# Sensifi: A Wireless Sensing System for Ultra-High-Rate Applications

Chia-Chi Li, Vikram K. Ramanna, Daniel Webber, Cole Hunter, Tyler Hack, and Behnam Dezfouli*
Internet of Things Research Lab, Santa Clara University, USA
{cli1, vramanna, dwebber, chunter, thack, bdezfouli}@scu.edu

***Abstract*—Wireless Sensor Networks (WSNs) are being used in various applications such as structural health monitoring and industrial control. Since energy efficiency is one of the major design factors, the existing WSNs primarily rely on low-power, *low-rate* wireless technologies such as 802.15.4 and Bluetooth. In this paper, by proposing Sensifi, we strive to tackle the challenges of developing *ultra-high-rate* WSNs based on the 802.11 (WiFi) standard. As an illustrative structural health monitoring application, we consider spacecraft vibration test and identify system design requirements and challenges. Our main contributions are as follows. First, we propose packet encoding methods to reduce the overhead of assigning accurate timestamps to samples. Second, we propose energy efficiency methods to enhance the system's lifetime. Third, to enhance sampling rate and mitigate sampling rate instability, we reduce the overhead of processing outgoing packets through the network stack. Fourth, we study and reduce the delay of processing time synchronization packets through the network stack. Fifth, we propose a low-power node design particularly targeting vibration monitoring. Sixth, we use our node design to empirically evaluate energy efficiency, sampling rate, and data rate. We leave large-scale evaluations as future work.**

***Index Terms*—Sensing, 802.11, WiFi, Low-Power, RTOS, Packet Processing, Vibration Test.**

## I. Introduction

The significant reductions in the cost and energy consumption of microprocessors, wireless transceivers, and sensors have facilitated the development of Wireless Sensor Networks (WSNs) for Structural Health Monitoring (SHM) as well as applications such as Process Automation (PA), Factory Automation (FA), and medical monitoring [1]–[5]. WSNs are being used in SHM applications to monitor spacecraft, aerial vehicles, high-rise buildings, dams, highways, and bridges [2], [6]–[8]. In these applications, sensor nodes collect information such as acceleration, ambient vibration, load, and stress at sampling frequencies usually above 100 Hz [9]. The replacement of cables with wireless links reduces design, deployment, and maintenance costs. Also, wireless links offer higher reliability compared to cables that are subject to wear and tear.

In this paper, we use the term *ultra-high-rate* to refer to the systems whose data transmission rate is in the range of few hundreds of Mbps to a few Gbps. The ultra-high-rate of communication is the result of high-rate, high-resolution sampling. Applications such as FA, Power Systems Automation (PSA), and Power Electronics Control (PEC) are representative of ultra-high-rate scenarios [10]. A prominent example of SHM is spacecraft vibration monitoring. These tests are performed to analyze a spacecraft's mechanical durability against the launch's powerful vibrational forces. These tests are conducted with a large number of wired accelerometers (more than 200), simultaneously sampled by expensive, high-performance Data Acquisition (DAQ) modules [11]. The large number of nodes and the high sampling rate (around 50 kHz) result in generating a massive amount of data that must be collected from the structure. This results in a disarray of wired connections that make installing and debugging these sensors an excessively laborious and time-consuming process. In addition to the setup difficulties, the extra weight applied to the spacecraft by the cables can skew vibrational measurement results [12]. The cables also introduce electrostatic noise that affects data collection accuracy.

Besides satisfying the high sampling rate required by these applications, the additional system requirements are: (i) *liveness*: collecting data from all the nodes while sampling is in progress, (ii) *scalability*: live data collection from a large number of nodes (e.g., more than 200 in spacecraft monitoring), (iii) *energy efficiency*: the nodes' longevity must satisfy the duration requirement of the experiment, (iv) *accuracy*: accurate timestamps must be assigned to the samples, and (v) *reliability*: lossless collection of raw samples is required. Unfortunately, the existing systems do not address these requirements. We highlight design challenges and the shortcomings of existing systems as follows. *First*, considering the energy limitations of sensor nodes, 802.15.4 and 802.15.1 are the primary wireless technologies used in existing works [13]. Consider a node collecting 12-bit samples at 50 ksps. This node needs to communicate at 600 kbps, excluding the overhead of the samples' timestamps; hence, the 250 kbps data rate provided by the 802.15.4 standard is less than the communication demand of *one node*. Similarly, the wireless technologies used in current industrial networks, such as WirelessHART [14], WIA-PA [15], ISA 100 [16], WSAN-FA/IO-Link [17], and WISA [18] cannot be used for ultra-high-rate sensing applications. *Second*, the liveness require-

*Corresponding Author.

ment is not satisfied if samples are stored in the main memory or a memory card and then transferred whenever the user requests for data delivery [19]–[22]. For example, spacecraft vibration tests require live response monitoring, identifying malfunctioning nodes, and making adjustments to the test configuration parameters. Besides, frequent placement and removal of nodes in hard-to-access areas are not desirable. *Third*, compression and in-network processing methods are lossy and cannot be used in scenarios where raw data collection is required to implement various data analysis methods [7], [23]. Furthermore, the compression level provided by these methods is not enough to adapt low-rate wireless technologies for ultra-high-rate applications. Also, the high processing overhead of these compression algorithms causes a considerable interruption in sample collection. *Fourth*, variations of inter-sample delay make assigning accurate timestamps to samples more challenging, and achieving sampling rate stability is particularly difficult in ultra-high-rate applications. Running additional tasks on the processor causes interruptions of sample collection and variations of the sampling rate. Therefore, the effect of running additional tasks, such as packet processing and wireless transmission, must be minimized. The existing work has identified this challenge; however, they do not offer a comprehensive evaluation or mitigation of these problems [20], [24].

In this paper, we propose Sensifi, a system based on 802.11 (WiFi) standard for ultra-high-rate sensing applications. Towards tackling the underlying challenges of developing this system, *the main contributions of this paper are as follows: First* (§IV), we show that assigning accurate timestamps ($\mathrm{sub}-\mu\mathrm{s}$ accuracy) to samples introduces a significant overhead and wastes precious payload bytes and wireless communication bandwidth. Considering the distribution of inter-sample intervals, we propose three encoding methods to reduce payload overhead. These methods achieve various levels of payload efficiency and execution time. By reducing the amount of data sent per node, we enhance system scalability and reduce the nodes' energy consumption to transmit the collected samples. *Second* (§V), we propose methods to reduce the energy consumption of nodes during their inactivity periods. In particular, since the 802.11 transceiver is the primary energy consumption source, we study two energy-efficiency methods: periodic association with the Access Point (AP), and extended-period beacon reception. We also propose and empirically evaluate mechanisms to enhance the energy efficiency of these methods further. Depending on the application scenario, the proposed methods and configurations can be used to achieve the desired energy-delay tradeoff. *Third* (§VI), we show that preparing and processing outgoing packets by the network stacks introduce a non-negligible delay that must be minimized to increase sampling rate and stability. Our work reveals that bypassing transport layer processing is essential to minimize the effect of packet processing on sampling rate, and this bypassing also translates into the higher performance of timestamp encoding. *Fourth* (§VII), we present a thorough study of time synchronization and identify the challenges caused by network stack complexity. We place the time synchronization function in the 802.11 subsystem, between the packet decoding logic and driver, and confirm that the maximum error is bounded by $0.25\ \mu$s. *Fifth* (§VIII), we present a modular, low-power node design. This node is capable of collecting 16-bit samples at rates up to 500 ksps. Therefore, it enables us to measure sampling rate variations, study the impact of packet processing on the sampling rate, and evaluate energy consumption. Compared with the existing works that rely on TinyOS [7], [24], [25], Linux [26]–[28], or theoretical performance analysis [29], [30], our hardware platform supports the two widely-used Real-Time Operating Systems (RTOSes) targeting resource-constrained devices: FreeRTOS [31], [32] and ThreadX [33], [34]. Also, we use the network stacks available for these RTOSes, namely, LwIP [35], NetX [36], and NetXDuo [37]. Therefore, a unique aspect of our work is identifying and tackling the underlying challenges of using these tools for developing ultra-high-rate applications. *Sixth* (§IX), using our node design, we empirically evaluate various aspects of the system and demonstrate the performance enhancements achieved with the proposed methods. However, we leave large-scale throughput evaluation of the system as future work. The methods proposed in this paper can be employed for developing WSN systems for applications such as PA, FA, Building Automation System (BAS), intra-vehicle communication, and Wireless Avionics Intra-Communications (WAIC).

## II. Sample Application Scenario

Although the proposed system can be used in various ultra-high-rate sensing applications, we overview spacecraft vibration monitoring to identify and clarify the unique challenges of this application and frame our system design.

Spacecraft vibration test monitoring is a SHM application that requires live, ultra-high-rate data collection from a large number of nodes (usually more than 200) [38], [39]. The main objective of this test is to excite components, subsystems, and nonstructural hardware that respond to the launch environments. The spacecraft is attached to a large shake table, and then, accelerometers are mounted. This process is referred to as the *mounting phase* or *Idle phase* because nodes do not perform sampling during this phase. This phase is the most laborious and time-consuming step of the process. Each sensor has a short wire with a connector attached at the end. Engineers at Space Systems Loral (SSL) have informed us that depending on sensor placements, partial disassembly of the satellite may occur to facilitate the sensor's placement in the exact locations needed to collect data. From there, longer cables are run from the sensors to the DAQ. The mounting process can take between one to two weeks to attach the devices before the test is started. The cables have to be carefully mounted to the spacecraft for two primary reasons: avoid damage to the spacecraft during the test, and prevent significant changes to the spacecraft's load properties. The latter is critical because if one wants to add more sensors to study the system better, the added weight may alter the system's load properties, which is in direct opposition to the test's goal. This is because the more sensors added, the more likely the results are not representative of the realistic properties of the spacecraft [12]. Given that each cable performs slightly differently and has a unique transfer function, an individual profile needs to be generated so that these differences can be calibrated out. This wired solution is costly and lacks versatility.

After the mounting phase, each sensor has to be manually verified. Verification is critical because engineers need to ensure that all sensors are functioning properly before the test is conducted. After verification, the tests begin. This process is referred to as the *Sampling phase*. During the test, sensors are sampled continuously for 20 to 40 minutes.

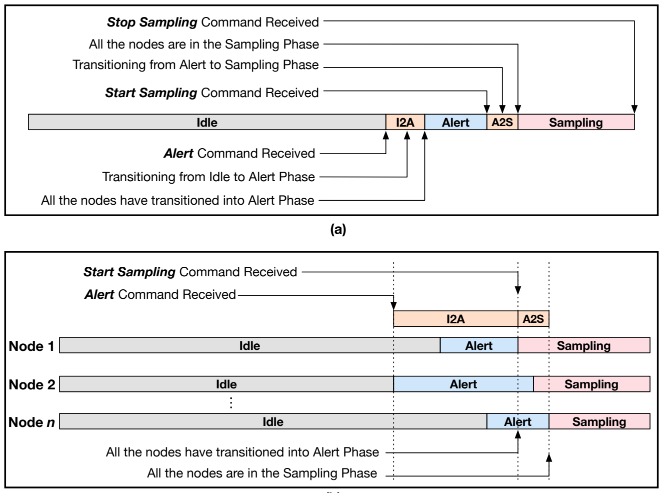

*Stop Sampling* Command Received
All the nodes are in the Sampling Phase
Transitioning from Alert to Sampling Phase
*Start Sampling* Command Received
Idle
I2A
Alert
A2S
Sampling
*Alert* Command Received
Transitioning from Idle to Alert Phase
All the nodes have transitioned into Alert Phase
(a)
*Start Sampling* Command Received
*Alert* Command Received
I2A
A2S
Node 1
Idle
Alert
Sampling
Node 2
Idle
Alert
Sampling
Node *n*
Idle
Alert
Sampling
All the nodes have transitioned into Alert Phase
All the nodes are in the Sampling Phase
(b)

Fig. 1. (a) The operational phases of the overall system. (b) The operational phases of nodes. Since the transition time of nodes varies depending on energy efficiency configuration, transition periods exist from the overall system's perspective. These transition periods are named Idle to Alert (I2A) and Alert to Sampling (A2S).

## III. System Overview

The system's three primary components are *nodes*, *AP*, and a *server*. Each node is a wireless sensor device capable of sampling, processing, and wireless communication. AP receives packets from nodes and forwards them to the server. The server coordinates network operation, decodes the received packets, processes the collected data, and generates time synchronization packets.

The basic components of a node are sensor (an accelerometer in our node design), Analog to Digital Converter (ADC), processor, and wireless transceiver. The ADC used in our node design is capable of sampling at up to 500 ksps and is interfaced with the processor using Serial Peripheral Interface (SPI) bus. The System on a Chip (SoC) is based on CYW54907 [40], which supports 802.11ac. This SoCs includes two ARM-Cortex R4 processors, one processor is dedicated to the *application subsystem*, and the other one is assigned to the *wireless subsystem*. The former subsystem runs user code and the latter runs 802.11 firmware to handle time-critical tasks such as backoff and acknowledgment packet generation. We discuss hardware design in §VIII.

Considering the complexities of 802.11 standard, in this paper, we show that using various power saving modes are necessary to facilitate system development for a wide range of applications. We refer to these power saving modes as *Idle* and *Alert*, where the former achieves higher energy efficiency while its responsiveness to incoming messages is slower than that of the latter. We will explain these in §V.

With regard to the vibration monitoring application (§II), Figure 1 presents the operational phases of the system. We detail Figure 1(a) as follows. During the mounting phase, which may last between one to two weeks, nodes are mounted on the spacecraft. Before mounting each node, the node is powered on and enters the Idle phase. Once a node receives the *Alert Command* from the server, the node transitions into the Alert phase and waits to receive a *Start Sampling* command from the server to start the *Sampling* phase. Figure 1(b) shows this transition for three nodes. Since the nodes transition from the Idle phase to the Alert phase at different times, there is a transition period for the system to ensure all the nodes are in the Alert phase. This delay is referred to as the Idle to Alert (I2A) transition. Similarly, there exist an Alert to Sampling (A2S) transition period. Depending on the energy efficiency mode employed (§V), I2A duration varies between a few seconds to a few minutes; whereas, A2S varies between a few hundreds of milliseconds to a few seconds.

