2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 979-8-3503-5841-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/RTAS61025.2024.00029

HAEST: Harvesting Ambient Events to Synchronize Time across heterogeneous IoT devices

Adeel Nasrullah
UMass Amherst
anasrullah@umass.edu

Fatima M Anwar
UMass Amherst
fanwar@umass.edu

***Abstract*—Synchronizing clocks is a resource-intensive and a resource-rigid task; this makes it challenging to align time across resource-constrained and heterogeneous IoT devices. Just as low-power IoT devices harvest energy from the environment, we propose to scavenge timing information from environmental events to conserve device power and bandwidth. We convert ambient events in an environment – sensed by various sensing modalities – into synchronization signals. Our approach, HAEST, leverages prevalent sensors on commodity platforms such as accelerometer, microphone, and optical sensors to timestamp commonly observed events. We present a light-weight and robust approach to detect events across different types of sensors and devices. This creates ample opportunities to align time and simultaneously avoid the clocks to drift apart. Through evaluation on a hardware testbed in a smart home and a wireless body area network, we show that HAEST achieves clock accuracy as low as sub-milliseconds with no cost for IoT devices. Importantly, our use of off-the-shelf IoT platforms in our evaluations, establishes the universality and applicability of our approach to a variety of smart spaces with heterogeneous devices.**

***Index Terms*—Time synchronization, Sensing modalities, Ambient events**

Application	**Data Rates (Hz)**	**Required Time-Sync Accuracy (ms)**	**Solution Deployed**
Wireless Earphones [17]	N.A.	30-0.5	Custom (over UDP/ICMP)
Visual Inertial Odometery [18]	10-100	10	N.A.
Fall Detection & Localization [19]	1000	1	GPS
Human Activity Recognition [20]	50	100	N.A.
Autonomous Vehicle sensor system [6]	30-44100	~ 20	Sensor Cross-Correlation
Muscle Fatigue Estimation [8]	500	~ 10	Serial Communication (wired)
Body Area Network (Parkinson's Disease Monitoring) [21], [22]	100	~ 10	FTSP

Tab. I: Time Critical Applications

I. Introduction

Aligning time accurately across distributed and heterogeneous devices in a network without additional communication cost on already resource-constrained devices is a critical area of research. Fundamental to most traditional synchronization protocols is to communicate timing packets over homogeneous networks [1]–[4]. We argue that networks are becoming increasingly heterogeneous i.e. distributed devices with different radios and sensing capabilities serve a variety of emerging IoT applications [5]–[9]. Additionally, the energy budgets for these networks are shrinking, which makes traditional protocols either incapable, inaccurate or costly for most applications. Our approach, HAEST, breaks away from these traditional trade-offs among performance, universality, and cost, and presents a time-sync service weaved into the diverse sensing infrastructure that relies on ambient sensing events as timing signals. We argue that a scheme synchronizing heterogeneous sensor networks would decrease system complexity [10], increase performance [11] and security [12] in edge deployments.

Synchronization is a fundamental service for emerging IoT applications ranging from safety-critical national, industrial, and medical [8] infrastructure to human-in-the-loop systems [13]. Interestingly, these disparate applications have diverse timing requirements as shown in the table I. For instance, driver assistance systems, enabled by the sensor data from multiple sub-systems, need to synchronize up to 10s of milliseconds [6], while sensor networks for occupancy and elderly fall detection need their time aligned up to a few milliseconds [14], and sensor networks for natural hazard monitoring may need sub-microseconds accuracy [9]. However, achieving these timing requirements in diverse and heterogeneous networks is challenging due to several reasons. First, non-deterministic timing and networking stacks introduce uncertainties in propagation delay measurements by the IoT devices [15]. These uncertainties result from delays caused by channel contention, asynchronous queues, and scheduling on multi-tasking devices. Second, recent sensor network deployments (indoors and outdoors) are increasingly adopting a mix of networking technologies [16] such as BLE, Zigbee, and WiFi. Devices with incompatible radios can only share time through a third server (or a gateway device) which comes at the cost of extra hops with radio dependent delays adding further clock errors. Finally, most devices are resource-constrained and cannot communicate frequently for accurate clocks.

The goal of this work is to break away from the established trade-offs in clock synchronization literature. Traditional packet exchange based clock alignment methods are either universal but with low clock accuracy (Network Time Protocol – NTP [1]), or provide high clock precision but with customized hardware or software solutions (Precision Time Protocol – PTP [2], GPS, Cross Technology Synchronization – CROCS [23], C-Sync [24]). Other methods that rely on universal sensing signals such as electromagnetic radiation from power lines provide a range of clock accuracy. However, they too require either custom hardware (Syntonistor [25], Electric Network Frequency – ENF [26]) or special conditions like human contact (TouchSync [27]), making these approaches only suitable for a limited class of devices. Our observation is that these methods provide an accurate clock by compromising either universality, applicability to heterogeneous devices, or hardware/network cost. In contrast, our proposed approach HAEST provides a *universal* timing service to heterogeneous IoT devices that achieve clock accuracy up to less than a millisecond without additional hardware, computational, or network cost for the IoT devices.

HAEST achieves time synchronization by harvesting ambient events captured by sensors on IoT devices, disentangling them from devices' networking capabilities. HAEST is a receiver (RX) to receiver (RX) synchronization approach [4], where two devices observe and timestamp a common ambient event using the same or different sensing modalities. Figure 1 shows a variety of dominant sensors in a smart space such as microphones, optical and Inertial Measurement Units (IMU). These sensors observe many ambient events. For example, opening and closing a door will produce sound, vibrations, and cause room luminosity levels to change, and their corresponding sensor types will capture and timestamp these events simultaneously. HAEST uses a gateway device to correlate the timestamps corresponding to the common events for estimating clock offsets and relative frequency drifts for the sensors involved. It moves the computational cost of time synchronization from the resource-limited IoT devices to the gateway. It also eliminates the need for dedicated message passing among devices. We argue that, like our example in Figure 1, most indoor and outdoor smart spaces deploy a range of sensors for monitoring activities that generate different types of events, thereby establishing the universality of event sensing and applicability of HAEST to many applications.

HAEST relies on the detection and timestamping of common events in the IoT sensor data, putting event detection at the heart of its design. Despite extensive research literature, the existing event detection solutions do not work well with HAEST. For instance, many event detection approaches focus on a fixed application [19], [28] and do not extend to other sensing modalities. In addition, generalized event detection approaches [29]–[33] often require manual calibration or parameter estimation for each new instance of a sensor which is impractical for HAEST's at scale adoption. Complex event detection [34], [35] approaches can be adopted across different sensors on a scale, however, they demand GPU resource that is often unavailable at a gateway device. We aim to detect a variety of events with various sensor types using off-the-shelf IoT and gateway devices with minimal calibration cost.

