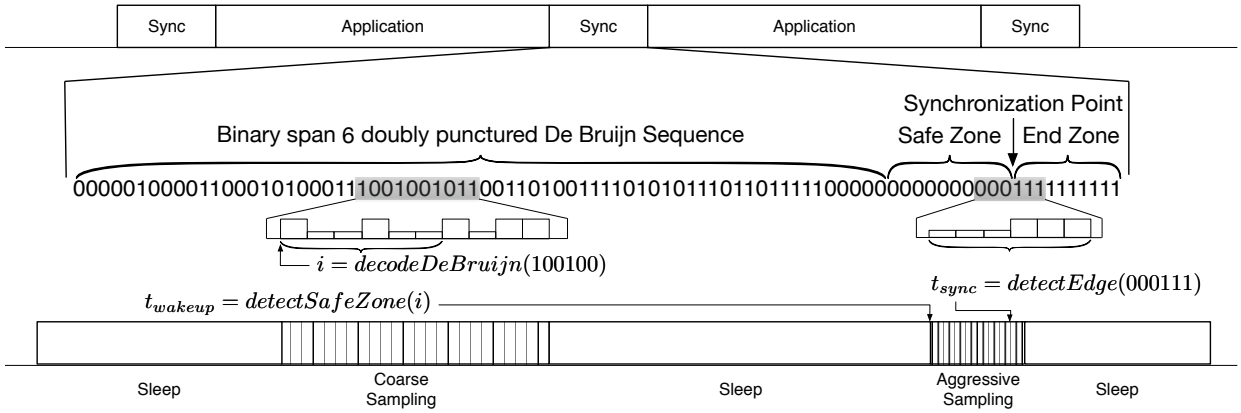


Fig. 5. A doubly punctured De Bruijn sequence emitted as preamble. All the duty-cycling devices sample the ambiance and get synchronized on the peak while transiting from the safe zone to the end zone.

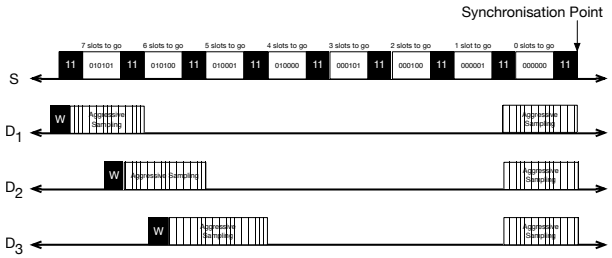


Fig. 4. A slotted preamble with encoded data allowing higher duty cycling.

in the De Bruijn sequence is known as the De Bruijn decoding problem and is often used in position sensing applications. In our application, we use the binary alphabet $\{0, 1\}$.

To illustrate how a De Bruijn sequence can be exploited, consider the sequence 00010111. Any 3-bit subsequence of this sequence is unique and can be used to determine the position of the subsequence. For example, the starting position is indicated by 000, followed by 001, 010, 101, 011, and finally 111. Hence, in our synchronization protocol, if a light source transmits the sequence (turning the LED on and off), a receiver can simply determine when the end of the sequence will be after reading 3 bits.

For sequence generation and decoding, we implement the algorithms described in Mitchell *et al.* [2], which recursively constructs special De Bruijn sequences that can be decoded in constant time with minimal memory overhead.

In our application, the sequence is generated offline. With a pulse or a bit duration of 1 ms, alphabet size $k = 2$, and $\nu = 10$, the light source needs to store a 1024-bit pattern for transmission over a cycle length of 1.024 seconds.

To perform decoding of the constructed De Bruijn sequence, the algorithm needs a small look-up table E which maps all the k-ary ν-tuples to their corresponding positions in the sequence. Due to the lack of space, we will not detail the decoding scheme here. It suffices to highlight that the algorithm decodes in constant time and the space required is small.

1) Efficiency of the De Bruijn Sequence: We compare the efficiency of the simple data encoding approach of Section IV-A2 with the use of a De Bruijn sequence. Assume that the pulse duration is 1 ms and the receiver would like to wake up only once every 1 s, resulting in 1000 slots. Using a De Bruijn sequence, a receiver can estimate time to synchronization after reading 10 slots.