The different energy efficiency methods used during the Idle phase and Alert phase justify the need for a low-cost transition from the Idle phase to the Sampling phase. Specifically, if we send a Start Sampling command to the nodes in the Idle phase, they start sampling at time instances that may be up to a few minutes apart. Therefore, while some nodes are consuming higher energy to collect and transmit samples, the rest are still in low-power mode; and this results in partial system monitoring and energy waste by some nodes. Considering the high energy consumption of nodes during the Sampling phase, we need to ensure all the nodes start the Sampling phase almost synchronously, however, this is not possible considering the deep sleep mode used during the Idle phase. Therefore, we use an intermediate energy efficiency method—the Alert phase—to minimize the intervals between the instances the nodes transition into the Sampling phase.

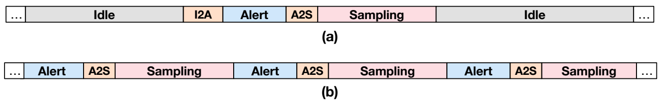


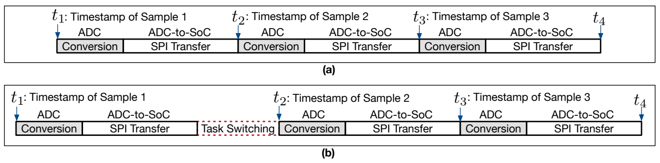
Fig. 2. (a) A system with long intervals between Sampling phases. (b) A system with short intervals between Sampling phases.

**Algorithm 1:** The Overall Control Flow of Each Node During the Sampling Phase

```
function main_thread()
    $n = 0$ /* initialize packet sequence number */
    while sampling_phase do
        $n = n + 1$
        for $i = 0 ; i < s ; i++$ do
            /* $s$ is the maximum number of samples per packet. We refer to $s$ as sampling batch size */
            $T[i]$ = get_nanosec_clock()
            $S[i]$ = get_ADC_sample() /* 2 bytes */
            if $i > 0$ then
                $I[i] = T[i] - T[i-1]$ /* calculate inter-sample interval */
        intv_freq_count($I[s], H[s], used$) /* calculate repetitions per inter-sample interval */
        intv_freq_ranking($H[s], used$) /* rank inter-sample intervals based on repetitions */
        packet_encoding($H[s], count, S[s], I[s], pkt$)
        packet_preparation($pkt$)
        packet_send($pkt$)
    return
```

The Idle and Alert phases can be used to develop various 802.11-based WSNs. For example, for an application that requires periodical sampling of a structure at long intervals, the above three phases can be used, as Figure 2(a) shows. For applications where the interval between Sampling phases is short, only the Alert phase can be used to enhance the energy efficiency of the system, as Figure 2(b) shows.

Algorithm 1 presents the operation of a node during the Sampling phase. Each node performs sampling, encoding, and packet transmission sequentially. These operations repeat until the Stop Sampling command is received from the server. The *sampling task* communicates with the ADC to collect $s$ samples (lines 4 to 9). We refer to $s$ as the *sampling batch size*. The collected samples and timestamps are then encoded to reduce the overhead of timestamp assignment. This is achieved by first calculating repetitions per inter-sample interval (line 10), and then ranking inter-sample intervals based on repetitions (line 11). We detail packet encoding in §IV. Next, the proper data structures are allocated to the packet (line 13), and then the packet is processed by the communication protocol stack (line 14). We discuss packet creation and processing in §VI.



$t_1$: Timestamp of Sample 1
$t_2$: Timestamp of Sample 2
$t_3$: Timestamp of Sample 3
$t_4$
ADC
ADC-to-SoC
Conversion
SPI Transfer
(a)
Task Switching
(b)

Fig. 3. Sample collection from ADC. Both the conversion time of ADC and sample transfer time affect inter-sample intervals. Also, further variations of inter-sample intervals occur when the processor switches between tasks.

## IV. Sampling and Packet Encoding

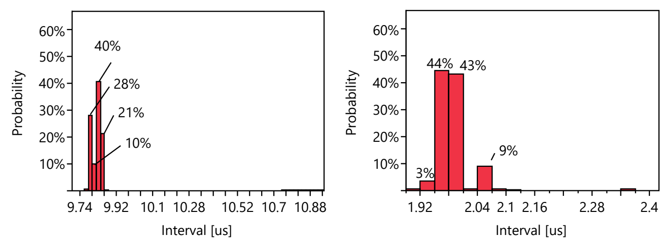
The server collecting samples must be able to accurately reconstruct the signals measured by the nodes. If samples are collected at equidistant intervals, one timestamp per packet can specify the timestamps of all the samples included in the packet. However, there are variations in sampling intervals; hence, including one timestamp per packet reduces the timing accuracy of the measurement event. On the other hand, assigning a per-sample timestamp introduces a significant overhead.

In this section, we propose methods for efficient and lossless compression of timestamps. Although these methods are application-agnostic, they are tailored for the packet size limitation of 802.11. In addition to *compression efficiency*, we also consider *execution time* as an important design metric to minimize the effect of encoding on sampling rate.

### A. Sampling Interval Variations

In this section, we focus on inter-sample intervals from two points of view: ADC's sample conversion time and the delay of transferring a sample from the ADC to the processor. The design of Successive Approximation Register (SAR) ADCs includes an internal switched-capacitor Digital to Analog Converter (DAC), also known as charge redistribution DAC. The DAC uses capacitors to store the voltage at the input and uses a switch to disconnect the capacitors from the input [41]. The switched-capacitor structure's conversion time scales exponentially with the resolution of ADC. Besides resolution, the amount of time to hold the capacitor value, impedance of the internal analog multiplexer, output impedance of the analog source, and the switch impedance are the factors impacting ADC's conversion time. In addition to conversion time, the amount of time required to transfer a sample from the ADC to the processor may vary. Figure 3(a) presents the process of sample collection from ADC. At the time $t_1$, and right before collecting the first sample, the processor collects the timestamp for Sample 1. The processor then triggers a pin of the ADC, at which point the ADC starts the conversion. Once the ADC finishes the conversion, the SPI communication starts to transfer the converted sample from the ADC to the SoC.

To measure variations of inter-sample intervals (i.e., differences among $t_2 - t_1$, $t_3 - t_2$, etc.), we placed a node



(a) 100 ksps at 25 °C (b) 500 ksps at 25 °C

Fig. 4. Distribution of sampling interval under different sample frequency. (a): Sampling rate 100 ksps. (b): Sampling rate 500 ksps. For these experiments, the only task being run by the nodes is sampling. Nearly 99% of the intervals are within four to six classes, and the remaining 1% of the intervals are distributed among several classes. It is worth mentioning that we observed similar distributions versus temperature variations.

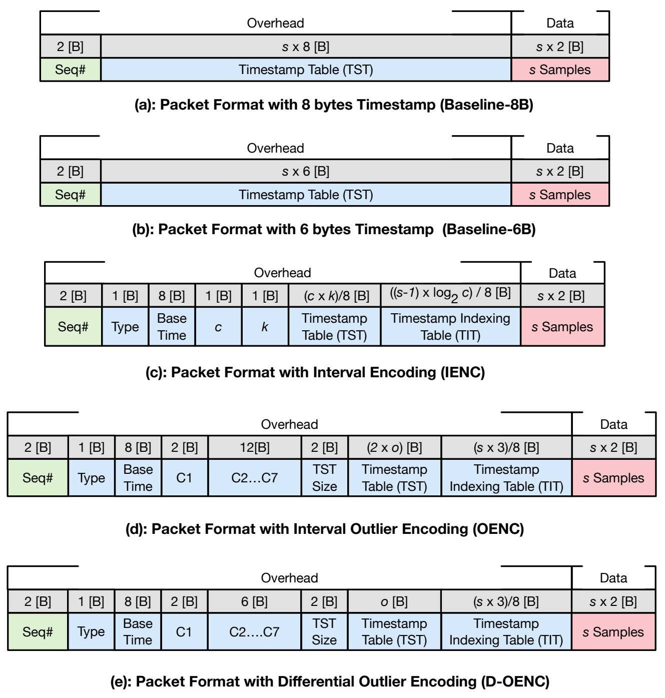
inside a controlled temperature chamber and used a high-accuracy logic analyzer to capture inter-sample intervals. For these experiments, only the sampling task is run by the node; this means the processor runs a thread responsible for triggering the ADC and transferring samples from the ADC to the processor. Figure 4 shows inter-sample intervals. For 100 ksps, we observe up to 0.11 $\mu$s variations around the mean value of 10 $\mu$s, and for 500 ksps, the variations are up to 0.016 $\mu$s around the mean value of 2 $\mu$s. Considering the 25 ns timing precision provided by the logic analyzer, nearly 99% of the intervals are within four to six classes for both sampling rates. The other 1% of the intervals are distributed among several classes.

Based on these observations, considering inter-sample variations is a crucial factor to improve timestamp encoding efficiency. If we include only one timestamp per packet, these errors accumulate as more samples are included in the packet. For example, when sampling at 500 ksps, for a packet including 500 samples, the error of the timestamp inferred for the last sample is about 8 $\mu$s. This error is about 55 $\mu$s for 100 ksps.

The inter-sample intervals reported in this section are irrespective of the load of the processor. However, as Figure 3(b) shows, switching to other tasks exacerbates the variations of inter-sample intervals. In §VI-B, we will demonstrate and mitigate the effect of running other tasks (such as packet processing) on the variations of inter-sample intervals. It is also important to note that the inter-sample variations reported in this section are pertaining to the specific hardware components used in our node design (§VIII). Mitigation or elimination of inter-sample variations via hardware design is out of the scope of this work and is left as future work.

## B. Encoding Methods

*1) Baseline:* Our baseline method refers to assigning a timestamp per sample without applying any encoding. We discuss two baseline methods as follows.



| Overhead | | Data |
|---|---|---|
| 2 [B] | s x 8 [B] | s x 2 [B] |
| Seq# | Timestamp Table (TST) | s Samples |

**(a): Packet Format with 8 bytes Timestamp (Baseline-8B)**

| Overhead | | Data |
|---|---|---|
| 2 [B] | s x 6 [B] | s x 2 [B] |
| Seq# | Timestamp Table (TST) | s Samples |

**(b): Packet Format with 6 bytes Timestamp (Baseline-6B)**

| Overhead | | | | | | | Data |
|---|---|---|---|---|---|---|---|
| 2 [B] | 1 [B] | 8 [B] | 1 [B] | 1 [B] | (c x k)/8 [B] | ((s-1) x $\log_2$ c) / 8 [B] | s x 2 [B] |
| Seq# | Type | Base Time | c | k | Timestamp Table (TST) | Timestamp Indexing Table (TIT) | s Samples |

**(c): Packet Format with Interval Encoding (IENC)**

| Overhead | | | | | | | | Data |
|---|---|---|---|---|---|---|---|---|
| 2 [B] | 1 [B] | 8 [B] | 2 [B] | 12[B] | 2 [B] | (2 x o) [B] | (s x 3)/8 [B] | s x 2 [B] |
| Seq# | Type | Base Time | C1 | C2…C7 | TST Size | Timestamp Table (TST) | Timestamp Indexing Table (TIT) | s Samples |

**(d): Packet Format with Interval Outlier Encoding (OENC)**

| Overhead | | | | | | | | Data |
|---|---|---|---|---|---|---|---|---|
| 2 [B] | 1 [B] | 8 [B] | 2 [B] | 6 [B] | 2 [B] | o [B] | (s x 3)/8 [B] | s x 2 [B] |
| Seq# | Type | Base Time | C1 | C2….C7 | TST Size | Timestamp Table (TST) | Timestamp Indexing Table (TIT) | s Samples |

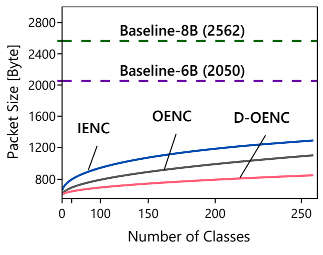
**(e): Packet Format with Differential Outlier Encoding (D-OENC)**

Fig. 5. Packet formats of various timestamp encoding methods. (a) Packet format with an 8 byte timestamp per sample (Baseline-8B). (b) Packet format with an 6 byte timestamp per sample (Baseline-6B). (c) Packet format using Interval Encoding (IENC). (d) Packet format using Outlier Encoding (OENC) where 2 bytes are assigned to each outlier. (e) Packet format with Differential Outlier Encoding (D-OENC) where 1 byte is assigned to each outlier.

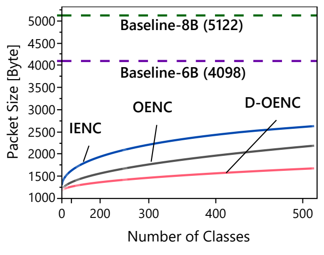
The SoC provides a nanosecond timer that can be read into an 8-byte variable. Figure 5(a) shows the baseline packet format where an 8-byte timestamp is added per sample. Each sample is 2 bytes. *Seq#* is the packet sequence number. Considering the 1472-byte Maximum Transmission Unit (MTU), we can include a maximum of 147 samples per packet, which means 80% of the payload is occupied by timestamps. We refer to this method as *Baseline-8B*, as demonstrated in Figure 5(a).

The time duration provided by an 8-byte, nanosecond-granularity variable is over 500 years, which is more than what is required in real-world applications. Assuming time synchronization is only required during the Sampling phase, a 60-minutes Sampling duration requires 42 bits to provide nanosecond timing accuracy. Therefore, we use a 6-byte nanosecond timestamp, which wraps around every 78 hours. With a 6-byte timestamp, we can include a maximum of 183 samples per packet, introducing a 75% overhead per packet for timestamps. We refer to this method as *Baseline-6B*, as demonstrated in Figure 5(b).

To reduce the overhead of adding timestamps and increase the number of samples sent per packet, we present three lossless encoding methods and compare their performance against the baselines, as follows.



(a) 256-Samples Batch



(b) 512-Samples Batch

Fig. 6. Packet size comparison across the three encoding methods. X-axis represents the number of classes in a batch of samples ($s$). Note that the x-axis increments by the power of 2.