To achieve universality and limited cost without compromising accuracy in HAEST design, we address the following challenges: 1) sensing event detection is a computationally intensive task, for example, Cadence [36], an event detection approach specifically designed for IoT streaming applications, has an inference throughput of 33 samples/second on a GPU-equipped machine. Still, it falls short of handling data from sensors operating at higher sampling rates (e.g., IMU and audio). HAEST presents unsupervised learning based robust event detection mechanism and optimization approaches to run a high-throughput inference on gateway devices, which

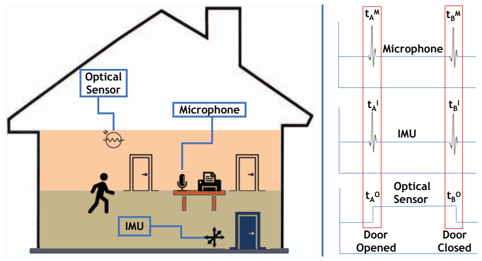


Fig. 1: HAEST overview: Co-located sensing devices observe numerous ambient events in common that can be leveraged to align time across them. On left, human activity is detected by three prominent sensors i.e. audio, inertial and optical sensors. On right, events appear in the data and their timestamp is recorded by these sensors simultaneously.

lack GPUs and accelerators. To achieve this goal, we leverage integer-only computations and a dual-stage sensor data encoding scheme (part-shared and part-specialized) and significantly improve inference speeds. 2) Our event detection algorithm, like any other deep learning algorithm, is error prone and may detect false or delayed true events. These delays are a result of different sensors that observe the same event but are not equidistant from the event source and capture the event several milliseconds apart. Our proposed clock parameter estimation approach mitigates event detection delays and filter out false events. We note that the timestamps of the events experiencing identical delays tend to cluster around a fixed value. Hence, the drift measured using timestamps of this cluster represents actual clock drift, since fixed delays do not impact the drift. Our approach obtains clock drift for each cluster separately and aggregates it to obtain the final drift. Our final challenge is that all devices in a sensor network do not observe the same events and lose opportunities to synchronize with each other. To achieve network-wide synchronization, we propose a clustering mechanism and a pair-wise synchronization scheme – based on devices' sensing and deployment properties – to align devices within and across clusters.

Contributions of the paper. *Our goal is to blend time synchronization service into the sensing infrastructure, enabling accurate clocks for various sensor platforms with diverse and disparate capabilities at no additional cost for the IoT devices.* Our main contributions are as follows:

1. We propose a robust event detection scheme based on auto-encoders with quantized weights to reduce cost. We use a common auto-encoder for event detection across different sensors for cost reasons, but to preserve event detection performance, we introduce a sensor-specific layer to the common auto-encoder, and propose a cascaded event detection approach for sensors operating at high sampling rates of several kHz.

2. We conduct a comprehensive study of timing characteristics on heterogeneous sensing platforms with the most prevalent sensing modalities. Our key findings are: (i) A sensor with lower sampling frequency dominates the pairwise clock error. For example a 50Hz accelerometer paired with a 200Hz one bounds the error in order of milliseconds compared to sub-millisecond. (ii) Different kinds of sensors can observe a common event but are restricted by large propagation delays (iii) Clock accuracy depends upon the number of mutually observable events. For example, microphones on two devices

would align their clocks well if the number of common events per minute is 5 instead of just one. As such, (iv) we present a quantification of the effect of sensor characteristics and event frequency on clocks, and propose a clock parameter estimation technique that aids in network-wide synchronization.

3. Our clock verification mechanism for a smart home and body area network deployment – consisting of heterogeneous platforms with different sensing modalities – shows that HAEST achieves an average error of a few milliseconds for a device pair under different sensing and environmental conditions. We also see that HAEST can achieve a sub-millisecond error for device pairs with better timestamping capabilities. The error increases to 10's of milliseconds for network-wide synchronizations. Under similar conditions, HAEST achieves as much as 5x smaller clock error compared to NTP while saving up to 36% in power consumption.

II. Literature Survey

Event Detection. Event detection, also known as time series segmentation, is integral to most distributed sensing applications [6], [8], [18]–[20], [22], and it is achieved by either i) each sensor in the network or ii) a gateway receiving data from a multitude of sensors [37]. The former is often application and sensor specific, e.g., natural hazard detection using geophones [9], eating detection with IMU [5], footstep detection from audio data [38]. However, in a heterogeneous sensor network, data is typically offloaded to a gateway device, for multi-sensor event detection (and classification) and many sensors lack resources for in-situ processing. This event detection is however application specific as opposed to HAEST which requires sensor and platform agnostic approach. SenseHAR [28] presents one such solution, however, it only works for data originating from a single platform. In contrast, HAEST detects events separately in each sensor's data and correlate them for time synchronization.

Generalized event detection methods can be broadly divided into two categories: i) probability distribution estimation [32], [33], [39] and ii) density ratio estimation [40]–[43]. However, these methods do not scale well as they either rely on i) manually calibrated parameters [32], [33], [39] or ii) simple statistical measures [42], [43] like mean, variance and spectrum which may not be the best predictors of events across all sensor types. Our work extends Lee et al.'s [44] non-parameterized event detection approach to achieve a lightweight and scalable event detection on edge gateways.

Packet-Exchange based Synchronization. Communication dependent time synchronization solutions for sensing applications are either resource-intensive in terms of high power consumption and bandwidth utilization (GPS, NTP, and PTP) or are resource-rigid, i.e., rely on dedicated infrastructure or specialized hardware ([7], [45]–[48]). C-sync [24], a recent work, provides resource-efficient time synchronization. However, it, too, is tied to a homogeneous networking stack. HAEST, in contrast, is designed for heterogeneous platforms and caters to the timing needs of a wide variety of devices without demanding resources from them. Finally, Crocs [23] leverages cross-technology communication [49], [50] to synchronize clocks between WiFi and ZigBee devices. However, in contrast to HAEST, it is tied to these two technologies and does not extend to other prevalent technologies such as BLE.

Sensing based Synchronization. There is an extensive literature exploiting sensing for clock synchronization. Electromegnetic Fields (EMF) radiated by power lines have been used for time sycnhronization across several studies [25]–[27], [51], [52]. However, these methods either require special hardware [25], [26], [51], [52] or special conditions [27]. Li [53] et al. explores optical sensing for time sync and Fridman et al. [6] uses cross-correlation to align accelerometer and optical flow data from an autonomous vehicle. However, the former is limited to a single sensor modality and the later is intended for vehicular sensor networks. HAEST can use each of these sensing modalities while exploring numerous cross-modalities to design universal time synchronization for heterogeneous off-the-shelf IoT devices.

III. Design Overview

HAEST timestamps sensing signals originating from ambient events as opposed to periodic radio communication by RBS [4], the other RX-RX time-sync approach. These signals are offloaded to a gateway that computes relative clock offset and drift across all devices. Thus, it allows heterogeneous sensing devices – without the same radio technology – to participate in time synchronization.

We argue that harvesting ambient events (reference signals) from the environment decouples synchronization from a particular sensing and radio technology. In addition, removing the need for packet exchange between devices, not only enables devices with different networking stacks to synchronize mutually through the gateway, but also limit extra communication costs at the IoT devices. Finally, HAEST avoids additional energy and computational costs for IoT devices by moving time synchronization calculations to the gateway server.

Figure 2 shows the HAEST design, which consists of two phases: i) the *bootstrapping* and ii) the *synchronization phase*. While the former is responsible for setting up event detection encoders for a variety of sensors and finding the optimal device pairs for network-wide sync, the latter detects mutual events and estimate clock parameters for all pairs in the network.