Using the patterns '00' and '01' for the bits '0' and '1', representation of 1,000 slots needs 10 patterns, i.e, 20 bits at least. Along with the pattern '11' as the delimiter, a node needs to read in the best case only 22 bits, or in worst case (19+22) bits, when the reading just missed the first bit of the first pattern after the delimiter '11'. So, on average, it will be (22+19+22) / 2, i.e., about 32 bits. A De Bruijn based solution needs only 10 bits. Thus, the improvement is (100*(32-10)/32)% = 68%.

Another advantage of the De Bruijn sequence is that it is easy to perform error correction, since for every extra bit read, the position indicated by the most recent subsequence must be the next position after decoding. Hence, a very simple error detection scheme can be efficiently implemented by reading k additional bits and then checking whether the decoded positions are correct.

V. Protocol

In this section, we describe the synchronization protocol in more detail. We separately describe the protocol for the initiators flashing the light, and the receivers being synchronized.

A. Initiators

The synchronizing light sources emit a specific doubly-punctured binary De Bruijn sequence of span ν and having a period of $2^\nu - 2$. Each of the ones in the sequence corresponds to a given time τ for which the LED is ON while each zero corresponds to an OFF state of equal duration. The total duration of the De Bruijn preamble is $T = \tau \times (2^\nu - 2)$. This is followed by a "safe zone" of a **0** pulse, and finally an "end zone" of a **1** pulse with the transition edge being the synchronization point. The sequence used is chosen so that two

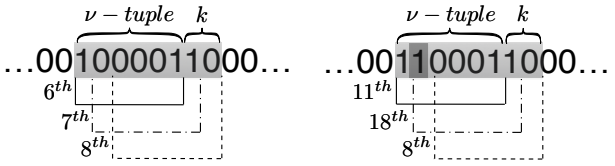


Fig. 6. Simple error detection algorithm using k additional samples.

specific strings are eliminated. These two strings are all zeros and all ones, as these are natural conditions when the light is always OFF and always ON respectively. Removing these two states reduces the chance that a device draws a wrong conclusion during normal changes in the light conditions.

B. Receivers

The receivers wake up periodically, at least once in every interval whose length depends on the duration of the preamble. When awake, a receiver collects s samples for each of the ν symbols of the emitted binary De Bruijn sequence of span ν. The $s \times \nu$ samples are collected at a low sampling rate, and they coarsely represents a unique ν-tuple. A device decodes its position in the preamble to infer unambiguously when the preamble will end.

After decoding, the device sleeps for the rest of the preamble and wakes up during the "safe-zone," at which point it performs aggressive sampling at the maximum allowable rate to detect the synchronization point accurately.

C. Preamble Detection

Detection of a ν-tuple is error-prone in a noisy environment. A single bit error in the preamble detection would cause a device to wake up at an incorrect interval and synchronize to a wrong edge. For example, detecting **100000** as **100001** would make the decoder return 6 instead of 62 for the De Bruijn sequence illustrated in Fig. 5. In this case, the device will wake up while the preamble is still being transmitted instead of in the safe zone.

In order to improve reliability, we adopt a lightweight error detection mechanism to check the correctness of the sensed samples. For a binary De Bruijn sequence of span ν, instead of collecting samples for ν symbols, we collect for $\nu+k$ symbols, where k corresponds to the error detection bits. These samples for k additional symbols give us a total of k additional tuples of the De Bruijn sequence. If correctly sampled, the decoding of each of the $(\nu + k)$-tuples in the $\nu + k$ samples should be consecutive as shown in Fig. 6 and we can correctly calculate the time to wake up. The value of k can be dynamically adjusted depending on how noisy the environment is.

A second issue is that the sleep interval and the length of safe-zone have to take into account the maximum clock drift during the preamble interval. For example, if the maximum clock drift is 20 ppm and preamble length is 10 s, the maximum clock drift is 200 µs either way. Thus, 400 µs of delay is added, 200 µs to the sleep interval and 200 µs for safe-zone.