*2) Interval Encoding (IENC):* The basic idea is to encode sampling interval classes into an index table. The overhead of timestamp encoding varies based on the distribution of the interval classes. Figure 5(c) shows the packet format of this encoding method. The two main tables used for timestamp encoding are Time-stamp Table (TST) and Time-stamp Indexing Table (TIT). The Time-stamp Table (TST) includes the classes of sample conversion intervals, and each interval is encoded with 2 bytes (instead of 8 bytes). The total number of interval classes is denoted by the variable $c$. Each interval is represented as a coefficient of 1 ns, and the number of bits required to represent each class is $k$. Two variables $c$ and $k$ are used to inform the server about the size of the table to decode. For each sample $S[i]$ where $i > 1$, there is a corresponding entry in the TIT that refers to an entry of TST. Since TST has $c$ entries, each entry in the TIT is $\log_2 c$ bits. With this, the TIT specifies the interval between each pair of consecutive samples. *Base Time* is the nanosecond timestamp of the first sample in the packet.

Figure 6 shows the theoretical comparison of packet size using different encoding methods. The x-axis of sub-figures (a) and (b) represent the number of possible interval classes considering 256- and 512-sample batches, respectively. Comparing with the baseline methods (horizontal lines), IENC reduces payload overhead by at least $48\%$ compared to Baseline-8B and $35.6\%$ compared to Baseline-6B.

With 512-sample batches, IENC hits the payload constraint when more than $12.5\%$ of the intervals are unique. When reaching the payload limitation, IENC stops processing the TST and encodes the samples into two 256-sample packets. Notice that these two sub-divided packets have their own Base Time, TIT, and TST to ensure a packet loss would not influence the other one. We use the *Type* field to notify the server if this packet was sub-divided. The value 0 means only one packet, value 1 means the first of two packets, and value 2 represents the second of two packets.

To further improve encoding efficiency, we note that IENC does not consider the repetitions of interval classes. We rely on this observation and propose two encoding methods, as follows.

*3) Outlier Encoding (OENC):* This method leverages the statistical pattern of the classes to reduce encoding overhead. Figure 4 shows that a few classes include the majority of the intervals. We refer to these classes as $C1$ through $C7$. For example, in most 512-sample batches, only 6 intervals are outside of these major classes. We use this observation to further reduce the TIT size. Figure 5(d) shows the packet format used by OENC. The basic idea is to reduce TIT size by using a 3-bit index for each sample. This method uses a fixed TIT size instead of letting the size grow by the number of interval classes. In the TIT, there is a 3 bit index for each sample; therefore the size of TIT is $s \times 3$ bits. Index values 000 through 110 refer to $C1$ through $C7$. If the index of a sample is 111, it means the interval is an outlier and must be found in the TST. For each index value 111 in the TIT, there is a corresponding outlier value in the TST. In other words, the number of entries with index value 111 in the TIT is the same as the number of entries in TST. The *TST Size* field is used to inform the server about the size of TST.

Figure 6 shows that OENC reduces packet payload size by at least $56\%$ compared to *Baseline-8B* and $45\%$ compared to *Baseline-6B*. Compared to IENC, the reduction is about $8\%$ with the maximum number of outliers. OENC hits the payload constraint when more than $23\%$ of the intervals in a 512-sample batch are unique. Once it reaches the limit, OENC stops processing the TST and sub-divides the samples into two packets. These two packets create their own $C1$ to $C7$, TST, TST size, and the TIT field values. Figure 6(a) shows that OENC's worst case scenario with 256 samples consumes 1132 bytes, which is within the payload limit. As Figure 5(d) shows, TIT always consumes 96 and 192 bytes of the payload for 256- and 512-sample batches, respectively; whereas, TST grows as the variations of sampling intervals increase. In the worst case, OENC's TST field occupies about $46\%$ of the payload, which is the second-largest payload consumption field after the samples field.

*4) Differential Outlier Encoding (D-OENC):* This method uses lossless compression of TST to reduce the number of bits required to encode each outlier. Instead of encoding actual interval values, we encode a signed number to indicate the difference between each outlier and the first majority class, i.e., $C1$. In other words, we encode the difference between the expected value and perceived value. This signed number represents the difference as a multiple of clock accuracy (one nanosecond). For example, if an inter-sample interval $I[i]$ is an outlier, the TST entry corresponding to this outlier is $I[i] - I[C1]$, where $I[C1]$ is the inter-sample interval of class $C1$. Figure 5(e) presents the packet format.

Regarding performance, Figure 6 shows that D-OENC's packet payload size is reduced by at least $66\%$ compared to *Baseline-8B* and $57\%$ compared to *Baseline-6B*. This is a $9\%$ reduction compared with OENC when all the outliers are unique. In addition, D-OENC hits payload constraint when more than $47\%$ of the intervals are unique with 512-sample

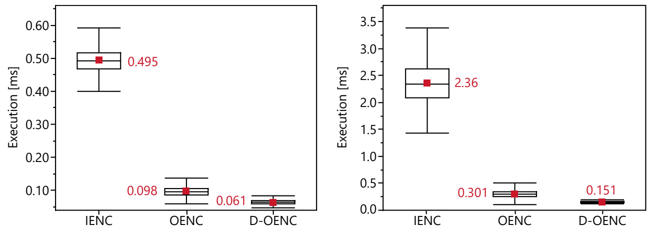
**Algorithm 2:** Interval Classification Algorithm

```
1  function intv_freq_count ($I[s], H[s], used$)
2      $used = 0$ /* consumed array space */
3      $packet\_count = 1$ /* number of packets */
       /* generate interval frequency hash table */
4      for $i = 0 ; i < s ; i++$ do
5          if $used < payload\_limit$ then
6              for $j = 0 ; j < s ; j++$ do
7                  if $i == 0$ then
8                      $H[used].value = I[i]$
9                      $H[used].count = 1$
10                     break
11                 else if $j > used$ then
12                     $used++$
13                     $H[used].value = I[i]$
14                     $H[used].count = 1$
15                     break
16                 else
17                     if $H[j] == I[i]$ then
18                         $H[j].count++$
19                         break
20         else
21             $packet\_count = 2$
22             break
       /* only for IENC and OENC */
23     if $packet\_count == 2$ then
24         generate the interval frequency hash table with $s/2$
25     return $H[s]$,$used$
```

batches. Once it reaches the limitation, D-OENC simply splits the 512 samples into two packets. As Figure 6(b) shows, D-OENC's worst-case scenario with a 512-sample batch consumes 1741 bytes. When split into two packets, each packet consumes 1229 bytes, well within the payload limit.

## C. Interval Classification Algorithm

We use the hash table as a simple and effective method to count the frequency of sampling intervals, as demonstrated in Algorithm 2. In our implementation, we only use stack memory to avoid the computation time uncertainty of dynamic memory. The variable $used$ keeps track of the consumed array space by the number of classes. Line 5 shows that when the number of classes reaches the $payload\_limit$, the algorithm breaks the loop and recomputes the hash table with half of the samples. This recomputing process happens with IENC and OENC only. After generating the frequency hash table $H[s]$, we use this table to compute the probability distribution of sampling intervals. Function `intv_freq_ranking()` is used to find the top seven classes—$C1$ through $C7$. The function computation time is based on the size of the consumed portion of $H[s]$, which is $O(used)$. Only OENC and D-OENC use this function to rank the top seven classes.



(a) 256-Sample Batch (b) 512-Sample Batch

Fig. 7. Empirical evaluation of the execution time of the proposed encoding methods. The processor is an ARM-Cortex R4 operating at 160 MHz. The numbers next to box plots represent mean values. IENC shows the largest variations because the chance of breaking a batch of samples into two packets is the highest with this encoding method. D-OENC achieves the minimum execution time among the three methods.

## D. Execution Time

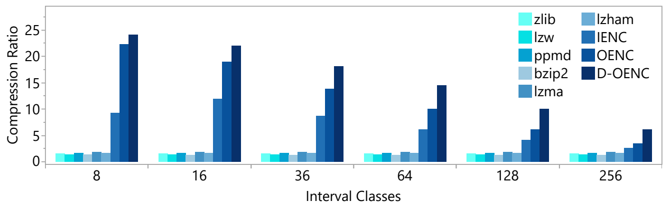
Figure 7 shows the empirical evaluation of execution time. The IENC computation time is more than 5x and 7x longer than OENC and D-OENC when using 256- and 512-sample batches, respectively. This is mainly because the number of interval classes impacts IENC's computation time linearly, causing these large variations. Using 512-sample batches, IENC hits the packet payload limit when more than 12.5% of the intervals are unique (64 interval classes). This situation results in encoding the samples into two packets, which requires IENC to recompute the hash table. The OENC method performs a similar operation when more than 23% of the intervals are unique. However, D-OENC does not need to recompute the hash table when splitting into two packets is required.

As Algorithm 1 shows, during the Sampling phase, each node sequentially performs sampling, encoding, packet preparation, and transmission. Therefore, a shorter encoding period is essential to achieve a stable and higher sampling rate. When the sampling rate is 100 ksps, these results show that IENC, OENC, and D-OENC result in missing about 49, 9, and 6 samples after collecting each 256-sample batch. When the batch size is 512, these values are 236, 30, and 15 samples. In summary, the results presented in Figures 6 and 7 confirm the superiority of D-OENC in terms of packet size efficiency and execution duration. When the batch size increases from 256 to 512, the execution time of D-OENC shows a 147% increase. Although using the smaller batch seems to be more efficient, we show in §IX that using the larger batch results in higher sampling efficiency.

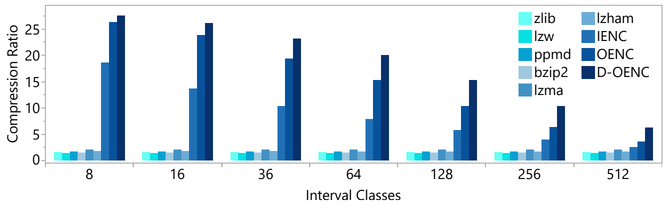
## E. Compression Ratio

This section compares the proposed algorithms versus a set of well-known lossless algorithms that use the static dictionary and adaptive dictionary techniques. Specifically, we consider the followings:

- Lempel-Ziv algorithms: Sliding Window LZ77 (*zlib*) and Dictionary Based LZ78 (*lzw*) [42],



(a) 256-Samples Batch



(b) 512-Samples Batch

Fig. 8. Comparing the compression ratio of the proposed algorithms versus standard lossless compression algorithms. The x-axes represent the number of the timestamp interval classes in a batch of (a) 256- and (b) 512-samples.

- Markov modeling following by arithmetic coding: Prediction with Partial Match (*ppmd*) [42],
- Burrows-Wheeler Transform (bzip2) [43],
- Lempel–Ziv–Markov chain algorithm: standard (*lzma*) and advanced (*lzham*) [44].

The datasets used in examining the algorithms include timestamps, and we consider a various number of classes in 256- and 512-sample batches. Figure 8 presents the empirical evaluation of the compression ratio. One of the unique features of the proposed algorithms is to encode the non-repetitiveness of timestamps as inter-sample intervals. And for D-OENC, it encodes the differences between expected and realized intervals between timestamps. The results demonstrate that the proposed algorithms benefit from encoding intervals and achieve a higher compression ratio with fewer classes. Because of the non-repetitiveness of timestamps, the standard lossless algorithms can only achieve a maximum compression ratio of 1.87 and 2.02 for 256- and 512-sample batches, respectively. Whereas, the proposed algorithms can achieve compression ratios up to 24 and 27 for 256- and 512-sample batches. Even with cases where all the sample intervals are unique, the proposed D-OENC method achieves a maximum compression ratio of 6.15 and 6.27 for 256- and 512-sample batches, respectively.

## V. Reducing Energy Consumption During the Idle and Alert Phases

Achieving energy-delay tradeoffs with the 802.11 standard is more complicated than low-power standards such as 802.15.4 and 802.15.1. This is specifically because 802.11 stations need to periodically receive the beacons sent by the AP, or reassociate with the AP when they need to communicate. Considering these challenges, we propose two methods for establishing energy-delay tradeoffs and facilitating the development of various 802.11-based applications, regardless of the hardware platform used.

### A. *Energy Efficiency Methods*

We detail two energy efficiency methods in this section: *periodic association*, and *extended-period beacon reception*.

*1) Periodic Association:* In this method, each node periodically wakes up and associates with the AP, checks for a pending command from the server, and then either returns to an off mode (if no pending command) or transitions into the requested mode. The periodic association is triggered by a timer that fires every $t_p$ seconds to perform the association. The energy consumed per second is computed as follows:

$$\bar{E} = \Big(P_{asc} \times D_{asc} + P_{off} \times (t_p - D_{asc})\Big)/t_p \quad \text{for } D_{asc} < t_p \tag{1}$$

where $P_{asc}$ refers to the average power consumed during the association process, and $D_{asc}$ is the duration of the association process (the time it takes for the node to associate with an AP). $P_{off}$ is power consumption during the off mode.

Although a node's energy consumption can be reduced by increasing $t_p$, it is also possible to enhance energy efficiency by improving the association process. We utilize two methods to reduce this energy. (i) *Specific Association*: During the association process, a node sends probe packets on all channels to discover nearby APs. To avoid this overhead, we program the MAC address and channel of the AP into the node's driver. This allows each node to directly send an association request to the intended AP, thereby reducing probing overhead. Resilience against interference and association failures can be provided by adding a channel scan method to the nodes' embedded software. Specifically, suppose a node fails to connect to the AP over the programmed channel. In that case, it switches to the scan mode to determine the new operational channel of the AP. This new channel is then programmed into the driver to be used in subsequent associations. (ii) *Pairwise Master Key (PMK) Offloading*: During the association process between a node and the AP, the node needs to establish a secure channel with the AP, and this entails generating a PMK, which is a process-intensive task. Using the driver's APIs, we offload this key generation process to the application subsystem and then transfer the key to the wireless subsystem.