Bootstrapping phase. In this phase, we first build a graph of sensor network, where, each node is a sensing device, and each edge represents a common event detected by a device pair. We then collect data from the graph network and use it to learn sensor data encodings by training autoencoders for event detection. The sensor encodings produce mutual event count i.e. the number of events observed by every pair, which is used to prune the original graph by selecting the appropriate synchronization pairs (*sync-pairs*) based on device pair characteristics (sensor types and sampling rates) and the frequency of common events. *Sync-pairs* are subsets of sensing pairs in the original graph and selected to ensure all nodes remain connected to each other via one or more hops. They

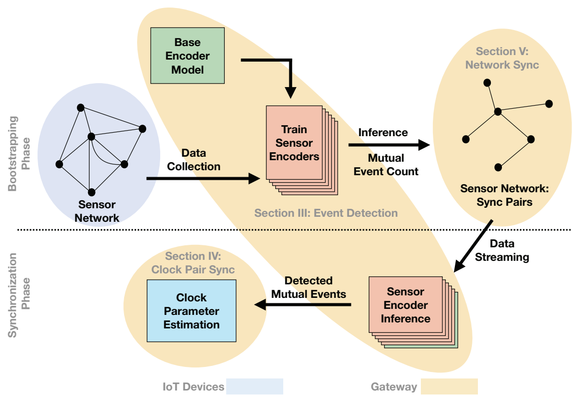


Fig. 2: HAEST Design Phases and Blocks

allow efficient network-wide time-sync rather than the brute-force method, which synchronizes along every graph edge.

Synchronization phase succeeds bootstrapping and uses autoencoders – trained during the previous phase – to detect common events between the *sync-pairs* and align their clocks using the detected mutual events. In doing so, HAEST achieves network-wide time-sync across all pairs through the network-effect i.e. devices are transitively aligned with other devices in the network over one or more hops.

Note that data collection occurs only at IoT devices; the rest of the steps in both phases are managed by the gateway. In both phases, HAEST overcomes challenges of heterogeneity, delays, detection errors, and learning cost. As such, Section IV describes our event detection approach, while Section V covers our clock parameter estimation approach guided by a study on sensing clock characteristics for a device pair. Finally, Section V-C introduces techniques to extend our device pair time-sync scheme to a heterogeneous sensor network.

IV. Event Detection

As described above, event detection is fundamental to both the bootstrapping and synchronization phases. It refers to the identification of an event's starting (and end) points in time series sensor data, and it is a well-studied problem [34]–[36], [44]. However, most of these approaches add computational complexity for high accuracy. It is at odds with our objective of parallel event detection for multiple sensor streams without exhausting the gateway device's resources. In a typical deployment, many devices with different sensors connect to a single gateway which detects events for the connected sensors.

Addressing the constraints of limited resources and sensing heterogeneity, HAEST **event detection design** at the gateway has three goals: i) a **light weight** design that can process data received from multiple devices using the limited computational resources; ii) **universality** i.e. the same event detection approach works for different sensor types, as resources of a typical gateway (e.g., Raspberry Pi4) may be insufficient to run a dedicated event detection for each sensor; and iii) **scalability** i.e. event detection without manual calibration for heterogeneous deployments, which may have the same types of sensors but from separate manufacturers with different characteristics (e.g., sensitivity and resolutions). Below, we present HAEST event detection techniques along with the challenges and solutions to meet its design goals.

A. Time Series Segmentation

Our event detection approach relies on the observation that the data points of a given sensor have a specific underlying distribution [44], which changes upon an event's occurrence. Given this observation, we expect consecutive sensor data points to have a smaller distance between them if recorded during an event or at rest since they belong to the same distribution. In contrast, the distance between the data captured at rest and the data points recorded during an event will be larger. We use this observation to predict events in the sensor data. However, raw sensor data often contains ambient noise (e.g., a microphone exposed to TV sound), which impacts the distance measurements. We use autoencoders to capture the underlying sensor data distribution. Autoencoders are neural networks that transform the raw data into an intermediate representation, called encodings, and then reconstruct the original data from these encodings. They have demonstrated the ability to capture distribution-specific characteristics of sensor data [54]. We obtain the sensor data encodings using auto-encoders and compute the distance between consecutive sensor data points. The peaks in this distance score indicate the occurrence of the ambient events.

For our design, we use an autoencoder composed of two fully connected neural networks: i) Encoder Enc that maps the raw sensor data X to the encodings $f = Enc(X)$ and ii) Decoder Dec that converts the encodings back to the raw data $\hat{X} = Dec(f)$. To train the autoencoder with sensor data X of length T and dimensions N_c, we take sensor data segments of length N_w such that for timestep t the segment starts at $t - N_w + 1$. It gives us a total of $T - N_w + 1$ segments, starting at $t \in \{0, 1, 2, ..., T - W_n + 1\}$, where each segment consists of a matrix $s_t \in \mathbb{R}^{N_c \times N_w}$. We collapse this matrix into a single column vector $s_t \in \mathbb{R}^{N_c N_w \times 1}$, which will be the input to our autoencoder and its expected output. After the autoencoder is trained, we obtain the encodings for each segment as $f_t = Enc(s_t)$, and compute the distance between the consecutive segments as $d_t = \frac{||f_t - f_{t-1}||_2}{\sqrt{||f_t||_2 * ||f_{t-1}||_2}}$. Here, d_t represents a distance score of the sensor data X_t at timestep t. Finally, we find the timestamps corresponding to the local maxima (detecting ambient event) in the distance sequence d.

We evaluate our event detection approach using two sensor data sets, that capture various human activities (ambient events), i) UCI [55]: 6 dimensional IMU data (accelerometer and gyroscope) and ii) Dcase2016 [56]: audio data. Figure 3 shows the ROC curve for the two data sets[1]. The area under the curve shows individual event detection accuracy for Dcase2016 and UCI datasets, which is 0.77 and 0.53, respectively. High false positive rate shows that events are detected when there aren't any, and not because of missing true events. However, we argue that for HAEST, we are only

[1]Appendix A provides the details related to our experimental setup and evaluation metrics, i.e., ROC

	Original		Quantized	
	Area under ROC	Inference speed (samples/sec)	Area under ROC	Inference speed (samples/sec)
DCase2016 (Audio)	0.77	147	0.60	3600
UCI HAPT (IMU)	0.53	147	0.55	3600

Tab. II: Event Detection Performance with Quantized Encoder on RPi4

	Original	Enc_b	$Enc_b + Enc_s$
	Area under ROC	Area under ROC	Area under ROC
DCase2016 (Audio)	0.77	0.7	0.8
UCI HAPT (IMU)	0.53	0.49	0.53

Tab. III: Event Detection performance using sensor-specific encoders

interested in mutually observed events by two or more sensors, and the probability of false positives for mutual event detection is low. We use this intuition later to filter the false events.

B. Quantization

Our event detection approach can only process almost 147 time steps/second on a RPi4, a typical gateway device. This speed is insufficient to process multiple sensing devices (operating at 100s of Hz) connected to one gateway. We propose a quantization approach to speed up event detection. As integer computations are significantly faster than floating-point computations on gateway devices lacking accelerator hardware like GPUs (e.g., RPi4), we use 8-bit post-training quantization of encoders Enc. Our quantization approach uses uniform affine transformation that requires the calculation of scale and zero point [57] (implemented by TensorFlow lite API). It means successful post-training quantization has a prerequisite, i.e., training data has a two-sided distribution. This property is critical to preserve the encoder's Enc event detection performance because, if the input to the network has a one-sided distribution (say $(0.5, 1)$), the model is still quantized as if the distribution includes 0 (that is, $(0, 1)$). It leads to big quantization steps and significant performance degradation. With post-training quantization in mind, we normalize the sensor data between $(-1, 1)$ for autoencoder training.