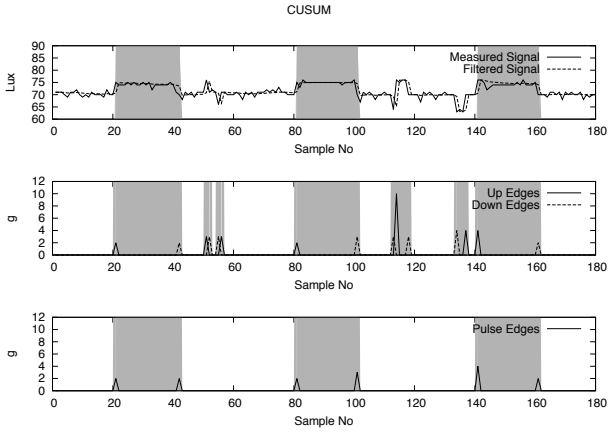


Fig. 7. Edge detection algorithm.

D. Synchronization Point Detection

In our protocol, at the synchronization point, the devices sample at the maximum allowed rate to minimize the synchronization error. Sampling at high rates causes data explosion, making data storage and offline processing infeasible. For real-time synchronization in a noisy environment, in order to handle this inflow of data, we adopt the Online Change Point Detection Algorithm (CUSUM) [18] with an Exponentially Weighted Moving Average filter which we call rlsCUSUM.

For the edge detection algorithm, there is a trade-off between MTD (Mean Time to Detection) and FAR (False Alarm Rate). Since our objective here is to minimize the synchronization error between devices, we allow higher FAR to reduce MTD.

In order to ensure that false alarms are kept low, we implement another layer of false alarm filtering on top of rlsCUSUM to increase synchronization accuracy further, which we call rlsCUSUM$'$. In our protocol, we generate a pulse at the end zone of duration τ_{ez}. With CUSUM in operation for decreased MTD, we get a number of false upward and downward edges due to noises. We mark all those upward and downward edges as invalid if they fail to satisfy the valid pulse width criteria $t_{\text{upwardEdge}} - t_{\text{downwardEdge}} \approx \tau_{ez}$. Fig. 7 compares rlsCUSUM (middle) and rlsCUSUM$'$ (bottom) for samples captured by a light sensor in a noisy environment (top).

VI. Implementation

There are two major components in our system: the synchronizers and the devices being synchronized. The synchronizing devices are light emitters. To validate the feasibility, we have tested different light sources such as simple LED on TelosB, LED lamps and the LED flash on smart phones. In principle, any programmable light source can act as a synchronizer. However, due to interaction with the operating system, high blinking rate with short pulses can only be implemented on devices with fine grained control over timing. Hence, we implemented and evaluated the light-based synchronization protocol in the Contiki OS [19] and tested on TelosB platform

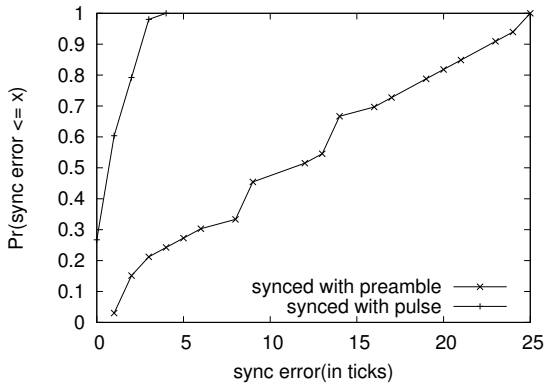


Fig. 8. Cumulative sync error using TelosB controlled LED as the light source. Pulse width $T = 1$ ms and $k = 10$.