*2) Extended-Period Beacon Reception:* In this method, as soon as a node is turned on, it is associated with the AP. To ensure low-power operation while maintaining association, we employ Power Save Mode (PSM), which is the fundamental power saving mechanism supported by the 802.11 standard. With PSM, an associated node wakes up every 102.4 ms to receive a beacon packet from the AP. When there is a buffered packet for a node, the AP sets a flag (per node) in the beacon packet to inform the node that it must request for the delivery of the buffered packet. Although the standard wake-up interval used by commercial APs is 102.4 ms, we can use a longer interval to reduce the overhead of beacon reception further. We refer to this method as *extended-period beacon reception*. To this end, we modified the PSM configuration in the driver (of nodes) to extend the default listen interval by a factor of $l$. We refer to $l$ as the *listen interval coefficient*. With this method, the energy consumed per second is calculated as follows:

$$\bar{E} = P_{bcn} \times \left( \frac{D_{bcn}}{l \times 0.1024} \right) + P_{slp} \times \left( 1 - \frac{D_{bcn}}{l \times 0.1024} \right) \quad (2)$$

where $P_{bcn}$ and $P_{slp}$ are power consumption during beacon reception and sleep modes, respectively, $D_{bcn}$ is the duration of a beacon reception, and $l$ is the coefficient of the listen interval used by the node. Note that $D_{bcn}$ includes the duration a node waits in receive mode to receive a beacon, as well as the actual duration of beacon packet reception.

## B. Transition Delay

Using the periodic association method, although a larger value of $t_p$ results in a deeper sleep mode, the larger value also affects the nodes' responsiveness to commands. Assuming the command packet sent by the server can be generated at any time, the delay of command reception by a node is a uniform random variable in the range $[0, t_p]$ second. Using the extended-period beacon reception method, the delay of receiving the command is a uniform random variable in the range $[0, l \times 0.1024]$ second. We introduce a definition of transition delay that makes it independent of command packet generation instance: *transition delay is defined as the interval between the time instances the first and last node in the network receive a command packet.*

When using the periodic association method, Figure 9(a) shows that the transition delay is $t_p - D_{asc}$. For periodic association, the wake-up instance of the nodes can be coordinated if they are time-synchronized. Although it is desirable to allow all the nodes to associate with the AP concurrently, this results in channel access contention, extends the association duration, and increases the nodes' energy consumption. An alternative approach is to time synchronize the nodes and serialize their associations, as Figure 9(b) demonstrates. For a system including $N$ nodes, this method reduces transition delay to $(N-1) \times D_{asc}$. For this method, a clock accuracy of up to a few tens of milliseconds would suffice. Each node synchronizes its clock with the server at each association instance.

For the extended-period beacon reception method, the server can issue the command right before all the nodes' wake-up instance. However, although each node wakes up every $l \times 0.1024$ second, their wake-up instances are not synchronized; therefore, transition delay equals $(l-1) \times$ 0.1024. This is demonstrated in Figure 10(a). We employ the following method to tackle this challenge. Once each node associates with the AP, the server informs the node about the sequence number of the initial beacon ($b_{init}$) packet, where $b_i = b_{init} + k \times l, k \in \mathbb{N}$ represents the beacon instances that must be received by all the nodes. Assume a node associates with the AP and receives beacon number $b_i$. If $b_i = b_{init} + k \times l$, the node immediately enforces listen interval coefficient $l$. Otherwise, the node uses $l = 1$ until $\min(b_{init} + k \times l)$, where $b_i < b_{init} + k \times l$ and $k \in \mathbb{N}$.

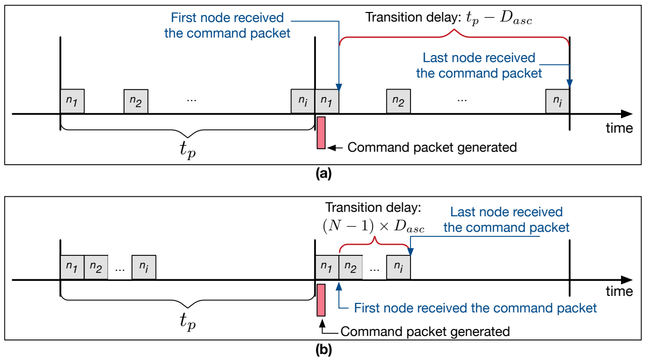

First node received the command packet
Transition delay: $t_p - D_{asc}$
Last node received the command packet
$n_1$
$n_2$
...
$n_i$
$n_1$
$n_2$
...
$n_i$
time
$t_p$
Command packet generated
(a)
Transition delay: $(N-1) \times D_{asc}$
Last node received the command packet
$n_1$ $n_2$ ... $n_i$
$n_1$ $n_2$ ... $n_i$
time
First node received the command packet
$t_p$
Command packet generated
(b)

Fig. 9. Transition delay of periodic association method. (a) Without applying time synchronization, the transition period is $t_p - D_{asc}$. (b) By coordinating the association time of nodes, the transition period is reduced to $(N-1) \times D_{asc}$ in a network including $N$ nodes.

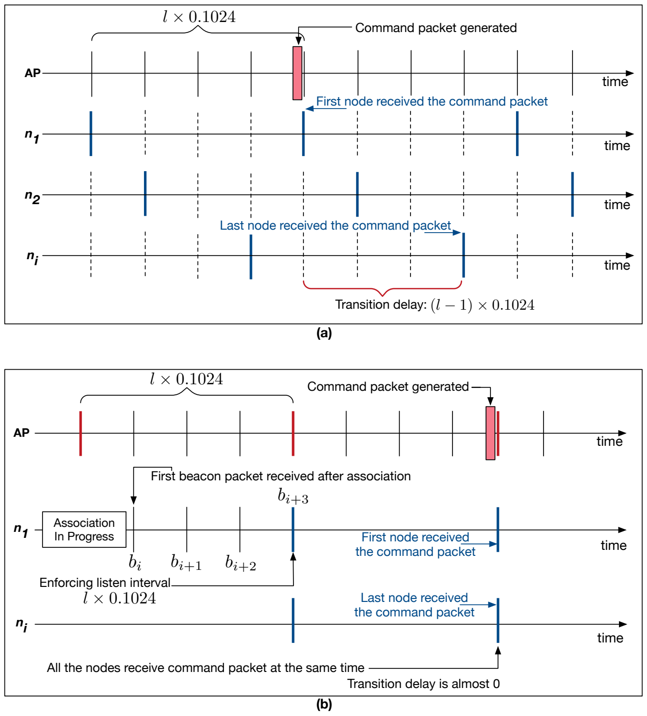
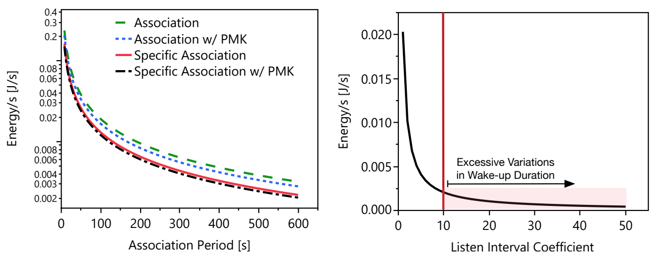

$l \times 0.1024$
Command packet generated
AP
time
First node received the command packet
$n_1$
time
$n_2$
time
Last node received the command packet
$n_i$
time
Transition delay: $(l-1) \times 0.1024$
(a)
$l \times 0.1024$
Command packet generated
AP
time
First beacon packet received after association
$b_{i+3}$
Association In Progress
$n_1$
First node received the command packet
time
$b_i$
$b_{i+1}$
$b_{i+2}$
Enforcing listen interval
$l \times 0.1024$
Last node received the command packet
$n_i$
time
All the nodes receive command packet at the same time
Transition delay is almost 0
(b)

Fig. 10. Transition delay of extended-period beacon reception method. (a) Without synchronizing the wake-up instance of nodes, transition duration is $(l-1) \times 0.1024$. (b) Transition delay reduces to almost zero by coordinating the wake-up time of all the nodes.



(a) Periodic Association (b) Extended-Period Beacon Reception

Fig. 11. Empirical energy evaluation of SoC. These results were collected in an interference-free environment.

Then, the node starts enforcing the listen interval coefficient $l$. For example, in Figure 10(b), after the association of node $n_1$ is complete, this node keeps receiving beacons $b_i$ through $b_{i+3}$. Once beacon $b_{i+3}$ is received, the node enforces the listen interval coefficient $l$. In the worst case scenario where a node completes its association right after receiving beacon $b_i = b_{init} + k \times l$, the node needs to receive $l$ beacons before enforcing listen interval. With this method, the transition delay is negligible.

## C. Energy Comparison

Figure 11 shows the empirical measurement of energy consumption per second by the SoC (CYW54907) for the two proposed methods. For these results, we first measured the duration and energy consumption of various operations, and then used Equations 1 and 2. Since the energy consumed during the association process is high ($P_{asc} = 1.2\,\text{J}$ for Specific Association w/PMK), a long association period ($t_p$) is required to achieve a reasonable level of energy efficiency. With beacon listen interval coefficient $l = 10$, the energy consumed per second is $0.00205\,\text{J}$, as Figure 11(b) shows. The same level of energy efficiency requires $t_p = 593$ when using the Specific Association w/PMK method, confirmed by Figure 11(a).

Although the extended-period beacon reception method seems to outperform the periodic association method, we noticed that as the listen interval coefficient increases, nodes become more sensitive to beacon loss. Specifically, with a larger listen interval coefficient, a beacon loss results in longer energy consumed per beacon reception instance. For example, we noticed that $l > 10$ triggers this behavior with CYW54907. We tried different hardware platforms (BCM4343W, CYW43455, CYW43364) and observed that various transceivers demonstrate such behavior for different listen interval threshold levels. For example, BCM4343W reveals this behavior for $l > 50$. By investigating the 802.11 firmware, we observed that the link quality measurement algorithm requires receiving a certain number of beacons per unit time to measure link quality to the AP. Therefore, in environments where beacon loss may happen due to interference or low link quality, the listen interval coefficient must be carefully adjusted based on the characteristics of the 802.11 transceiver used.

It is worth noting that in this paper, we relied on the energy efficiency methods available by 802.11ac. The subsequent of 802.11ac, known as *802.11ax*, provides a new feature called Target Wake-up Time (TWT), which allows stations to negotiate recurring wake-up instances with the AP. We leave the utilization of this method for achieving energy-latency trade-offs as future work.

## D. Energy Efficiency Parameters

In this section, considering the energy efficiency methods given in §V, we present models for configuring $t_p$ and $l$. These models can be leveraged for the design of various 802.11-based applications.

Considering the scenario given in Figure 1, the following inequality must hold to satisfy the energy-efficiency requirement:

$$\begin{aligned} E_{idl}(t_{idl}) + E_{i2a}(t_{i2a}) + E_{alt}(t_{alt}) + E_{a2s}(t_{a2s}) \\ + E_{smp}(t_{smp}) < E_{bat} \times 80\% \\ \text{Subject to}: \; 1 \le l \le 10 \;\text{ and }\; D_{asc} < t_p \end{aligned} \quad (3)$$

where $E_{idl}(\cdot)$, $E_{i2a}(\cdot)$, $E_{alt}(\cdot)$, $E_{a2s}(\cdot)$, and $E_{smp}(\cdot)$ are functions to compute the energy consumed during the Idle phase, I2A transition, Alert phase, A2S transition, and Sampling phase, respectively. $E_{bat}$ is the total available energy of the battery, and we assume only $80\%$ of the battery capacity can be used. In the rest of this section, we assume $E_{a2s}(t_{a2s}) = 0$ as we showed in §V. Considering the energy efficiency models given in §V, we extend Inequality 3 as follows:

$$\underbrace{t_{idl} \times \Bigg( \Big( P_{asc} \times D_{asc} + P_{off} \times (t_p - D_{asc}) \Big) / t_p \Bigg)}_{\text{(a) Idle phase}} +$$

$$\underbrace{(l \times 0.1024) \times \Bigg( P_{bcn}(\frac{D_{bcn}}{0.1024}) + P_{slp}(1 - \frac{D_{bcn}}{0.1024}) \Bigg)}_{\text{(b) Alert phase: Enforcing Listen Interval } l} +$$

$$\underbrace{\Bigg( (N-1) \times D_{asc} - (l \times 0.1024) \Bigg) \times \Bigg( P_{bcn}(\frac{D_{bcn}}{l \times 0.1024}) +}_{\text{(c) Alert phase: I2A transition of other nodes}}$$

$$\underbrace{P_{slp} \times (1 - \frac{D_{bcn}}{l \times 0.1024}) \Bigg)}_{\text{(c) Alert phase: I2A transition of other nodes (continued)}} +$$

$$\underbrace{(l \times 0.1024) \times \Bigg( P_{bcn} \times (\frac{D_{bcn}}{l \times 0.1024})}_{\text{(d) Alert phase: Enforcing Listen Interval } l \text{ by last node}} +$$

$$\underbrace{P_{slp} \times (1 - \frac{D_{bcn}}{l \times 0.1024})\Bigg)}_{\text{(d) Alert phase: Enforcing Listen Interval } l \text{ by last node (continued)}} +$$

$$\underbrace{t_{smp} \times \bar{E}_{smp}}_{\text{(e) Sampling phase}}$$

$$< 0.8 \times E_{bat}$$

$$\text{Subject to}: \; 1 \leq l \leq 10 \;\text{ and }\; D_{asc} < t_p \qquad (4)$$

We detail this model as follows. We assume the periodic association method is used during Idle phase, and the extended-period beacon reception is used during the Alert phase. This is because the periodic association method can be used to achieve a deeper sleep mode. $E_{idl}(t_{idl}) = t_{idl} \times \bar{E}_{idl}$, where $\bar{E}_{idl}$ is computed in Equation 1, as demonstrated in Equation 4(a). Once the first node transitioned into the Alert phase, it takes $(N-1) \times D_{asc}$ seconds for the rest of the nodes to complete their transition. During this transition duration, the energy consumed by the node can be computed using Equation 2. However, note that the node may need up to $l \times 0.1024$ seconds before enforcing the listen interval $l$, as discussed in §V. The energy consumed in this mode is computed as $(l \times 0.1024) \times \bar{E}_{alt}$, where $\bar{E}_{alt}$ is computed by Equation 2. Note that the value of $l$ in Equation 2 must be 1. This is demonstrated in Equation 4(b). After enforcing the listen interval $l$, the energy consumed during $(N-1) \times D_{asc} - (l \times 0.1024)$ is computed in a similar manner using Equation 2, as demonstrated in Equation 4(c). Once all the nodes transitioned into the Alert phase, the last node that has just transitioned may need up to $l \times 0.1024$ seconds to synchronize with all the nodes' wake-up time. The energy consumed during this duration is demonstrated in Equation 4(d). Finally, $E_{smp}(t_{smp}) = t_{smp} \times \bar{E}_{smp}$, where $\bar{E}_{smp}$ is the energy consumed per second during the Sampling phase. We will present this energy in §IX-B.