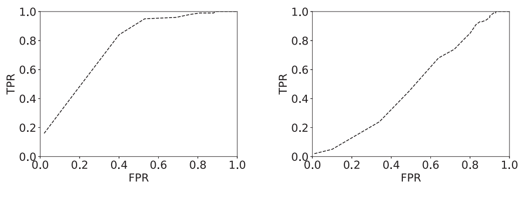
We compare quantized encoder's event detection performance to the original for both datasets and show results in Table II. We see more than 20× improvement in the encoder's inference speed upon quantization, thus satisfying our goal of *light-weight* design. Note that there is no significant impact on event detection performance for IMU data, but there is a performance drop of 20% for audio data. As described earlier in section IV-A, this high false detection will be mitigated using our false event detection filter.

C. Sensor Specific Encoders

Our event detection approach requires a dedicated encoder for each sensor connected to the gateway, but the resources may be insufficient to meet this demand. For example, in a scenario where RPi4 is the gateway device for 6 sensors, it would need to run 6 encoders which will quickly overwhelm its 4 CPU cores. To make our approach lightweight and universal, we divide the event detection encoder Enc into two parts. First is the base encoder Enc_b that is common to all sensors, and second is a dedicated sensor-specific encoder Enc_s for each sensor connected to the gateway.

We design Enc_s as a single-layer encoder that performs a sensor-specific transform on encodings computed by the base encoder. For sensor data X, its feature vector in the latent space would now be $f = Enc_s(Enc_b(X))$. Having this in place, the data from the sensors m are merged into a matrix $X = [X_1, X_2, ..., X_m]$ where X_i is the data vector of the i-th sensing device. We obtain a feature matrix $f_b = Enc_b(X)$, where $f_b = [f_b1, f_b2, ..., f_bm]$. Finally, the sensor specific features $f_1, f_2, ..., f_m$ are obtained as $f_i = Enc_s(f_{bi})$.

In our design, the sensor-specific encoder Enc_s is only 9.1% of the base encoder Enc_b (see Appendix A). For our earlier example, this approach only needs computations equivalent to running two encoders for 10 sensors. Table III shows event detection performance with sensor-specific encoders for the IMU and Audio data sets. The base model trained on both sensors' data shows degraded performance. However, the addition of a sensor-specific encoder mitigates this performance loss. Sensor-specific encoders enable different sensor types to share the base encoder Enc_b, meeting our goal of *universal* event detection approach.



(a) Audio (b) IMU

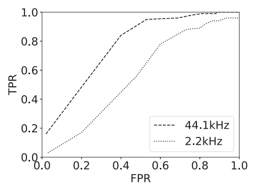
Fig. 3: ROC curves for a) DCASE 2016 dataset b) UCI Human activity recognition dataset.

D. Cascaded Detection

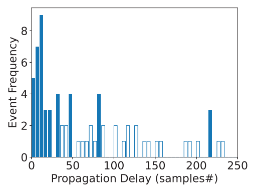
Despite being lightweight, our event detection approach can only process a few thousand sensor data samples per second (Table II). While it is sufficient for most commodity sensors (IMU, optical sensors), it cannot process audio data often sampled at high frequencies (typically 44.1 kHz). We design a cascaded event detection approach for sensors operating at high sampling rates. This approach consists of the following steps: i) downsample the sensor data (down to 1-2kHz), ii) perform event detection on this downsampled data, and iii) if we detect an event during an interval, re-perform event detection for this same interval using original data.

For a signal $S = s_0, s_1, s_2, ..., s_N$ recorded over time period T, $S_d = s_0, s_c, s_2c, s_3c, ...s_N/c$ is signal downsampled by a factor of c. Our cascade detection approach detects events on this signal S_d. Upon an event detection at timestep kc, we obtain a segment $W = s_{kc-w/2}, s_{kc-w/2+1}, ..., s_{kc}, s_{kc+1}, ..., s_{kc+w/2}$ from the original signal S of the length w centered at kc. Event detection using the segment W provides a more precise timestamp, which would reduce errors in our estimation of the clock parameters.

Figure 4a shows ROC results for event detection performance on downsampled audio data (0.57) compared to the original (0.77). We argue that detecting true events with downsampled data is still effective.



(a) Event Detection Performance on Downsampled Audio



(b) Propagation delay distribution for a pair of audio sensors operating at 2kHz

Fig. 4: (a) ROC curve for event detection using downsampled audio data, (b) Distribution of propagation delay observed by a sensor pair

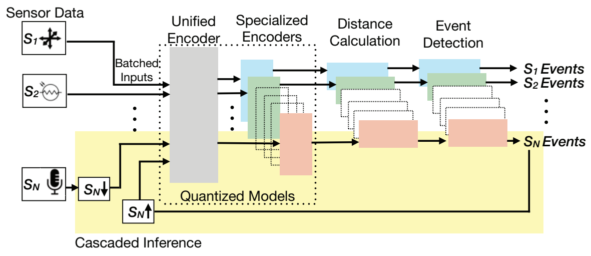


Fig. 5: HAEST Event Detection.

E. False Event Detection Filter

This filter empirically determines a rolling window of fixed duration to identify and reject false mutual events. It counts the number of detected mutual events; a single event within the window is kept as a true event, while multiple detected events in the window show the presence of false events and lead to the dropping of all events in that window. While this strategy eliminates false events, it can potentially also drop the true coinciding mutual events. We cannot correlate these events uniquely as the order of their arrival at the two sensors is unknown. Thus we may lose synchronization opportunities, however, coinciding events are likely to occur in a high event rate environment, and we expect sensor pairs to get ample synchronization opportunities despite our filter.

F. Error Reduction in Event Detection

HAEST event detection is intended to identify synchronization signals in the sensor data. And it is important to identify an accurate timestamp for detected events. However, the noise in distance scores causes time errors in event detection. The peaks in distance scores that indicate the location of a common event do not align between two sensors, which adds to the time error. Figure 6 shows HAEST distance scores for common audio events captured by two sensors in its second column. We see a large time error (in the order of hundreds of timesteps), and the error is inconsistent across events. To minimize this time error for better clock offset and drift estimation, we propose a two-step signal processing technique:

Low Pass Filter: Our first step removes noise from the distance scores, where we chose *Savitzky-Golay*[2] [58], as a low-pass filter with the following properties important to the next step: i) it fits a polynomial curve to a local segment of the signal which removes the noise but maintains the signal's global structure, and ii) when used with higher order polynomial, it preserves the height and width of signal peaks.

First Order Derivatives: We observe that the increase in the magnitude of the distance score preceding local maxima for a given event is more consistent between devices than the local maxima itself. We capture this increase using a first-order derivative of the low-pass filtered distance scores (shown as final scores in the last column of Figure 6), and treat the corresponding maxima as our detected event. Properties of the *Savitzky-Golay* filter, mentioned earlier, ensure that changes in distance scores (signal peaks) are preserved during noise removal and captured accurately by the derivative.

Applying our time error reduction technique to the audio events in Figure 6 shows negligible time error between the events in the third column. We will use these precise event timestamps for clock parameter estimation.

G. Putting everything together

Figure 5 shows the complete HAEST event detection design. Data from all sensing devices are batched together for inference with the base encoder, followed by sensor-specific encoder inference for each sensor data. Note that all encoders are quantized, and sensor data with higher sampling rates undergo cascaded inference. Using the features computed by the encoders, we compute distance scores for each sensor data and find local maxima, i.e., the detected events.