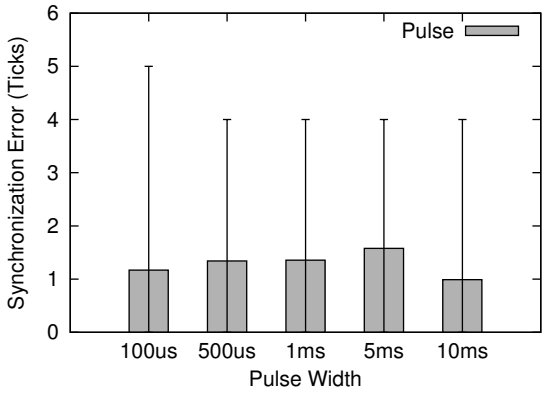


Fig. 9. Impact of pulse width, when synced with pulse.

with an on-board sensor suite including light, temperature and humidity sensors.

The protocol uses two levels of light sensor sampling, as described earlier. There is a coarse-grained synchronization phase using the preamble and a fine-grained synchronization phase using the final synchronization pulse. Since synchronization accuracy depends on the sampling rate, we perform sampling at the fastest rate. Light sensors on the TelosB platform have a rise time of $\approx 0.5\,\mu$s with maximum observed sampling interval being $\approx 7\,\mu$s. Thus it takes $\approx \tau_{\text{sample}} = 8\,\mu$s to turn on, collect an individual sample and go to sleep.

Considering each pulse of duration τ, with $2^\nu - 2$ pulses in the De Bruijn sequence followed by a safe zone and an end zone of τ duration each, the sync phase lasts for $2^\nu \times \tau$ time period.

A. Background Light Intensity Estimation

Before each synchronization phase, we need to obtain an estimate of the background light intensity level. This is required to differentiate the ON and OFF pulses during the preamble sampling phase and also to set the parameters of the edge detection algorithm. This estimation process is performed for a duration of τ_{noise} (typically a few microseconds) to collect a few samples. The duration is negligible in comparison to the duration of the remaining two sync phases.

B. Preamble Detection

The preamble detection phase involves coarse-grained sampling to collect ν-tuple with an additional k error detection bits from the entire preamble. If sampled correctly, the devices can decode the correct position of the collected samples in the preamble and estimate time to the synchronization point.

During this phase, the devices collect s samples for every $\nu + k$ pulses of the preamble, thus needing the sensors to be on for $(s(\nu+k) \times \tau_{\text{sample}})$. With $s = 1$ and $\tau_{\text{sample}} = 8\,\mu$s in our implementation, sensors remain on in this phase for $\tau_{\text{coarse}} \approx 8(\nu + k)\,\mu$s.

At the end of this stage, a node is already approximately synchronized. As the device can sample at any point in the light pulse, the granularity of the timing is on the order of the width of the pulse (e.g., 1 ms) plus clock drift to the common synchronization point. Depending on the synchronization accuracy required by the application, the protocol may stop here if such accuracy is sufficient, which we call "sync with preamble". Otherwise, it proceeds to the next phase to complete the synchronization using the common pulse transition, which we call "sync with pulse".

C. Edge Detection

In the final stage, devices wake up in the safe zone and perform sampling aggressively to search for the final synchronization point. Using the edge detection algorithm with a signal smoothing filter, a node detects the rising and the falling edge of the synchronization pulse.

During this phase, in the worst case, a device turns on its light sensor at the beginning of the safe zone and performs aggressive sensing until the end of the synchronization phase. Hence, this aggressive sampling duration of this phase only needs to be slightly larger than τ.

D. Duty Cycle of the Light Sensors

The duty cycle of the devices undergoing synchronization can be approximated as follows:

$$\textbf{Duty cycle} = \frac{\tau_{\text{noise}} + \tau_{\text{coarse}} + \tau_{\text{aggressive}}}{\tau_{\text{sync}}} = \frac{\tau_{\text{noise}} + 8(\nu + k) + 2\tau}{2^\nu \tau} \approx 2^{(1-\nu)}.$$

Therefore, duty cycling decreases exponentially with length of the tuple ν and extremely low duty cycle is feasible with relatively short tuple. If $\nu = 10$, duty cycle is 0.2%. A duty cycle of 10^{-6} can be achieved with $\nu = 21$.