## VI. Packet Creation and Protocol Stack Processing

Compared to the 802.15 family of standards, the protocol stack of 802.11 is more complicated and introduces unique challenges in terms of packet processing delay and its effect on the sampling rate and inter-sample interval. In this section, we propose methods to tackle these challenges. We consider three IoT protocol stacks: LwIP [35], NetX [36], and NetXDuo [37]. LwIP is compatible with FreeRTOS, and NetX and NetXDuo work with ThreadX. Although NetX and NetXDuo support UDP and TCP, the major difference is that NetXDuo supports both IPv4 and IPv6 stacks. Considering the higher overhead of TCP, and since packets are transmitted over a single hop, we use UDP. We assume reliability is addressed by other means, such as link-layer retransmissions.

Despite using these operating systems and protocol stacks, the observations and proposed methods can be generalized to various systems, especially those including a single-core application processor.

### A. Packet Processing Delay and its Effect on Sampling Rate

When samples and their timestamps are ready, there are three delay components incurred to transmit a packet: packet preparation, network stack processing, and packet transmission by the transceiver.

Both FreeRTOS [31], [32] and ThreadX [33], [34] require the application thread to create a packet data structure. This process, which we refer to as *packet creation*, entails buffer allocation from a pool available by the network stack. Next, the application payload is sent to the network stack to compute and add proper headers, including TCP/UDP, IP, and MAC. This process is referred to as *packet processing*. Finally, the frame is passed to the wireless subsystem for transmission. Packet creation and packet processing tasks run on the application subsystem, and low-level packet transmission operations (e.g., carrier sensing, backoff) are handled by the wireless subsystem. Therefore, it is essential to reduce the overhead of packet creation and processing to minimize their impact on the sampling task.

Figure 12(a) compares the packet preparation delay of LwIP, NetX, and NetXDuo. The average packet creation delay is around $2.5\,\mu$s for LwIP and $0.5\,\mu$s for NetXDuo and NetX. The main cause of this difference is the memory allocation method used for packet buffers. With LwIP, the packet structure prepared in the application thread is copied to the network stack; whereas, NetX and NetXDuo provide a zero-copy method to prevent this overhead.

Once packet preparation is complete, the `packet_send()` API is called (cf. Algorithm 1). To measure packet processing duration, we modified the network stack to keep track of two events for each packet: (i) when the packet arrives in the network stack and (ii) when the packet is fully processed by the driver and is ready to be sent to the firmware (wireless subsystem). Figure 12(b) demonstrates packet processing delay. In this figure, the name of a protocol stack refers to standard packet processing, including generating UDP, IP, and MAC headers. The results demonstrate that the mean packet processing latency of NetX and NetXDuo is about 65% lower than that of LwIP. One of the main reasons causing the poor performance of LwIP is that it performs multiple validation steps while the packet is passed through the stack. For example, LwIP checks if the socket is valid before further packet processing. Also, both NetX and NetXDuo implement an enhanced UDP processing implementation [36], [37].

To reduce protocol stack processing overhead, we modified NetX and NetXDuo to *bypass UDP checksum generation.* These enhanced stacks are denoted as NetX-NC and NetXDuo-NC, where 'NC' refers to No Checksum. The results in Figure 12(b) show that bypassing UDP checksum

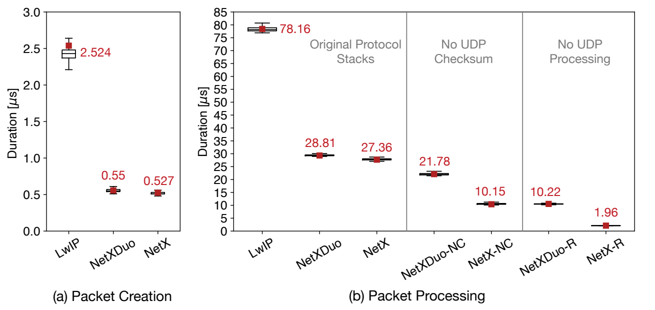


Fig. 12. Empirical evaluation of (a) packet creation, and (b) packet processing duration. The number on each box plot presents the mean value. NetX-NC and NetXDuo-NC refer to bypassing UDP checksum, where 'NC' stands for No Checksum. NetX-R and NetXDuo-R refer to *raw* socket processing, which represents fully bypassing the UDP layer.

reduces the packet processing time of NetX and NetXDuo by 62% and 24%, respectively.

To further reduce packet processing delay, we fully bypass UDP processing. These protocol stacks are called NetX-R and NetXDuo-R, where 'R' stands for *raw*. With this enhancement, NetX-R and NetXDuo-R achieve 1.96 $\mu$s and 10.22 $\mu$s delay, respectively. Considering the execution duration of these network stacks, when collecting 256-sample batches at 500 ksps, the packet processing delay of LwIP, NetX, and NetXDuo result in missing about 39, 13, 14 samples per packet processing, respectively. With NetX-R and NetXDuo-R, the number of missing samples is reduced to 1 and 5 samples per packet processing, respectively. *Considering the lower packet processing delay of NetX, we use this network stack in the rest of this paper.*

## B. Impact on Sampling Stability

In §IV-B we assumed the only task being run by the processor is the sampling task, which is the task responsible for communicating with the ADC and transferring samples to the processor. In this section, we use the term 'Sampling-Only' to refer to this scenario. The results presented in §IV-B demonstrated that even the 'Sampling-Only' scenario results in inter-sample variations. Since in the realistic scenarios the collected samples must be transmitted, in this section, we study how packet preparation, processing, and transmission exacerbate the variations of inter-sample intervals.

In a normal producer-consumer analogy, the sampling task (producer thread) passes the samples to the protocol stack (consumer thread) for processing. Suppose the sampling and packet processing tasks are assigned the same priority level. In that case, the processor switches between the two tasks until the packet is fully processed, and both context switching and packet processing disturb the sampling task. For example, in Figure 3(b), the processor switches to another task before collecting Sample 2. In this case, the interval between the collection of Sample 1 and Sample 2 increases and results in higher differences between $t_2 - t_1$ and $t_3 - t_2$.

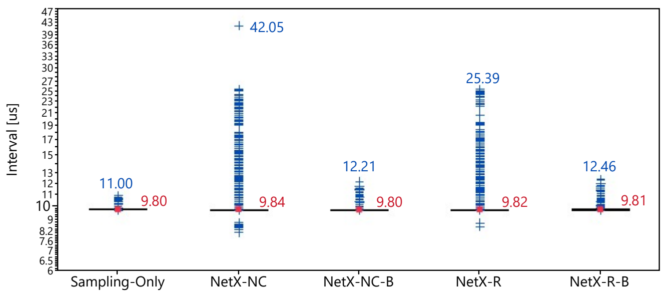


Fig. 13. Variations of inter-sample intervals when performing 60000 iterations of collecting 512-sample batches. The sampling rate is 100 ksps. The numbers on top of the outliers represent the maximum outlier value, and the numbers on the right side of each box plot represent the mean values. *Sampling-Only* refers to the case where no packet processing task runs on the system. NetX-NC and NetX-R refer to *no UDP checksum* and *raw* packet processing, respectively. NetX-NC-B and NetX-R-B refer to the *blocking call* versions of packet processing using NetX-NC and NetX-R, respectively. The blocking call implementations reduce interval variations by up to 70%. Compared to Sampling-Only, the higher variations of NetX-NC-B and NetX-R-B are caused by the communication between driver and firmware.

As Figure 13 demonstrates, the inter-sample intervals achieved by using NetX-NC and NetX-R are up to 42.05 and 25.39 $\mu$s, respectively, which are significantly higher than Sampling-Only results. These results confirm that running packet processing while the sampling task is in progress causes higher variations in inter-sample intervals. An alternative is to assign a higher priority to the sampling task. In this case, however, the high sampling rate and involvement of the processor in sample collection through the SPI bus prevents the allocation of enough resources to packet processing, and this ultimately results in a buffer overflow. The other alternative is to assign a higher priority to the packet processing task. Although, in general, this configuration reduces the inter-sample variations, the packet processing task has the authority to interrupt the sampling task at any time. For example, when an incoming broadcast packet arrives, it causes context switching to the packet processing task, and the utilization of the processor by this task causes interruptions of the sampling task.

To remedy this problem, we modified the NetX stack by converting the `packet_send()` API into a blocking call, while both sampling and packet processing tasks run at the same priority level. The new API returns when the packet passes through all the layers and is processed by the driver. With this method, outgoing packet processing runs sequentially after collecting each batch of samples, and packet processing receives full processor attention. Also, in the case of incoming packets, the processor is shared between the two tasks. These methods are named NetX-NC-B and NetX-R-B in Figure 13, where 'B' refers to *blocking*. As the results show, for NetX-NC-B and NetX-R-B, the maximum inter-sample variation is dropped by 70% and 50%, respectively. Note that converting `packet_send()`

to a blocking call does not mean that packets cannot be buffered. We provisioned a 200-packet buffer in the driver. Once the protocol stack fully processes a packet, the control returns to the sampling task if the buffer is not full. If the buffer is full, the packet processing API waits for the packet buffer to become available before resuming sampling.

Even with using a blocking call, the range of variations is still higher than Sampling-Only. We identified that the driver-firmware interactions cause these outliers. The SoC we used in our node design has two processors, as discussed in §III. The packet exchange between the wireless and application processors is via Direct Memory Access (DMA). Once the driver completes processing a packet, it is sent to the firmware immediately or later. The delay of sending a packet to the firmware is because the firmware, which resides in the wireless subsystem, may be busy sending another packet. Part of initiating a driver-firmware communication through DMA requires attention from the application processor; therefore, the sampling rate is affected. Nevertheless, these variations account for less than 1% of inter-sample intervals.

It is important to note that the inter-sample variations reported in this section are pertaining to the processor type used in our node design. For example, a processor with a lower operating frequency may demonstrate higher variations due to the longer duration of packet processing. Such studies across various hardware platforms are left as future work.

## VII. Time Synchronization

Accurate analysis of the server's collected data requires establishing time synchronization across all the nodes and timestamping all the samples. In the proposed system, we achieve time synchronization by the periodic transmission of a broadcast time synchronization packet from the server to all the nodes. Each broadcast packet is sent once by the AP, and all the nodes receive this packet concurrently. When a node receives this packet, it modifies its internal timer by running a Time Synchronization Method (TSM) that adjusts the node's clock with the one received in the packet. The time synchronization packet's payload size is nine bytes, where eight bytes represent nanosecond time and one byte is used for packet type identification.

Although, in general, establishing time synchronization in single-hop networks is more straightforward than in multi-hop networks, the use of 802.11 standard introduces unique challenges. This section focuses on the delay of processing incoming packets as the leading cause of time synchronization inaccuracy.

### A. Packet Reception Delay

We study how protocol stack layers, from the 802.11 firmware up to the UDP layer, affect the delay of processing an incoming time synchronization packet. We modified these network stacks and placed the TSM in different layers.

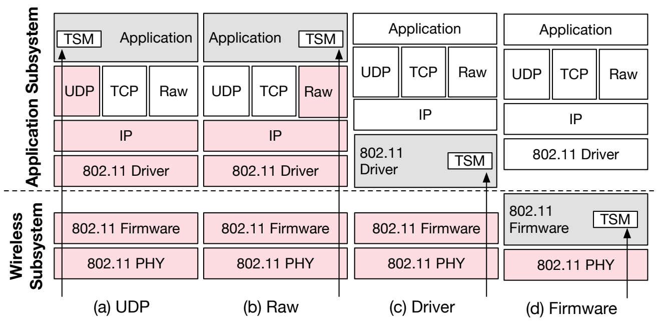


Fig. 14. An incoming broadcast packet used for time synchronization passes through all the shaded layers until it reaches the layer that includes TSM.

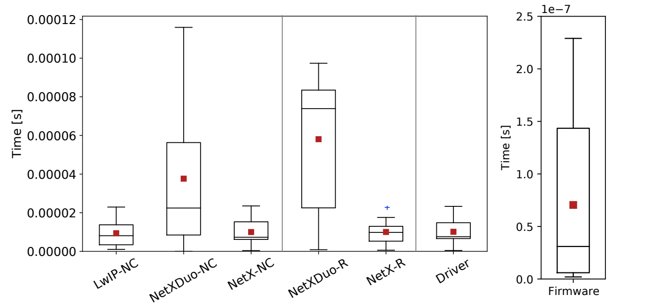


Fig. 15. Empirical evaluation of time synchronization error. When the TSM is implemented in the driver or a layer above it, the worst-case accuracy surpasses 20 $\mu$s. When the TSM is implemented in the firmware, the worst-case accuracy is less than 0.25 $\mu$s. These results confirm that firmware-level time synchronization is required to achieve sub-$\mu$s accuracy.

Figure 14 shows these scenarios. Since the ultimate goal of time synchronization is to ensure all the nodes use the same clock value when adding timestamps to samples, we measured the time difference between two nodes immediately after processing the time synchronization packet. This experiment was repeated for over one million iterations. Figure 15 presents the empirical results. It is important to note that these results present the *difference* between the delay of processing time synchronization packets by the two nodes in our testbed. We detail these results as follows. Bypassing UDP checksum (denoted as 'NC') in LwIP and NetX results in close to 30 $\mu$s error. With NetXDuo, however, the max error is close to 120 $\mu$s. We identified this increase due to the higher processing overhead of NetXDuo caused by its IPv6 supporting feature. The results also show that, although using raw sockets entirely bypasses UDP layer processing, it does not reduce time synchronization error. The same observation is made even when the TSM is placed in the driver. These results indicate that the causes of processing time variability are located before the packet is delivered to the driver. In particular, *once subtracted across the two nodes*, the delay of memory allocation to the packet and processing it in the driver and upper layers is almost negligible compared to the delays caused before the packet arrives in the driver. As discussed in §VI, packet exchange between the wireless and application subsystems is via DMA. With this architecture, time synchronization error is caused by the instance interrupts are raised by the wireless subsystem of each node. Based on this observation, we moved the TSM to the firmware, hence it is run by the wireless subsystem. Thus, time synchronization happens before the received packet is transferred to the application subsystem via DMA. With this enhancement, as Figure 15 shows, the maximum error drops to less than $0.25\,\mu$s. This accuracy is sufficient to maintain time synchronization accuracy for sampling rates up to 1 Msps.