HAEST event detection, relying on unsupervised training of sensor encoders shared between various sensors, requires negligible manual calibration, and it is **scalable** across deployments. However, the window size N_w of the input data to the encoder needs one time calibration. It is a function of the sampling rate and sensor type, that is, $N_w = 400$ at 50Hz for IMU while $N_w = 2400$ for audio data at 44.1kHz and 2.2kHz. For each new sensor type and sampling rate combination when deployed for the first time, N_w will be determined empirically. Since there are finite sensor types and sampling rate combinations, the need to determine N_w will decrease as the number of deployments increase.

V. CLOCK SYNCHRONIZATION

The clock parameters (offset and drift) of a device pair are estimated using common events detected by HAEST event detection. Consider two devices A and B detect a common event and obtain timestamps t_A and t_B for this event, using their local clocks. The offset between the two devices will be $offset = t_A - t_B$ and their relative drift is the rate at which the offset changes $\frac{d(offset)}{dt}$. However, the two sensors may

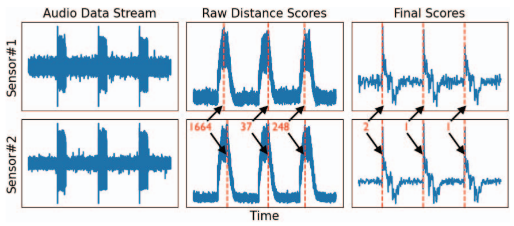


Fig. 6: HAEST event detection using audio data from two different sensors. The events detected using raw scores (section IV) exhibit large errors. The post-processing event detection using final scores exhibits negligible error.

[2]We use Scipy's implementation of this filter. We use a seventh-order polynomial with a window size of 101 data-points.

detect a common event with a delay t_P in which case, the offset equation becomes $offset + t_P = t_A - t_B$. We name t_P as *propagation delay*. For the correct clock parameter estimation based on common events, it is crucial to mitigate the propagation delay. This section presents our clock parameter estimation scheme, for a device pair, mitigating propagation delay. We follow it by an in-depth study evaluating our proposed method. Finally, we describe our network sync protocol that extends the pair-wise clock parameter estimation scheme to synchronize the whole sensor network.

A. Clock Parameter Estimation

Our technique has to overcome the challenge of *propagation delay* caused by heterogeneous sensors and deployments. We know that different sensors can recognize different aspects of the same event, e.g., a microphone hears a door open, while an optical sensor detects a change in light intensity from the same door opening event. However, the sound signal propagates to the nearby microphone as soon as the doorknob moves, but the change in light is propagated after the door is half open. Additionally, difference in signal propagation speeds and the distance travelled by the two signals also adds to the propagation delay to as high as 100 milliseconds.

Delay Adjusted Clock Sync. Unmitigated propagation delays cause uncertainty in offset and drift estimation. While large in magnitude, as in the example above, propagation delay t_P stays unchanged for the repeat occurrence of the same event; because event delay depends on sensor types, event locations, and signal propagation speeds, all of which stay fixed across different instances of the same event. For example, each time a door opens, the sound reaches a wall-mounted audio sensor earlier[3] than the audio sensor on a desk with the same delay. Given this assumption, any change in $t_A - t_B$ for the repeat occurrence of an event reflects the change in $offset$ between two devices. With this in mind, as a device pair observes a few events repeatedly, measurements $t_A - t_B$ will cluster around distinct values. Each cluster spans a small range due to the $offset$ component of these measurements.

We test our observations using an audio sensor pair (interfaced ESP32 at 2kHz sampling rate) deployed near a window. These sensors are fed with an intermittent sound pulse from an equidistant smartphone. In addition, the audio devices are exposed to sounds originating from human activity inside the room, adjacent corridor, and from outside the building. Figure 4b shows the distribution of $t_A - t_B$ measurements clustered into bins spanning 5 timesteps each. As expected, we see that most of these measurements cluster into distinct groups (solid bars) where each has experienced the same propagation delay. The larger number of events experience a very small delay which corresponds to synthetic events from the smartphone and activity inside the room. While larger delays belong to the events occurring at a distance in the corridor. We use these observations for drift estimation.

[3]sound travels faster through solids than air

Our *delay adjusted clock sync* scheme uses common event timestamps obtained within a rolling window of 10 minutes. We chose this window length as it allows the time difference between the devices to grow bigger, making it easier to observe this change through sensor timestamps. Next, it clusters the measured offset values $t_A - t_B$ into groups spanning a few timesteps. We achieve this using a standard k-means clustering algorithm where k is the number of offset groups. This algorithm clusters the data into k groups such that their means are as distinct as possible. For our experiments, we chose $k = 20$ which allows clusters to span a small range, putting events with the same t_P into common groups. Next, it drops the offset clusters with less than 3 measurements. It, then, calculate $drift$ for each of the remaining cluster as $drift = \frac{d(t_A - t_B)}{dt} = \frac{d(offset + t_P)}{dt} \approx \frac{d(offset)}{dt}$ assuming t_P as constant. The average of the $drift$ values from all clusters is our final drift value. Finally, it selects the mean of the $offset$ group with the smallest mean as the final offset value. This offset may still contain propagation delay t_P, however, computing the $offset$ using the smallest measurements minimizes the error contributed by this delay.

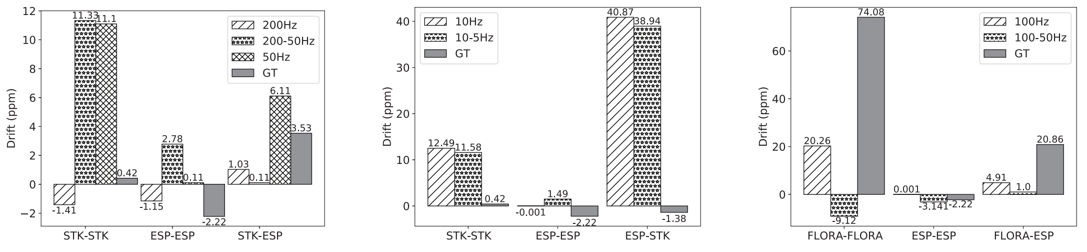
Given the event detection and clock parameter estimation techniques for sensor networks, we are now interested in studying the sensing characteristics that provide the most optimal clock performance. We perform a measurement study on IoT devices equipped with a variety of sensors.

B. Measurement Study

Using commodity platforms with prevalent sensors, we empirically study the effect of sensor sampling frequency on timestamping, the ability of different kinds of sensors to align their clocks, and characterize the impact of event occurrence rate on synchronization error, thus giving us insight into sensing capabilities that yield the best clock performance. Our study estimates clock drift for sensing pairs using our event detection and clock parameter estimation approach.

Testbed Setup. For our experiments, we use three embedded platforms: CC2650 SensorTag STK [59] by TI, ESP32 Things [60] device from Sparkfun, and Flora [61] from AdaFruit. STK has an onboard sensor suite, a separate microcontroller for applications and radio, and runs BLE stack 2.2.1. ESP32 Things device is interfaced with an external IMU, an optical sensor, and a microphone. It is a dual core device, with on-chip WiFi module and runs FreeRTOS based software stack. Our third device, Flora, extends the peripherals of ATmega32, an 8-bit microcontroller, interfaced with a microphone. Flora is connected to the gateway device via USB. We choose this set of devices to show that HAEST works across a range of platforms with heterogeneous compute, network, and hardware resources. ESP32 is the most resourceful of the group with a multitasking software stack, while Flora being an 8-bit micro-controller has the least resources.