VII. Experimental Evaluation

In this section, we will evaluate the performance of our light based synchronization. First we present the experimental setup followed by evaluations on the impact of pulse width and the distance between a light source and receivers on the synchronization accuracy. Next, we perform power measurements to evaluate the energy efficiency of our protocol. Finally, we

present experimental results when multiple LEDs send light (with some delays) to a single receiver.

A. Experiment Setup

In our experiments, we use a 3 W off-the-shelf LED. To have finer control of the light source's timing, we constructed a simple multi-LED light source controlled by a single TelosB as shown in Fig. 14a. With this setup, the LED can be strictly controlled to provide light pulses on a time scale of 100 µs. In the evaluation, unless otherwise stated, we set the pulse width T to be 1 ms, ν to be 10 and the synchronization cycle lasts for about 1 s.

The receivers in our experiments are TelosB motes running Contiki. The default built-in light sensor is used. After the synchronization is completed, a packet transmission is initiated to the receiver(s). We measure the accuracy of time synchronization by comparing the clock obtained by our synchronization protocol with the clock obtained using a Glossy-like synchronization protocol that reads the local clock immediately after the packet reception. Time is measured in units of clock ticks where one clock tick is approximately 32 µs.

B. Synchronization Accuracy

We measure the performance of two different approaches: the full protocol that synchronizes all nodes on a single transition, and the approximate scheme where devices synchronize using only the preamble. In this experiment, the light source and the receiver are kept close enough so that the light pulse can be detected with minimum error.

In Fig. 8, the cumulative error distributions of both the schemes using a TelosB/LED as the light source are shown. We can see that the full PSync scheme is able to achieve average error of 1.3 clock ticks, with a maximum error around 4 clock ticks. For the approximate scheme, since sampling is done at a low rate, the error is much larger. The average error is 11 clock ticks and the maximum error reaches the duration of one pulse width of 1 ms or 32 clock ticks.

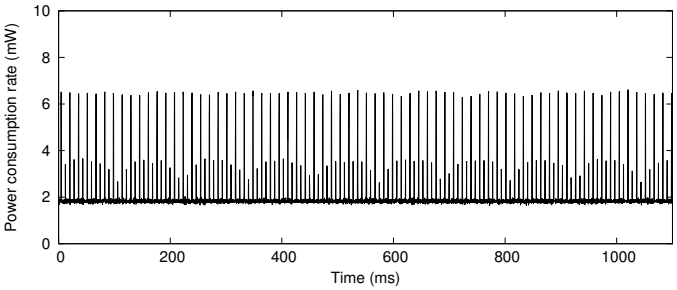
For the scheme that attempts to synchronize using only the preamble, the inaccuracy is mainly caused by different sampling positions in a bit, i.e., where the two motes sense different time points of one bit in the preamble.

We also evaluate the impact of pulse width on the synchronization accuracy by varying the pulse width from 100 µs to 10 ms. As shown in Fig. 9, since the synchronization point is based on transition, as expected, pulse width has fairly minimum impact on accuracy.

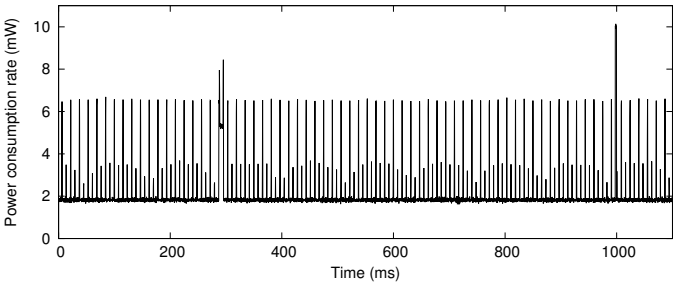
C. Power Measurement

We measure the power consumption of running PSync in TelosB motes, the platform we used to implement our algorithm. Power measurements are performed using the Monsoon power meter.