## B. Periodic Time Synchronization

The server must periodically transmit a time synchronization packet to ensure that the nodes' clocks do not deviate beyond a specific limit. Determining the proper period depends on the clock stability of the nodes' processor. The processor's clock frequency stability, denoted as $\nu$, varies based on changes in temperature, voltage, output load, and aging. Frequency stability is typically expressed in parts per million (ppm). Peak frequency variation can be presented as $d_f = f \times \nu$, where $f$ is processor frequency [45]. The maximum time difference between two nodes per second can be expressed as $d_t = \frac{1}{(f-d_f)} - \frac{1}{(f+d_f)}$. The CYW54907 [40] used in our node design runs at $160\,\text{MHz}$ and its $\nu = \pm 2.5\,\text{ppm}$. Therefore, its peak frequency variation is $d_f = 400\,\text{Hz}$, and the maximum timing error per second between two nodes is $d_t = 5\,\mu\text{s}$. The maximum synchronization error immediately after processing the time synchronization packet is less than $0.25\,\mu$s. When sampling at 500 ksps, a time synchronization error of less than $1\,\mu$s allows the server to correlate the samples, which are collected every $2\,\mu$s. To achieve a maximum error of $0.75\,\mu$s, the period of time synchronization must be less than 150 ms. For 100 ksps, a time synchronization period of 1.95 s limits the error to $9.75\,\mu$s.

## VIII. Hardware Design

The primary hardware design considerations are low-power consumption and supporting high sampling rates. Figure 16 presents the prototype and block diagram of a Sensifi node. At a high level, our proposed node design consists of three main units: (i) The SoC is used to collect data from the sensing unit, construct data packets, send packets to an AP, and control the sensing and power management units. (ii) The sensing unit is responsible for measuring acceleration forces and converting analog measurements to digital data. (iii) The power management unit supplies all the components with a stable power source, provides an accurate reference voltage to the sensing unit, measures the node's power consumption, and protects the node from under-voltage discharge. A node includes components (such as a current monitor, battery gas

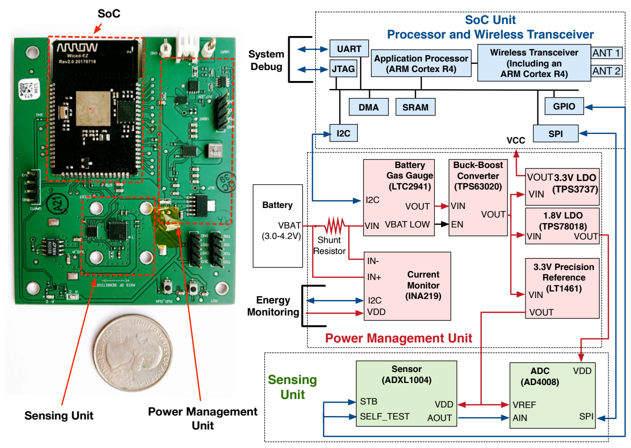


Fig. 16. The (a) prototype and (b) block diagram of a Sensifi node.

gauge, and probe pins) to facilitate performance study and adapt this design for various applications.

Since we have discussed SoC in §III, we focus on the sensing and power management unit in this section. Although this node is geared towards vibration test monitoring, it can be simply modified for various application types.

## A. Sensing Unit

The sensing unit consists of two components, a Micro-Electro-Mechanical Systems (MEMS) accelerometer and an ADC. The details of these components are discussed as follows.

*1) Sensor:* Vibration test requires accelerometers to measure frequencies from 10 Hz to 10 kHz. In addition to accuracy, the sensor must be low-power, low-cost, and available in a small form factor. Considering these requirements, capacitive accelerometers are suitable for measuring static and dynamic accelerations, offering linearity, showing high output stability, and low cost. The sensor used in our node design is ADXL1004 [46], a MEMS capacitive accelerometer with a measurement range of $\pm\,500$ g-force and capable of measuring frequencies of up to 24 kHz. Its maximum power consumption is $5\,\text{mW}$ and its noise range is up to $125\,\mu\text{g}/\sqrt{\text{Hz}}$. Sensor sensitivity is $2.64\,\text{mV/g}$ with an input voltage of 3.3 V. To achieve this sensitivity, we use a precision voltage rail for the sensor's power input to provide a low variance ($\leq\ 1\%$) across temperature and node-to-node.

*2) ADC:* Since the highest sampling frequency measured by the sensor is 24 kHz, according to the Nyquist theorem, the sampling rate must be at least $2 \times 24\,\text{kHz}$. The sensor has $2.64\,\text{mV/g}$ sensitivity with input voltage 3.3 V. Although a 12-bit ADC with 1.6 mV resolution and input voltage of 3.3 V is sufficient to acquire the sensor's output accurately, we use an ADC that provides higher resolution and is capable of sampling at considerably higher rates. With this design, the ADC can interface with different types of sensors and be used in applications with higher demands. Specifically, our design includes a 16-bit AD4008 [47] that supports a maximum sampling rate of 500 ksps. To interface the ADC with the SoC, we developed a custom SPI bus driver. To this end, in addition to the datasheet, we used the FPGA-based evaluation board and the Windows-based software available for this ADC to understand its timing requirements.

The CONV pin of AD4008 is not an interrupt-driven method to detect conversion completion; instead, this pin is used to trigger conversion. Although some ADCs (e.g., ADS8166 and MAX12555) provide a pin to detect sample conversion time, the processor involvement is necessary to send and receive messages over the SPI bus (as discussed in §VI-B). Although some SoCs include ADC and support DMA transmission to the main memory, they do not support 802.11 communication, and therefore it is necessary to transfer the data to another SoC; however, this process introduces additional overhead. We leave the design, development, and potential benefits of such node architecture as future work.

To prevent aliasing and non-linear charge kickback at the start of each ADC acquisition phase, we placed a first order anti-aliasing RC filter at the ADC's input. In addition, AD4008 provides a high-z mode feature that mitigates the effects of non-linear charge kickback and reduces distortion over a wide frequency range up to 100 kHz.

## B. Power Management Unit

The power management unit must be capable of operating over a wide load profile. The gas gauge is used to protect the battery (lithium-polymer) from under-voltage discharge and monitor the battery's remaining charge. The bucket-boost converter (TPS63020) is part of the node's main voltage rail and feeds into the cascaded linear regulators. By simulating performance across the design input voltage range and load currents up to 500 mA, the bucket-boost converter shows an efficiency range between $88\%$ at light current load and up to $94.5\%$ for the max current load [48]. However, the bucket-boost converter introduces a switching noise that affects other sensitive analog components when the performance efficiency reaches 90%. To reduce the noise and provide a steady voltage output, our design uses TPS3737 [49] 3.3 V and TPS78018 [50] 1.8 V low-dropout (LDO) regulators. The 3.3 V regulator provides a stable voltage for the processor and wireless transceiver unit, and the 1.8 V regulator provides a stable voltage for the ADC. Since AD4008 and ADXL1004 only consume a maximum of 2 mA during sampling, the impact of dissipated power loss for the 1.8 V LDO regulator is negligible.

Our design uses a LT1461 [51] to provide an extremely constant voltage reference that is resistant to temperature, noise on the supply rail, and process variations [52]. We connect the LT1461 to the ADXL1004 and AD4008 to prevent the risk of switching noise or voltage ripple, thereby

TABLE I
CURRENT CONSUMPTION BY THE POWER MANAGEMENT UNIT

| | Active Component's Current (mA) | | | | | Total Current (mA) |
|---|---|---|---|---|---|---|
| | Battery Gas Gauge | Buck Boost Converter | 3.3V LDO | 1.8V LDO | 3.3V Ref | |
| Idle\Alert | 0.1 | 0.105 | 0.297 | - | - | 0.502 |
| Sampling | 0.1 | 0.105 | 0.297 | 0.872 | 0.07 | 1.444 |

reducing measurement inaccuracies. It can also deliver up to 50 mA, and the output voltage is guaranteed to be within $\pm 0.08\%$ of the nominal value with a temperature drift of only $12\ \mathrm{ppm}/^{\circ}\mathrm{C}$. For developmental purposes and to eliminate the need to use an external power meter, an INA219 [53] is used for in-circuit power measurement.

We used a high-precision Digital Multimeter (DMM) [54] to study the power consumed by the components of the power measurement unit. Table I presents the results. During the Idle phase and Alert phase, both the ADXL1004 and AD4008 are completely turned off, and only the SoC needs to be powered. Table I shows that the current consumption of the power management unit during these phases is $502\ \mu\mathrm{A}$. The sensor and ADC must be turned on during the Sampling phase. Due to the activation of the 1.8 V LDO and the 3.3 V reference voltage, the power consumption of the power management unit increases to 1.444 mA. We present the overall energy consumption of a node during the Sampling phase in §IX-B.

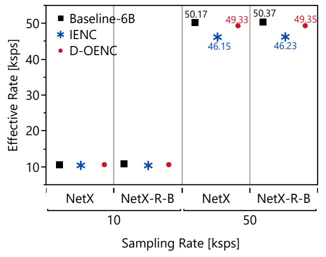
## IX. OVERALL SYSTEM EVALUATION

In this section, we evaluate the overall performance of the proposed system in terms of sampling rate, data generation rate, and energy efficiency.

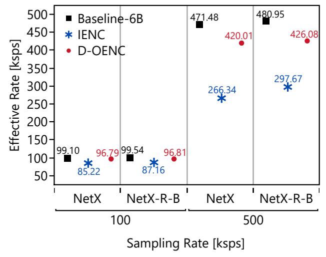
## A. Effective Sampling Rate and Data Rate

During the Sampling phase, each node switches between sampling, packet encoding, packet creation, and packet processing. To quantify the processing overhead of these tasks on the sampling rate, we define *effective rate* as the actual number of samples collected and transmitted per second by a node. Figure 17 shows the empirical evaluation of the effective rate. On the other hand, to quantify the effectiveness of the encoding methods, Figure 18 presents the empirical measurement of wireless communication rate.

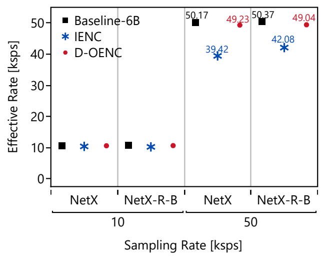
Both Figures 17 and 18 include Baseline-6B, which represents the case where no encoding is applied on timestamps (§IV-B). Note that the number of samples per packet when using Baseline-6B is always 183, independent of batch size. As Figure 17 shows, we observe that the sampling rate achieved by Baseline-6B is closer to the target rate, compared to the cases where encoding is applied. This is because packet encoding introduces a processing time that causes sampling interruption. However, as Figure 18 shows, the



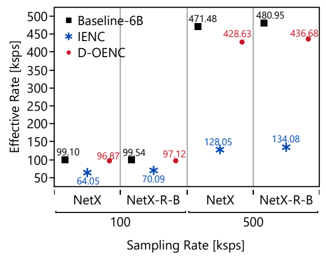
(a) 256-Sample Batch



(b) 256-Sample Batch

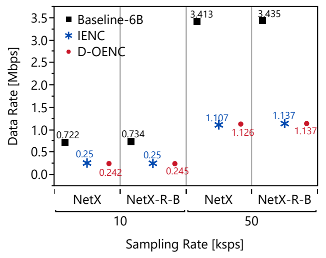


(c) 512-Sample Batch

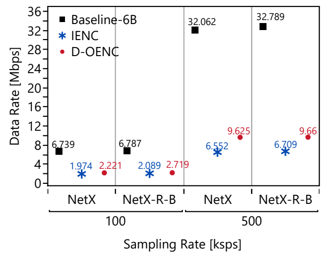


(d) 512-Sample Batch

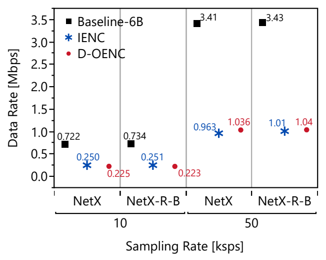
Fig. 17. Empirical measurement of *effective rate* for (a) and (b): 256-sample batches, and (c) and (d): 512-sample batches. The effective rate is the actual number of samples collected per second by a node. Note that the number of samples per packet when using Baseline-6B is always 183, independent of batch size. NetX refers to the default protocol stack, and NetX-R-B refers to the blocking call implementation of raw packet processing.



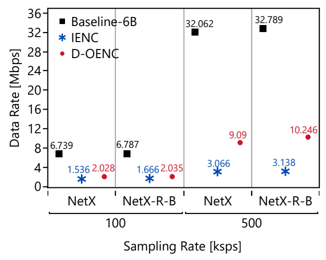
(a) 256-Sample Batch



(b) 256-Sample Batch



(c) 512-Sample Batch



(d) 512-Sample Batch

Fig. 18. Empirical measurement of wireless communication rate. The data generated per second by a node includes samples, timestamps, packet encoding data, and packet headers. Note that the number of samples per packet when using Baseline-6B is always 183, independent of batch size.

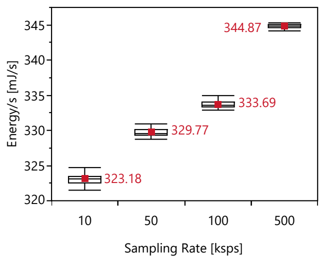
data rate generated per node is considerably higher when using Baseline-6B. For example, when sampling at 500 ksps, the sampling rate of D-OENC is 10% lower than that of Baseline-6B (Figure 17(d)), whereas, the data rate generated by Baseline-6B is 220% higher (Figure 18(d)). Regardless of the channel access mechanism, the significantly higher data rate of Baseline-6B increases the number of APs required to ensure the bandwidth and reliability requirements of communication are met. Also, note that for sampling rates less than 500 ksps, the effective rate achieved by D-OENC is almost the same as the target rate, but the amount of data generated by Baseline-6B is considerably higher.