We conduct experiments on three of the most prevalent sensors in smart devices: Inertial Measurement Unit (IMU), ambient light sensors (OPT), and microphones (AUD). We use an Android smart phone to generate periodic (10-15



(a) IMU-IMU (b) OPT-OPT (c) AUD-AUD

Fig. 7: Relative drift estimation among different device pairs using same sensor type

sec) events for the sensors, i.e. vibration for IMU, light for OPT, and sound events for microphone. We establish each device pair's ability to estimate relative clock drifts, which is compared to the ground truth (GT) drift. To obtain GT offsets and drift, we provide a pulse-per-second signal interrupting a GPIO pin and timestamp on two devices under study.

Timestamping. Delay in timestamping is a function of software stack operation, task scheduling, and stability of timer generating interrupts that prompt sensor readings. ESP operates at higher clock frequency and low clock drift; we configure a hardware timer on ESP to generate periodic interrupts to sample sensor data. STK uses a software timer to obtain periodic sensor readings and timestamps using a lower clock frequency. The software stack on STK has more abstraction layers between sensing and timestamp generation compared to ESP. Flora reads data in a software loop and the sampling period is maintained using delay functions. Flora has larger sampling delays as compared to timer interrupt based sensor sampling in ESP and STK. Also, Flora operates at 8MHz, a much smaller clock frequency, and has an unstable clock that drifts approximately at 50 ppm. In short, timestamps are obtained at different layers of the stack for all devices, and these stack delays contribute to the eventual clock accuracy.

Our first set of experiments use the same sensing types on the same and different devices (Figure 7), while the rest use different sensing types to estimate clock drift (Figure 8a).

1. Observations for Same Sensing Pairs. We collect data from same kind of sensors (IMU-IMU, OPT-OPT, and AUD-AUD) on five IoT device pairs (ESP-ESP, STK-STK, FLORA-FLORA, STK-ESP, FLORA-ESP). The sensor sampling frequency is varied from low to medium in the range of 5 Hz to 200 Hz. It is important to note that the source of all timing errors are IoT devices, which observe events from a single source located equidistant from device pairs.

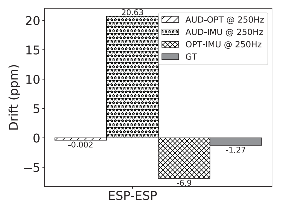
Figure 7 shows the disparities in the relative drift estimations between device pairs at different sampling frequencies. For ESP, we observe drift estimates much closer to the GT drift as compared to STK and Flora across all sensors. The GT drift of Flora is large compared to the other two devices, and the estimated drift is also farther from GT because of large software stack delays affecting timestamping delays and hence the drift estimates. We see in Figure 7 (c) that GT relative drift of one Flora with another Flora device is 74.08ppm, while one Flora to ESP is 20.86ppm. This means that the other Flora drifts in opposite direction at approximately 50 ppm. Similarly, in Figure 7 (a), we see that STK-ESP performs better than the STK-STK drift estimations. On the basis of these observations, we deduce that devices with better hardware and software resources – ESP in this case – get better drift estimates.

In Figure 7(a), we see better drift estimation for a higher sampling frequency because offsets are estimated with higher resolution (time moves in small steps). At high frequency, offset measurements have less error that translate into drift estimates closer to GT drift. However, for Figure 7 (b), the frequencies available for the STK device are 10 and 5 Hz. At such low frequencies, resolution for offset measurements is in 100's of msec causing huge in offset and drift. In essence, *higher sampling frequency yields better clock estimates*.

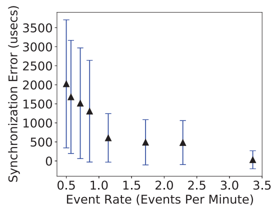
2. Observations for Cross Sensing Pairs. To study the effect of different sensor types on clock performance, we perform experiments only on ESP device pair while using all kinds of sensors i.e. IMU, OPT, and AUD, noting that observations made for ESP will also extend to other devices. Only one sensor is activated on each ESP for a given experiment that resulted in three different pairs i.e. AUD-OPT, AUD-IMU, and OPT-IMU. All sensor pairs are operating at the same sampling frequency. For our controlled experiments, we use smartphone applications that generate periodic vibration, light, and sound events manually aligned with each other. The manual alignment results in a small delay (orders of 100 millisecs) between two signals, e.g., light and vibrations, which simulates propagation delays.

The drift estimates for different sensing pairs, that is, AUD-OPT, AUD-IMU, and OPT-IMU are plotted in Figure 8a. AUD-OPT pair provides the best drift estimate among all, and the other two pairs with IMU have large errors because on the ESP device, IMU communicates with the processor over an I2C bus, whereas, both OPT and AUD sensors operate in analog mode. The overhead in reading analog signals is less than in a serial I2C read. Therefore, for both OPT and AUD data, timestamps are more accurate than IMU, justifying better drift estimates using the AUD-OPT pair. We deduce from this experiment that *some sensor types on a platform outperform the others even if all operate at the same sampling frequency*.

3. Event Rate Effect. Figure 8b shows an *inverse relationship between event occurrence rate and clock error* where



(a) Cross modality drift estimates



(b) Clock error vs Event Rate

Fig. 8: (a) Relative drift for cross sensing pairs; (b) Clock error versus number of events

low event rate results in large error spread. These results are obtained from an OPT-IMU sensing pair across ESP devices at 250Hz. With more than 3 events per minute, we obtain 100μsec mean and 360μsec std, whereas for 1 event per two minutes, the error increases to 2 msec mean and 3.5 msec std. This relationship between event rate and clock error is analogous to the relationship between packet exchange rate and clock error in traditional synchronization protocols.

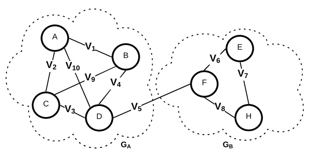
4. Sensing Pair Selection Mechanism. Often embedded devices are equipped with more than one sensor, and each sensor operates at a different sampling frequency. In this case, the devices may choose to synchronize through one or more sensing pairs. While they could synchronize using all available sensing pairs, we are motivated by our study that drift estimates are affected by sampling rate, event rate and sensing type. And if one of the sensing pairs involved is providing sub-optimal estimates, the overall synchronization error will deteriorate despite good estimates provided by others.

We propose a sensing pair selection strategy for the devices in a sensor network. Our observations suggest that 1) higher sampling rates yield better timestamps and accurate clock parameter estimates. Also, 2) high event rate provides greater number of synchronization opportunities leading to improved performance. Accordingly, we propose a simple strategy for sync pair selection that works for heterogeneous devices: 1) choose the pair with highest sampling rates for time-sync, and if this leaves us with more than one sensor pairs, 2) choose the pair which observes greater number of mutual events.

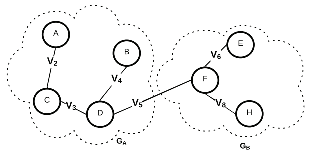
C. Network-wide Synchronization

Previous section established HAEST's device pair synchronization mechanism. In practice, sensing applications rely on a multitude of devices rather than a single device pair. Consider a typical smart space equipped with various sensing devices. A person may perform many activities while moving throughout this space. Each instance of human activity is observed by a group of co-located sensing devices monitoring a particular subspace. These devices collaborate to recognize the given activity, and would benefit from a tight synchronization between them [7]. While over an extended period, all sensors in the network would coordinate to build a timeline of these activities requiring rather coarse network-wide synchronization [62].