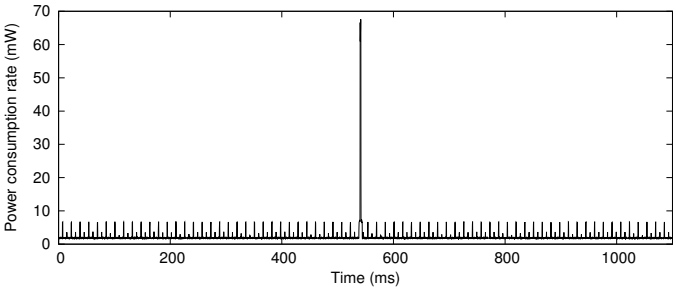
PSync has two main stages that consume power, the coarse sampling period and the aggressive sampling period. We measure the power consumption of the two stages and compare



(a) Idle states power consumption.



(b) Light sync power consumption.



(c) Radio sync power consumption.

Fig. 10. Power profiles for different operations: (a) CPU idle, (b) running PSync every second, (c) reception of a 1-byte packet over IEEE 802.15.4 radio.

with the power consumed by radio based synchronization protocol. As a baseline for radio based synchronization, we consider the power consumed by receiving a small 1-byte packet using the IEEE 802.15.4 radio at 2.4 GHz on the TelosB. The sender is programmed to transmit one small packet per second. The results are shown in Fig. 10.

Fig. 10a shows the power profile when the CPU is idle. The processor wakes up periodically triggered by timer events. The power consumption rate varies from 1.73 mW to 6.52 mW, with an average of 2.01 mW.

Fig. 10b shows the power profile when PSync is run using a pulse width of 1 ms. The power consumption varies from 2.00 mW to 8.00 mW and the additional energy consumed on performing one cycle of PSync is 57.49 µJ.

Fig. 10c shows the power profile when a single 1-byte packet is received. The power consumption varies from 2 mW to 70 mW and the additional energy consumed is 171.61 µJ.

As the processing is relatively simple, the major power consumption in PSync is actually the energy consumed to turn the light sensor ON for the duration of the coarse sampling

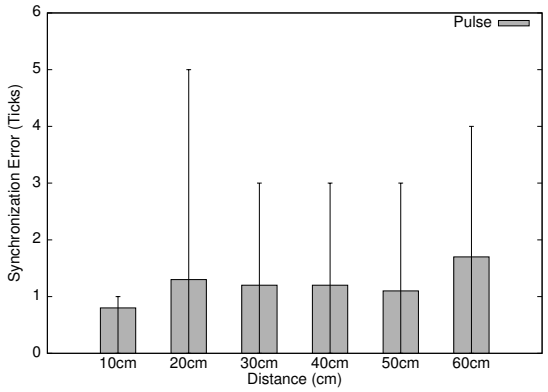


Fig. 11. Synchronization error of PSync when varying distances.

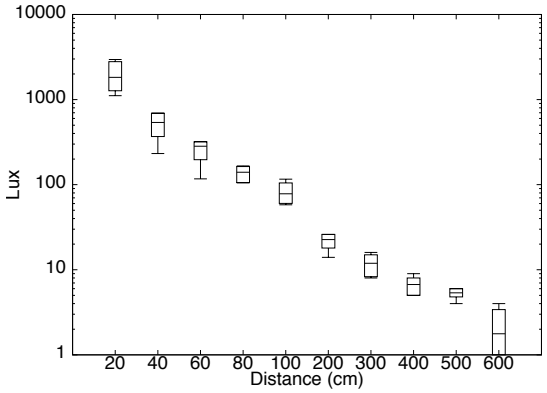


Fig. 12. Light Intensity vs. distance using a 9 W LED and light sensor on Galaxy S II.

period. Further, once a light sensor on the TelosB is turned on, it stays on for at least 2 ms to 3 ms. Nevertheless, the sampling duration can be shortened if a shorter pulse width is used. With a smaller pulse width of 0.5 ms and 0.1 ms, the additional energy consumed to perform one cycle of PSync reduces to 32.43 µJ and 20.42 µJ respectively.

D. Varying Distances

In this evaluation, we investigate how the accuracy varies with distance between the light source and the devices to be synchronized. Using the 3 W off-the-shelf power by battery and controlled by TelosB, the distance up to which one can safely synchronize is observed to be 60 cm as illustrated in Fig. 11. While the error does increase slightly as the distance increases, the average error is still less than 2 ticks, even at the maximum distance evaluated.