As the sampling rate increases, the duration required to collect a batch drops, but the overhead of encoding and packet processing remains unchanged; thereby, the effective rate drops as the sampling rate increases. With D-OENC, although the results in §IV-D demonstrated a 147% increase in processing time when switching from 256- to 512-sample batch, the results show almost similar performance for the two batch sizes. There are two reasons for this observation: First, packet processing overhead halves when the batch size doubles. Second, due to the SPI bus initialization time, the time it takes to collect 512 samples consecutively is less than collecting this number of samples in two 256-sample batches. In summary, by using D-OENC and NetX-R-B, we are able to collect up to 436 ksps when the ADC operates at 500 ksps, as Figure 17(d) shows. With IENC, however, we observe a considerably lower effective rate when comparing the smaller and larger batch sizes, i.e., Figures 17(a) and (b) against (c) and (d). Based on the results presented in §IV-D, increasing the batch size from 256 to 512 exhibits a 376% increase in the execution time of IENC. This significant increase in encoding time dominates the overheads caused by packet processing and SPI bus initialization.

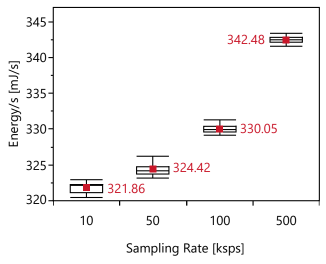
These results also show the effect of network stack on both effective rate and data rate. Compared to NetX, using NetX-R-B results in faster packet processing, thereby increasing the effective rate. On the other hand, since NetX-R-B bypasses UDP processing, the packets do not include the UDP header, hence reducing the amount of data generated per second. For example, when sampling at 500 ksps, Figure 17(b) shows that using D-OENC with NetX-R-B increases the number of samples collected per second by about 6000, compared to NetX. This comes at 0.025 Mbps increase in data rate, as Figure 18(b) demonstrates. With IENC, Figure 18(b) shows that sampling rate increases by about 31000, and this causes 0.157 Mbps increase in data rate.

## *B. Energy Consumption During Sampling Phase*

This section presents the energy consumption of a node that uses D-OENC and NetX-R-B during the Sampling phase. Figure 19 shows the empirical measurement of energy consumption for various sampling rates. As we increase the sampling rate from 10 ksps to 500 ksps, the energy consumed per second increases by 6.7% and 6.4% when using 256- and 512-sample batches, respectively. The higher energy consumption when using 256-sample batches is due to the higher packet header overhead caused by sending more



(a) 256-Sample Batch



(b) 512-Sample Batch

Fig. 19. Empirical measurement of energy consumption during the Sampling phase. While the effective rate of the 512-sample batch size is higher (Figure 17), its energy consumption is lower because of the lesser overhead of packet processing and transmission.

packets. Also, with the smaller batch size, packet processing overhead increases and the number of switching between reception and transmission mode doubles.

## X. Related Work

The existing SHM systems are built on top of the 802.15 standards to reduce installation and maintenance costs for structural monitoring [24], [55], [56], machine health monitoring [3], [57], space applications [58], [59], and natural hazard monitoring [4], [60]. These standards offer low energy consumption, low cost, ease of integration with various microprocessors, and the simplicity of configuring and customizing physical layer and MAC parameters [13], [24], [59], [61], [62]. However, these systems do not support ultra-high-rate applications; instead, they either focus on the networking and energy efficiency aspects of multi-hop communication or propose methods for distributed processing and data aggregation [25]. Although using distributed processing can reduce communication rate demand, not all applications can benefit from this method. The applications that require long sampling duration or raw sample collection across nodes (similar to that discussed in §II) require centralized processing to enhance analysis accuracy and reduce the processing burden of nodes [20], [63].

Abedi et al. [64] showed that at the physical layer, the energy consumption of 802.11 is 10 - 100 nJ/bit, while the range is 275 - 300 nJ/bit for Bluetooth Low Energy (BLE). They also argued that the overall energy consumption of 802.11 is higher than BLE due to the overheads pertaining to association and maintaining association. The benefits of using 802.11 for building high-rate, energy-efficient WSNs have been studied recently. Tramarin et al. [65] evaluated the suitability of 802.11n in industrial applications from its physical layer point of view. Luvisotto et al. [10] provided an overview of the requirements of industrial wireless networks and studied the suitability of 802.11 to address these requirements, compared to cellular and 802.15 networks.

Yu et al. [66] proposed an 802.11-based WSN to monitor potential damages to bridge parts while they are being lifted and mounted. The sampling rate of each node is 100 sps, each packet contains 80 samples, and the size of each sample is 16 bits. RT-WiFi [26], [28] can sample at up to 6 ksps. The basic idea is to modify the driver and mac80211 module on a Linux machine and enforce transmission schedules received from a central controller. They also evaluated system performance in a gait rehabilitation application where a mobile assistive robot and a smart shoe collect and transmit samples at 100 sps and 1 ksps, respectively, and the controller replies at 100 sps. Their work, however, does not address the challenges pertaining to scalability, high-rate transmission, and energy efficiency. Dombrowski and Gross [67] proposed a distributed and deterministic channel access method to enhance the reliability of using token-ring for channel access arbitration. They utilize software-defined radios operating in a 10 MHz channel. The sampling rate of this solution is 20 sps. Khorov et al. [68] presented the periodic reservation methods available in various 802.11 standards including 802.11s, 802.11ad, 802.11ax, and 802.11be. They also addressed the challenges of establishing communication reservation periods for exchanging real-time multimedia traffic between two stations in the presence of interference. The use of 802.11 standards in real-time multimedia applications has been studied in [69] and [70]. Seno et al. [27] proposed a centrally-controlled 802.11 network to decide the admission of nodes that require real-time communication. They use an Earliest Deadline First (EDF) scheduling method to admit nodes into the network, subject to their deadline and retransmission requirements. Their proposed method has been implemented on the Linux operating system and the ath9k driver. They present a thorough analysis of reliability; however, there is no study of the sampling rate. Bartolomeu et al. [71] addressed the challenges of transmitting real-time packets in the presence of non-real-time traffic.

In summary, the sampling rates achieved by the aforementioned works are far below the requirements of ultra-high-rate applications. Also, in contrast to our work, the use of extended-period beacon reception and periodic association has not been studied in the existing works. In particular, existing studies are primarily focused on reducing communication overhead in smartphones, and none of them propose methods for applications with long inactivity periods.

In this paper, we underlined the importance of time synchronization for accurate sample timestamping. Achieving accurate time synchronization across nodes is also essential for TDMA-based channel access and real-time communication, and therefore, it has been studied by existing works [26], [28], [72]–[75]. IEEE 802.11 specifies a Time Synchronization Function (TSF) to time synchronize the wake-up instances of stations for beacon reception. However, as per industrial implementations, the accuracy of this method is within 25 $\mu$s; thereby, it cannot be used in high-rate applications. Sevani et al. [76] utilized Madwifi driver (for Linux) and employed hardware packet timestamping to achieve a synchronization error of 10 $\mu$s. Uchimura et al. [72] considered using TSF for time synchronization in adhoc 802.11 networks. They analyzed the effect of channel access contention and beacon collision on accuracy. Their empirical results using RT-Linux show the clock accuracy of 27 $\mu$s. Wei et al. [26] modified the ath9k driver on Linux to utilize TSF for achieving time synchronization in a single-hop network. Assuming beacon transmission every 100 ms, they consider a maximum clock drift of 20 $\mu$s. To achieve nanosecond precision synchronization, Seijo et al. [77] proposed a timestamping method that, in particular, takes into account the packet arrival rate caused by the multipath effect. They utilize simulation-based studies to evaluate the proposed method. In short, the existing time synchronization methods (e.g., 20 $\mu$s in [75] and [26], 27 $\mu$s in [72], and 10 $\mu$s in [76]) do not provide the accuracy required for sample timestamping in ultra-high-rate applications.

Although modification of the 802.11 stack and bypassing layer-3 and layer-4 processing are the most common approaches employed to develop 802.11-based WSNs and enhance time synchronization accuracy, there is no study concerning the effect of packet processing on sampling jitter. Furthermore, in contrast with the existing work that utilize the Linux operating system (e.g., [26]–[28], [72], [73], [78]), in this paper we utilized RTOSes and lightweight protocol stacks primarily designed for IoT applications.

Some WSN designs employ non-live data collection to cope with the low transmission rate of 802.15.4 links or reduce the effect of wireless transmission on the sampling rate. For example, in [20], each node stores the collected samples in its local storage, and then the saved data is sent over 802.15.4 links. The impact of writing to flash storage on the stability of sampling rate was studied in [20]. They demonstrated that when sampling at 5 ksps, the sampling jitter is increased by 10 $\mu$s. They argued that a multi-processor node design is required to tackle this challenge. Dai et al. [79] showed that the process of sample collection and writing to a memory card reduces the sampling rate. To address this problem, they used the DMA controller to directly transfer data from ADC to SRAM. The user can then request for wireless transfer of the collected samples. Although they show the enhancements in terms of the processor cycles saved, no evaluation of the actual sampling rate was presented. Phanish et al. [61] also used DMA to enhance sampling rate stability. Specifically, each sampling round collects and transfers 16 samples to SRAM via DMA. Then, each batch of samples is sent as one packet. Compared to our work, none of the existing works studied the effect of packet preparation and transmission on sampling rate.

In addition to non-live data collection, various compression methods have been used to reduce the amount of data transmitted per node. One common approach for time-series-based data is to compute auto-regressive/auto-regressive moving average (AR/ARMA) coefficients locally by each node and transmit this data instead of the raw sensor data [80], [81], or use AR methods with random decrements to compress sensor data by averaging a large number of time segments [82]. Nevertheless, these approaches are lossy and cannot be used in scenarios where raw data collection is required to implement various data analysis methods [7], [23]. Furthermore, even if a drop in monitoring accuracy is tolerated, the compression level provided by these methods is not enough to adapt low-rate wireless technologies for ultra-high-rate applications. Finally, the high processing complexity of these compression algorithms directly affects the sampling rate. Another approach is to process data locally and send samples only if a particular event is detected [23], [30].

## XI. Conclusion

This paper presented Sensifi, a system based on the 802.11 (WiFi) standard for ultra-high-rate sensing applications. Our main observations and contributions are as follows: (i) We showed that timestamp encoding is necessary to reduce the overhead of data transmission. We proposed a lossless encoding method that achieves a high compression rate and low processing overhead. (ii) Considering the unique operational phases of the 802.11 standard, we proposed methods to enhance network longevity. In particular, we showed that periodic reassociation with the AP may be preferred over long beacon reception intervals. (iii) We profiled the overhead of various network stacks in terms of outgoing packet preparation, processing, and their impact on sampling rate and stability. We significantly reduced packet processing overhead by bypassing the UDP layer. Also, by implementing a blocking packet processing call, we minimized the effect of packet processing on sampling stability. (iv) We studied the delay of processing time synchronization packets from firmware to the application layer, and we proved that firmware-based processing is necessary to achieve a sub-$\mu$s accuracy. (v) We proposed a low-power node design for ultra-high-rate applications and presented empirical micro-benchmarks and overall system evaluation results using this node design. Our work leveraged the two widely used RTOSes (FreeRTOS and ThreadX) and three low-power network stacks (NetX, NetXDuo, and LwIP).

Although Sensifi provides live data transmission, no per-packet delivery delay is enforced. To enable the adoption of the proposed work in applications where real-time data delivery is a requirement [1], an area of future work is to establish a time-slotted channel access scheduling among nodes. In this work, we enhanced scalability by reducing the amount of data generated per node. An area of future work is to evaluate system scalability and communication reliability using a large testbed and considering various channel access methods. Compared to 802.11ac used in this paper, the 802.11ax standard offers OFDMA, which can be leveraged for the allocation of sub-carriers to nodes based on their buffered data. In addition, MU-MIMO allows the

AP to concurrently receive and decode signals received from multiple nodes. Both of these methods enhance the packet delivery rate and reduce the energy consumption of nodes. Such evaluations require a large-scale testbed and alternative hardware platforms. Novel methods must also be developed for systems that include more than one AP. For example, managing the association point of each node can be leveraged to reduce the number of retransmissions and enhance system reliability.

## References

[1] B. Dezfouli, M. Radi, and O. Chipara, "Rewimo: A real-time and reliable low-power wireless mobile network," *ACM Transactions on Sensor Networks (TOSN)*, vol. 13, no. 3, pp. 1–42, 2017.

[2] J. K. Notay and G. A. Safdar, "A wireless sensor network based structural health monitoring system for an airplane," in *Proceedings of ICAC*. IEEE, 2011, pp. 240–245.

[3] Q. Huang, B. Tang, and L. Deng, "Development of high synchronous acquisition accuracy wireless sensor network for machine vibration monitoring," *Measurement*, vol. 66, pp. 35–44, 2015.

[4] L. Girard, S. Gruber, S. Weber, and J. Beutel, "Environmental controls of frost cracking revealed through in situ acoustic emission measurements in steep bedrock," *Geophysical Research Letters*, vol. 40, no. 9, pp. 1748–1753, 2013.

[5] T. H. Loutas, A. Panopoulou, D. Roulias, and V. Kostopoulos, "Intelligent health monitoring of aerospace composite structures based on dynamic strain measurements," *Expert Systems with Applications*, vol. 39, no. 9, pp. 8412–8422, 2012.

[6] S. Wijetunge, U. Gunawardana, and R. Liyanapathirana, "Wireless sensor networks for structural health monitoring: Considerations for communication protocol design," in *Proceedings of ICT*. IEEE, 2010, pp. 694–699.

[7] X. Liu, J. Cao, and P. Guo, "SenetSHM: Towards practical structural health monitoring using intelligent sensor networks," in *Big Data Cloud Comput. Soc. Comput. Netw. Sustain. Comput. Commun.* IEEE, 2016, pp. 416–423.

[8] T. Harms, S. Sedigh, and F. Bastianini, "Structural health monitoring of bridges using wireless sensor networks," *IEEE Instrumentation and Measurement*, vol. 13, no. 6, pp. 14–18, 2010.

[9] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *Proceedings of SenSys*, 2004, pp. 13–24.

[10] M. Luvisotto, Z. Pang, and D. Dzung, "Ultra high performance wireless control for critical applications: Challenges and directions," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1448–1459, 2017.