For such use cases, HAEST provides a graph-based protocol that leverages the network effect to synchronize all devices through our pairwise synchronization method in Section V-B. It aims to provide tight synchronization for co-located devices and coarse network wide sync as indicated above.



(a) Fully Connected Clusters



(b) Pruned Clusters

Fig. 9: Network synchronization through pair-wise clock alignment within clusters and across clusters

Synchronization Clusters. We frame our network synchronization task into a graph clustering problem. Each device is a node in this graph, while edges represent the common event count between nodes captured during the bootstrapping phase. Each node is connected to every other node, if they detect at least one common event. Figure 9a shows an example graph for a sensor network with two clusters. The device nodes are labeled from A to H, while the edge weights V_1 to V_{10} represent the nonzero event count between the devices. This event count depends on number and kind of sensors on each device. Once we form a graph using common event information, we use the Markov clustering algorithm to determine the clusters within the graph. One can also use other standard clustering algorithms to achieve the same end result.

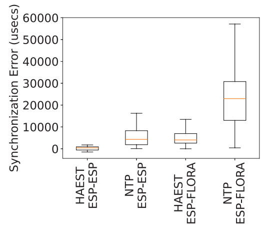
Intra-cluster Alignment. Once we group the devices into clusters, the nodes in a cluster (Figure 9a) are connected to multiple other nodes. We can synchronize each node with all other nodes. It will yield $m*(m-1)/2$ synchronization pairs for a cluster with m nodes. However, this naive approach results in multiple clock alignments for the same pairs causing inconsistencies and wastes resources. To achieve better performance and avoid prohibitive costs, we synchronize each device with only one other device. We use optimal sensing pair selection from Section V-B-4 to prune excess graph edges. Only those edges used for pair-wise synchronization (Figure 9b) remain. This approach aligns with our objective of minimizing the error between co-located devices while maintaining network-wide sync using the network effect.

Network Effect [63] is a phenomenon in a group of devices where pairwise alignment leads to whole group alignment via transitive synchronization. This effect has been leveraged in data centers as well [64]. For example, the subgraph G_B in Figure 9b has two synchronization pairs, i.e., (E,F) and (F,H). If device E clock is aligned with F and F is aligned with H, we say that E is transitively synchronized with H through F.

Inter-cluster Alignment. With clocks in each cluster aligned, the final step achieves time-sync across the clusters. To do so, we choose device pairs from the two clusters that can observe common events and select the pair with the highest common event count. We call these devices 'bridge devices' as they bridge the error gap between different clusters.

VI. Evaluation

We first compare HAEST performance with NTP, the only other protocol that universally synchronizes heterogeneous devices. We, then, evaluate the effectiveness of HAEST in predicting clock errors using a variety of case studies: one with fixed deployment in a smart home with activity-related


(a) HAEST comparison with NTP over USB

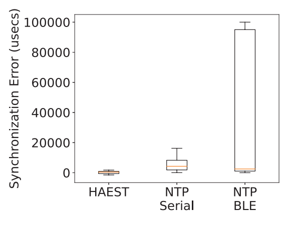

(b) HAEST comparison with NTP over BLE

Fig. 10: HAEST & NTP comparison over different communication channels. Unlike NTP, HAEST performance is agnostic to networking technologies

events, the other case study is on a wireless body area network (WBAN); a mobile deployment exposed to movement-related events. Final study concentrates on the performance of a hybrid of static smart space and mobile WBAN deployment.

A. Clock Synchronization Performance

Comparison with NTP. Network Time Protocol (NTP) is the most widely used and universal synchronization protocol. To compare it with HAEST, we interface two ESP and one Flora with a Raspberry Pi4 (the gateway). NTP is enabled on Pi via internet. The ESP and Flora communicate with Pi over USB or a BLE connection, and synchronize their local clocks to Pi following SNTP principles. This setup corresponds to real world edge deployments: most devices do not have internet access, instead they communicate with an NTP-synchronized central device over wired or wireless medium.

HAEST synchronizes two ESPs using IMU-IMU pair at 200Hz, and the ESP-FLORA pair via AUD-AUD at 100Hz. Figure 10a shows clock errors median and spread for both HAEST and NTP. We can see that HAEST achieves more than 5x better performance than NTP. We also compare HAEST with NTP over BLE in Figure 10b. It is evident that NTP over BLE further degrades clock quality for IoT devices while HAEST thrives irrespective of the communication medium between the IoT device and the gateway.

HAEST's computations are offloaded to the gateway with no timing cost at the IoT devices while NTP puts both computational and network overhead on the IoT device and increases power consumption to 36% compared to our setup. The power numbers in Table IV correspond to a NTP setup where one timestmap session lasts 1 sec after every 15 sec. One can reduce the frequency of NTP sessions but it will degrade its performance further. The computational cost at the gateway is addressed later in this section.

To summarize, HAEST's synchronization quality across a network of heterogeneous devices is better than or comparable to the NTP (the state-of-the-art) for heterogeneous networks. For perspective, Matter, a recent set of standardized specifications for IoT devices, refers to NTP and GPS for synchronization in its documentation [16]. HAEST achieves better accuracy than the former and incurs much lower costs than the later. Further, in contrast to GPS it also works indoors. Setting aside heterogeneity, it may not outperform the best (e.g. C-Sync [24], TPSN [65] etc.) but it still offers an advantage to battery powered sensors by extending their lifetime by saving power that would otherwise be spent on time synchronization (e.g., while using HAEST ESP32 Things spend 36% less power than NTP as discussed above).

Device	Sync Protocol	Average Power Consumption (mW)
ESP32 Things	HAEST	42.08
ESP32 Things	NTP	57.19

Tab. IV: Power Consumption of HAEST vs NTP over BLE

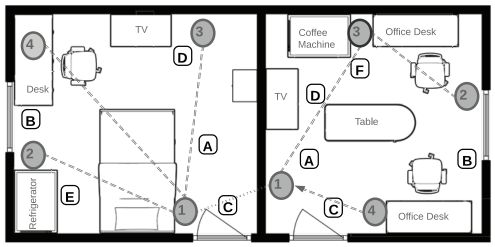


Fig. 11: Floor plan shows devices 1, 2, 3, 4 (explained in Table V) deployment and event sources A, B, C, D, E, F (explained in Table VI) in a smart home

Case Study 1: Smart Home. We implement HAEST design for a smart home and deployed devices as shown in Figure 11. It consists of a bedroom and a living room (also a make-shift office) adjacent to each other. Eight devices of three different kinds (ESP, STK, Flora) carrying three types of sensors (IMU, AUD, OPT) are deployed. Details of device and sensor types are in Table V, and Table VI compiles a few of the most common ambient events generated by humans, pets, and machine activity. Most of these events e.g., opening the door and windows produce signatures detected by all sensors.

Our smart home setup in Figure 11 forms two device clusters connected to each other via bridge devices deployed at the adjacent doors. We analyze the performance of every pair in a cluster and also between the two clusters, and compare clock errors via single and multiple intermediary nodes.