In order to increase the distance between sender and receiver, one can either increase the intensity of the light source or the sensitivity of the light sensor. In the next experiment, we measure the light intensity between a stronger light source (an off-the-shelf 9 W LED light bulb) and a more sensitive light sensor (the light sensor on the Galaxy S II). Fig. 12 shows the corresponding results. It can be seen that the sensing range has increased to more than 5 m. The use of an even brighter source or a more sensitive light sensor will increase the sensing range further.

E. Multiple Light Source

One potential challenge for application of light based synchronization is that a sender might not be able to correctly receive a signal from multiple light generators. Multi-hop synchronization might also be needed when devices are spread over a larger area.

We investigate the sub-problem whereby a receiver can detect light from 2 sources. Problem happens when these 2 sources are not well synchronized, leading to the transmitted sequences going out of sync. This can cause difficulty for the receiver to decode the sequence. A similar challenge exists in a radio based protocol, which can be mitigated by exploiting constructive wireless interference and capture effect.

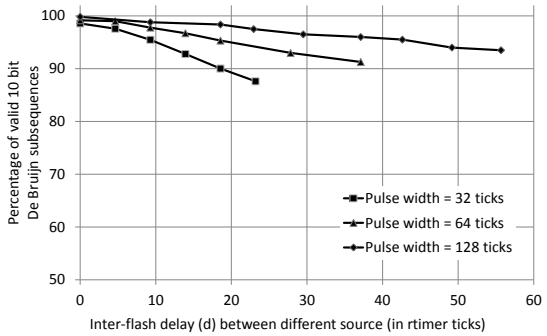


Fig. 13. Plot showing the percentage of valid subsequences of a De Bruijn sequence decoded by a receiver with two concurrent light sources flashing with varying delays.

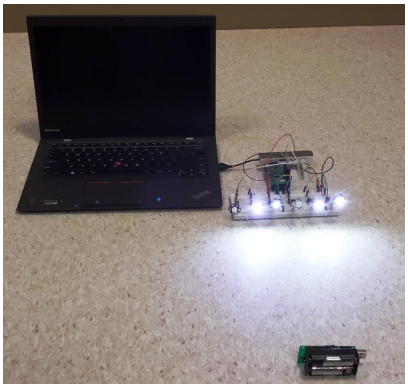
In this experiment, we look at the impact of slight offset between preamble generated by two light sources on its successful decoding. The experimental setup is shown in Fig. 14b. The sender has 1 TelosB controlling 6 LEDs through a driver circuit. In the experiment, 2 randomly selected LEDs are used. We vary the offset between the LED transmissions and check the reception reliability by calculating how likely the receiver can decode the preamble represented by the ratio of the number of correctly decoded 10-bit subsequences out of the 1022 possible subsequences. The result shown in Fig. 13 suggests that in the worst case, 85% of the 10-bit subsequence can be correctly received.

Recall that for correct decoding, all we need is for the receiver to be able to correctly decode just $\nu + k$ bits out of the 2^ν bits of the De Bruijn sequence. If an error is detected, the receiver simply moves on to later part of the preamble and restart decoding. Based on the result in Fig. 13, we can see that as long as the delay between the light sources is a small fraction of the pulse width, it is very likely that a receiver can correctly decode the $\nu + k$ sub-sequences.

VIII. Discussion

So far, PSync has mainly been evaluated in scenarios where the receiver can sense a light directly from the light source. In many deployments, a typical LED light bulb would be

(a)



(b)

Fig. 14. Experimental setup.

sufficient to synchronize devices within an area, e.g., within a room. As time synchronization in an IoT context is for a cluster of devices within proximity, one can scale up the coverage through a central control. All LED lights within all rooms in a house or a single floor can be controlled centrally through a single controller. In this way, synchronization would be similar to the multiple light sources scenario we have evaluated. One interesting behavior of light-based synchronization is that there is no destructive interference as is the case of packet transmission. Instead, the last transition observed will be considered as the synchronization point.