[11] P. M. Fantasia. (2011) Vibration and Acoustic Test Facility (VATF): User Test Planning Guide. [Online]. Available: https://www.nasa.gov/centers/johnson/pdf/639503main_VATF_User_Test_Planning_Guide.pdf

[12] M. Lindstrom. (2019) What's wrong with my piezoelectric accelerometer? [Online]. Available: http://www.pcb.com/Contentstore/MktgContent/LinkedDocuments/Technotes/TN-30-what's_wrong_Lowres.pdf

[13] B. Dezfouli, M. Radi, S. Abd Razak, T. Hwee-Pink, and K. A. Bakar, "Modeling low-power wireless communications," *Journal of Network and Computer Applications*, vol. 51, pp. 102–126, 2015.

[14] FieldComm Group. (2020) HART Communication Protocol Specification, Revision 7.7. [Online]. Available: https://library.fieldcommgroup.org/20013/TS20013/7.7/#page=1

[15] IEC. (2015) Industrial networks - Wireless communication network and communication profiles - WIA-PA. [Online]. Available: https://webstore.iec.ch/publication/23902

[16] ISA 100 Wireless. (2020) ISA 100 Wireless. [Online]. Available: https://isa100wci.org

[17] R. Heynicke, D. Krush, G. Scholl, B. Kaercher, J. Ritter, P. Gaggero, and M. Rentschler, "IO-link wireless enhanced sensors and actuators for industry 4.0 networks," in *Proceedings of AMA*, 2017, pp. 134–138.

[18] G. Scheible, D. Dzung, J. Endresen, and J. E. Frey, "Unplugged but connected [design and implementation of a truly wireless real-time sensor/actuator interface]," *IEEE Industrial Electronics Magazine*, vol. 1, no. 2, pp. 25–34, 2007.

[19] S. Valenti, M. Conti, P. Pierleoni, L. Zappelli, A. Belli, F. Gara, S. Carbonari, and M. Regni, "A low cost wireless sensor node for building monitoring," in *Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*. IEEE, 2018, pp. 1–6.

[20] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of IPSN*, 2007, pp. 254–263.

[21] A. Araujo, J. García-Palacios, J. Blesa, F. Tirado, E. Romero, A. Samartín, and O. Nieto-Taladriz, "Wireless measurement system for structural health monitoring with high time-synchronization accuracy," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 3, pp. 801–810, 2011.

[22] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proceedings of SenSys*, 2003, pp. 14–27.

[23] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[24] M. J. Whelan, M. V. Gangone, K. D. Janoyan, and R. Jha, "Real-time wireless vibration monitoring for operational modal analysis of an integral abutment highway bridge," *Engineering Structures*, vol. 31, no. 10, pp. 2224–2235, 2009.

[25] M. M. Alves, L. Pirmez, S. Rossetto, F. C. Delicato, C. M. de Farias, P. F. Pires, I. L. dos Santos, and A. Y. Zomaya, "Damage prediction for wind turbines using wireless sensor and actuator networks," *Journal of Network and Computer Applications*, vol. 80, pp. 123–140, 2017.

[26] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-time high-speed communication protocol for wireless cyber-physical control applications," in *Proceedings of RTSS*, 2013, pp. 140–149.

[27] L. Seno, G. Cena, S. Scanzio, A. Valenzano, and C. Zunino, "Enhancing communication determinism in Wi-Fi networks for soft real-time industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 866–876, 2016.

[28] F. Tramarin, A. K. Mok, and S. Han, "Real-time and reliable industrial control over wireless lans: Algorithms, protocols, and future directions," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1027–1052, 2019.

[29] M. Islam, N. N. Soualihou, and S. Faizullah, "Structural health monitoring by payload compression in wireless sensors network: An algorithmic analysis," *International Journal of Engineering and Management Research (IJEMR)*, vol. 8, no. 3, pp. 184–190, 2018.

[30] Q. Ling, Z. Tian, Y. Yin, and Y. Li, "Localized structural health monitoring using energy-efficient wireless sensor networks," *IEEE Sensors Journal*, vol. 9, no. 11, pp. 1596–1604, 2009.

[31] Amazon Inc. (2020) FreeRTOS: Real-time operating system for microcontrollers. [Online]. Available: https://www.freertos.org

[32] Amazon Inc. (2020) FreeRTOS AWS Reference Integrations. [Online]. Available: https://github.com/aws/amazon-freertos

[33] Microsoft Inc. (2020) Azure RTOS ThreadX documentation. [Online]. Available: https://docs.microsoft.com/en-us/azure/rtos/threadx/

[34] Microsoft Inc. (2020) Azure RTOS ThreadX. [Online]. Available: https://github.com/azure-rtos/threadx

[35] Free Software Foundation Inc. (2020) LwIP: A Lightweight TCP/IP stack. [Online]. Available: https://savannah.nongnu.org/projects/lwip/

[36] Microsoft Inc. (2020) Azure RTOS NetX documentation. [Online]. Available: https://docs.microsoft.com/en-us/azure/rtos/netx/

[37] Microsoft Inc. (2020) Azure RTOS NetX Duo documentation. [Online]. Available: https://docs.microsoft.com/en-us/azure/rtos/netx-duo/

[38] K. Chang, "Deep space 1 spacecraft vibration qualification testing," *Institute of Environmental Sciences and Technology, 46th Annual Technical Meeting*, 2000.

[39] A. R. Kolaini, W. Tsuha, and J. P. Fernandez, "Spacecraft vibration testing: Benefits and potential issues," *Advances in Aircraft and Spacecraft Science*, vol. 5, no. 2, p. 165, 2018.

[40] Cypress Semiconductor Corporation. (2020) CYW54907 - WICED™ IEEE 802.11 a/b/g/n/ac SoC with an Embedded Applications

Processor. [Online]. Available: https://www.cypress.com/file/407801/download

[41] Y. Zhai, J. Li, C. Ding, S. Luan, and M. Jin, "A Sample-and-Hold Circuit for a Resolution Pipelined ADC," in *Proceedings of ICEESD*. Atlantis Press, 2018.

[42] K. C. Barr and K. Asanović, "Energy-aware lossless data compression," *Transactions on Computer Systems (TOCS)*, vol. 24, no. 3, pp. 250–291, 2006.

[43] V. Pankratius, A. Jannesari, and W. F. Tichy, "Parallelizing bzip2: A case study in multicore software engineering," *IEEE software*, vol. 26, no. 6, pp. 70–77, 2009.

[44] A. Habib, M. J. Islam, and M. S. Rahman, "A dictionary-based text compression technique using quaternary code," *Iran Journal of Computer Science*, pp. 1–10, 2019.

[45] F. Tirado-Andrés and A. Araujo, "Performance of clock sources and their influence on time synchronization in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 15, no. 9, p. 1550147719879372, 2019.

[46] Analog Devices Inc. (2018) ADXL1004 MEMS Accelerometer. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/adxl1004.pdf

[47] Analog Devices Inc. (2017) AD4000/AD4004/AD4008 16-Bit, 2 M/1 M/500 ksps, Precision, Pseudo Differential, SAR ADCs. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD4000-4004-4008.pdf

[48] Texas Instruments. (2019) TPS6302x buck-boost converter. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps63020.pdf

[49] Texas Instruments. (2015) TPS737xx 1 A Low-Dropout Regulator. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps737.pdf

[50] Texas Instruments. (2015) TPS780xx 150 mA Low-Dropout Regulator. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps780.pdf

[51] Linear Technology. (1999) LT1461 Voltage Reference. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/1461fb.pdf

[52] Texas Instruments. (2018) Tips and tricks for designing with voltage references. [Online]. Available: http://www.ti.com/lit/ml/slyc147/slyc147.pdf

[53] B. Dezfouli, I. Amirtharaj, and C.-C. C. Li, "Empiot: An energy measurement platform for wireless iot devices," *Journal of Network and Computer Applications*, vol. 121, pp. 135–148, 2018.

[54] Tektronix, Inc. (2017) DMM7510 $7\frac{1}{2}$-Digit Graphical Sampling Multimeter. [Online]. Available: https://www.tek.com/tektronix-and-keithley-digital-multimeter/dmm7510

[55] J. M. Samuels, M. Reyer, S. Hurlebaus, S. H. Lucy, D. G. Woodcock, and J. M. Bracci, "Wireless sensor network to monitor an historic structure under rehabilitation," *Journal of Civil Structural Health Monitoring*, vol. 1, no. 3-4, pp. 69–78, 2011.

[56] J. P. Lynch, K. H. Law, A. S. Kiremidjian, E. Carryer, C. R. Farrar, H. Sohn, D. W. Allen, B. Nadler, and J. R. Wait, "Design and performance validation of a wireless sensing unit for structural monitoring applications," *Structural Engineering and Mechanics*, vol. 17, no. 3-4, pp. 393–408, 2004.

[57] B. Bengherbia, M. O. Zmirli, A. Toubal, and A. Guessoum, "FPGA-based wireless sensor nodes for vibration monitoring system and fault diagnosis," *Measurement*, vol. 101, pp. 81–92, 2017.

[58] L. Swathy and L. Abraham, "Analysis of vibration and acoustic sensors for machine health monitoring and its wireless implementation for low cost space applications," in *Proceedings of ICCSC*. IEEE, 2014, pp. 80–85.

[59] J. Dodson and A. Eglin, "High-g Shocking Testing of the Martlet Wireless Sensing System," *Sound & Vibration*, p. 6, 2018.

[60] S. Weber, S. Gruber, L. Girard, J. Beutel, and K. M. Hinkel, "Design of a measurement assembly to study in situ rock damage driven by freezings," in *Proceedings of ICOP*. University of Zurich, 2012, pp. 437–442.

[61] D. Phanish, P. Garver, G. Matalkah, T. Landes, F. Shen, J. Dumond, R. Abler, D. Zhu, X. Dong, Y. Wang *et al.*, "A wireless sensor network for monitoring the structural health of a football stadium," in *Proceedings WF-IoT*. IEEE, 2015, pp. 471–477.

[62] B. Dezfouli, M. Radi, K. Whitehouse, S. A. Razak, and H.-P. Tan, "Cama: Efficient modeling of the capture effect for low-power wireless networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 11, no. 1, pp. 1–43, 2014.

[63] K. Mechitov, W. Kim, G. Agha, and T. Nagayama, "High-frequency distributed sensing for structure monitoring," in *Proceedings of INSS*, 2004, pp. 101–105.

[64] A. Abedi, O. Abari, and T. Brecht, "Wi-le: Can wifi replace bluetooth?" in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, 2019, pp. 117–124.

[65] F. Tramarin, S. Vitturi, M. Luvisotto, and A. Zanella, "On the use of IEEE 802.11n for industrial communications," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1877–1886, 2015.

[66] Y. Yu, F. Han, Y. Bao, and J. Ou, "A study on data loss compensation of wifi-based wireless sensor networks for structural health monitoring," *IEEE Sensors Journal*, vol. 16, no. 10, pp. 3811–3818, 2015.

[67] C. Dombrowski and J. Gross, "EchoRing: A low-latency, reliable token-passing MAC protocol for wireless industrial networks," in *Proceedings of European Wireless Conference*. VDE, 2015, pp. 1–8.

[68] E. Khorov, A. Lyakhov, A. Ivanov, and I. F. Akyildiz, "Modeling of Real-Time Multimedia Streaming in Wi-Fi Networks With Periodic Reservations," *IEEE Access*, vol. 8, pp. 55 633–55 653, 2020.

[69] J. S. Blanes, J. Berenguer-Sebastiá, V. Sempere-Paya, and D. T. Ferrandis, "802.11n performance analysis for a real multimedia industrial application," *Computers in Industry*, vol. 66, pp. 31–40, 2015.

[70] S. Santonja-Climent, D. Todoli-Ferrandis, T. Albero-Albero, V.-M. Sempere-Payá, J. Silvestre-Blanes, and J. Alcober, "Analysis of control and multimedia real-time traffic over SIP and RTP on 802.11n wireless links for utilities networks," in *Proceedings of ETFA*. IEEE, 2010, pp. 1–4.

[71] P. Bartolomeu, M. Alam, J. Ferreira, and J. A. Fonseca, "Supporting deterministic wireless communications in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4045–4054, 2018.

[72] Y. Uchimura, T. Nasu, and M. Takahashi, "IEEE 802.11-based wireless sensor system for vibration measurement," *Advances in Civil Engineering*, vol. 2010, 2010.

[73] 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, 2016.

[74] J.-P. Sheu, C.-M. Chao, W.-K. Hu, and C.-W. Sun, "A clock synchronization algorithm for multihop wireless ad hoc networks," *Wireless Personal Communications*, vol. 43, no. 2, pp. 185–200, 2007.

[75] D. Zhou and T.-H. Lai, "A compatible and scalable clock synchronization protocol in IEEE 802.11 ad hoc networks," in *Proceedings of ICPP*. IEEE, 2005, pp. 295–302.

[76] V. Sevani, B. Raman, and P. Joshi, "Implementation-based evaluation of a full-fledged multihop tdma-mac for wifi mesh networks," *IEEE Transactions on Mobile Computing*, vol. 13, no. 2, pp. 392–406, 2012.

[77] Ó. Seijo, J. A. López-Fernández, H.-P. Bernhard, and I. Val, "Enhanced timestamping method for subnanosecond time synchronization in ieee 802.11 over wlan standard conditions," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5792–5805, 2020.

[78] A. Mahmood, R. Exel, and T. Sauter, "Delay and jitter characterization for software-based clock synchronization over wlan using ptp," *IEEE Transactions on industrial informatics*, vol. 10, no. 2, pp. 1198–1206, 2014.

[79] X. Dai, S. Gao, K. Pan, J. Zhu, and H. F. Rashvand, "Wireless piezoelectric sensor systems for defect detection and localization," in *Wireless Sensor Systems for Extreme Environments: Space, Underwater, Underground and Industrial*. John Wiley & Sons, 2017, pp. 201–219.

[80] K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri, "Monitoring civil structures with a wireless sensor network," *IEEE Internet Computing*, vol. 10, no. 2, pp. 26–34, 2006.

[81] X. Liu, J. Cao, S. Lai, C. Yang, H. Wu, and Y. L. Xu, "Energy efficient clustering for wsn-based structural health monitoring," in *Proceedings of INFOCOM*. IEEE, 2011, pp. 2768–2776.

[82] Q. Hu, X. Liu, Q. Wu, J. Cao, Y. Liu, and Y. Tan, "Cluster-based energy-efficient structural health monitoring using wireless sensor networks," in *Proceedings of CSSS*. IEEE, 2012, pp. 1951–1956.