Another option to extend the coverage of PSync is to execute the protocol in a multiple-hop manner with multiple light sources. Note that only a subset of the devices need to serve as light sources. However, all light sources must be within the visible light range of at least one other light source and the union of their ranges cover the entire area of interest. Starting from a single light source, clock information is propagated outwards. Two additional functions are needed to be incorporated into PSync. First, the next cycle of synchronization starts after a fix duration after the end of the previous round's synchronization point. Second, addition clock information needs to be added into the bit sequence so that the offset from the source can be known. Naturally, the error will grow as the hop count increases.

IX. Conclusion

We have presented the design and evaluation of the visible-light based synchronization protocol PSync. Experiments show that the protocol can achieve good accuracy while consuming very little power. As PSync requires only LED and light sensors, it can be easily integrated into most IoT devices, including devices that are extremely resource-constrained or have no radio onboard. PSync provides a viable alternative solution for systems that require synchronization with minimum overhead using existing lighting infrastructure.

References

[1] S. Rajagopal, R. D. Roberts, and S.-K. Lim, "IEEE 802.15.7 visible light communication: Modulation schemes and dimming support," *IEEE Commun. Magazine*, vol. 50, no. 3, pp. 72–82, Mar. 2012.

[2] C. J. Mitchell, T. Etzion, and K. G. Paterson, "A method for constructing decodable de Bruijn sequences," *IEEE Trans. Inf. Theory*, vol. 42, no. 5, pp. 1472–1478, Sep. 1996.

[3] J. Pages, J. Salvi, and J. Forest, "A new optimised De Bruijn coding strategy for structured light patterns," in *Proc. Int. Conf. Pattern Recognition (ICPR)*, 2004, pp. 284–287.

[4] S. Schmid, G. Corbellini, S. Mangold, and T. R. Gross, "LED-to-LED visible light communication networks," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput. (MobiHoc)*, 2013.

[5] L. Li, P. Hu, C. Peng, G. Shen, and F. Zhao, "Epsilon: A visible light based positioning system," in *Proc. USENIX Symp. Netw. Syst. Design and Implementation (NSDI)*, 2014, pp. 331–343.

[6] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Commun.*, vol. 39, no. 20, pp. 1482–1493, Oct. 1991.

[7] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *Proc. Symp. Operating Syst. Design and Implementation (OSDI)*, 2002, pp. 147–163.

[8] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2003, pp. 138–149.

[9] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *Proc. Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2004, pp. 39–49.

[10] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2011, pp. 73–84.

[11] L. Li, G. Xing, L. Sun, W. Huangfu, R. Zhou, and H. Zhu, "Exploiting FM radio data system for adaptive clock calibration in sensor networks," in *Proc. Int. Conf. Mobile Syst., Appl., and Services (MobiSys)*, 2011, pp. 169–182.

[12] Z. Li, C. Li, W. Chen, J. Dai, M. Li, X.-Y. Li, and Y. Liu, "FLIGHT: Clock calibration using fluorescent lighting," in *Proc. Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2012, pp. 329–340.

[13] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2004, pp. 95–107.

[14] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: A short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proc. Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2006, pp. 307–320.

[15] S. Liu, K.-W. Fan, and P. Sinha, "CMAC: An energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks," in *Proc. IEEE Int. Conf. Sensor and Ad Hoc Commun. Netw. (SECON)*, 2007.

[16] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proc. Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, 2004, pp. 95–107.

[17] A. El-Hoiydi and J.-D. Decotignie, "WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks," in *Proc. Int. Symp. Comput. Commun. (ISCC)*, 2004, pp. 244–251.

[18] M. Basseville and I. V. Nikiforov, *Detection of Abrupt Changes: Theory and Application*. Upper Saddle River, NJ, USA: Prentice-Hall, 1993.

[19] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki – a lightweight and flexible operating system for tiny networked sensors," in *Proc. IEEE Int. Conf. Local Comput. Netw. (LCN)*, 2004, pp. 455–462.