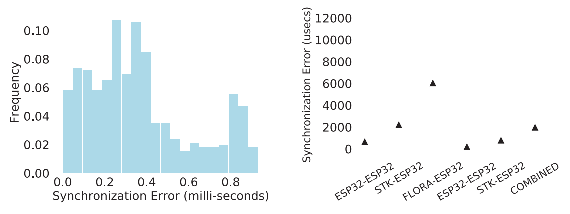
Figure 12 and 13 show the distribution, absolute mean and standard deviation of clock errors for intra- and inter-cluster devices respectively in our floor plan. For inter-cluster, we analyze the errors among those device-pairs that do not belong to the same cluster. These device-pairs are synchronized with each other via multiple intermediary devices including bridge devices. We can clearly see that error means and spread are better for intra-cluster as compared to inter-cluster pairs. Except in the case of Flora-ESP, where intra-cluster performance is comparable with inter-cluster device pairs

Device ID	Device Platform	Sensing Modalities	Sampling Frequency (Hz)
1	ESP32 Things	AUD / IMU	8000 / 200
2	ESP32 Things	IMU / OPT	200 / 50
3	Sensortag CC2650	IMU / OPT	200 / 50
4	AdaFruit Flora	AUD	100

Tab. V: Device Configuration Table

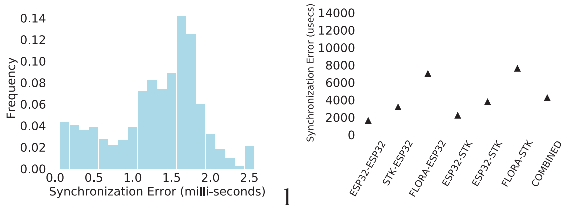
Event ID	Event Source	Potential Events	Sensing Type
A	Humans, Pets	Footsteps, Physical Interactions	AUD, IMU
B	Windows	Window Open/Close Window Shades On/Off	AUD, IMU, OPT
C	Doors	Door Open/Close	AUD, IMU, OPT
D	LED Screen	Light Intensity Changes	OPT
E	Refrigerator	Door Open/Close	AUD, IMU, OPT
F	Coffee Machine	Sound Level Changes Surface Vibrations	AUD, IMU, OPT

Tab. VI: Common Ambient Events available to HAEST



(a) Clock error distribution (b) Error Mean and Spread

Fig. 12: Intra-cluster clock errors in smart home study



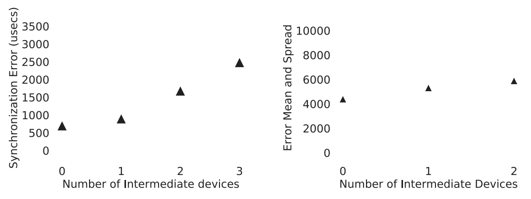
(a) Clock error distribution (b) Error Mean and Spread

Fig. 13: Inter-cluster clock errors in smart home study

involving FLORA. This is due to Flora's inability to perform accurate drift estimates, as we observed in Figure 7c. Better intra-cluster performance validates the use of our clustering approach to synchronize sensor networks.

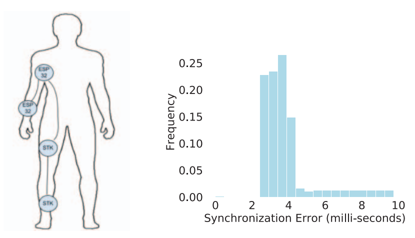
Intuitively, large inter-cluster error is due to accumulation of error via multiple intermediary devices. Figure 14a validates our intuition that devices connected directly are more tightly synchronized with each other than the devices connected over one or more hops. This result is in line with traditional multi-hop synchronization protocols where errors always accumulate over more number of hops. It is to be noted that error accumulation in HAEST is due to network effect.

Case Study 2: Wireless Body Area Network (WBAN). We also evaluate HAEST's performance under demanding conditions of a Wireless Body Area Network (WBAN) in Figure 15a. The network consists of 2 ESP on the person's right arm and a pair of STK devices on the right leg. Each



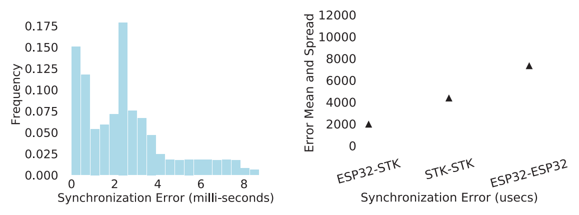
(a) Smart home case (b) WBAN case

Fig. 14: Effect of intermediary devices on clock error in (a) smart home and (b) WBAN deployment



(a) WBAN deployment (b) Pair-wise clock error between WBAN and smart home device

Fig. 15: (a) WBAN study; (b) distribution of pair-wise clock errors between static and a mobile device



(a) Clock error distribution (b) Error Mean and Spread

Fig. 16: HAEST performance in WBAN study

device is interfaced with an IMU operating at 200Hz. The network is structured as a single cluster. Sensing devices monitor person's indoor activities such as walk, exercise and desk work. Ambient events from these activities, especially physical motion in the form of walk and exercise, provide many synchronization opportunities to the devices. It is important to distinguish this network from smart home where all sensing devices are stationary, whereas WBAN devices are moving along-with body parts and their relative distance changes due to limb movements. In addition, the devices are exposed to a lot of background noise induced by slow, asynchronous or asymmetric body movements.

Most of the actions generating ambient events are from coordinated movements of various body parts, but even the highly coordinated movements are not ideally synchronized. This leads to larger variations in propagation delays observed by the devices in WBAN. These variations in propagation delays mean that errors between devices (i.e. ESP-ESP, STK-STK, ESP-STK) have higher magnitude as compared to smart home case. Under these challenging and noisy conditions, we are able to achieve a decent clock accuracy over multiple hops in WBAN as shown in Figure 14b. In addition, Figure 16 demonstrates that HAEST overcomes effects of environmental noise on clock errors in heterogeneous mobile deployments.

Case Study 3: Joint WBAN and Smart Home. Figure 15b shows the error between an IMU sensor installed on a person's leg as part of WBAN and a smart home sensing device with microphone at 200Hz. The two devices mutually observe events originating from a person's activity. In this case, person is walking around, interacting with various objects and doing desk work. This puts the two devices in relative motion. Events such as walking and moving objects occur in proximity of the microphone are mutually observed by the two devices. HAEST leverages these events between a mobile and a stationary device to achieve an error of several millisec. This experiment demonstrates that HAEST is robust to changing network deployments and device mobility.

B. *Robustness & Overhead*

Availability of Ambient Events. HAEST relies on ambient events for synchronization, which raises concerns regarding the events' availability. We argue that enough events are available in most smart spaces, as demonstrated by our case studies. Previous studies also show that sensing applications experience frequent ambient events. For example, PEEVS [66] and Fridman et al. [6] show that sensors in a smart office and autonomous cars experience frequent common events.

Deployment density. For devices to observe and timestamp mutual events, they should be located in proximity governed by the event and sensor type. This assumption is inline with existing (e.g. activity recognition [20]) and emerging (e.g. LiDAR-Camera perception [67]) distributed sensing applications which improve performance by relying on multiple sensors observing the same events. The deployment density in these sensor networks, by design, ensures they observe common ambient events and hence they are well suited to benefit from HAEST time-sync.

Overhead. By design, HAEST leverages the gateway's resources to synchronize time between IoT devices. As discussed in section III, devices only stream timestamped data to the gateway moving additional computational, communication or energy costs away from the IoT devices and to the gateway. From an IoT device's perspective, it can achieve time-sync without expending precious device resources. The biggest beneficiary of this would be battery powered devices which could see a significant increase in their battery lifetime when using HAEST as compared to traditional time-sync protocols. However, from the gateway's (RPi4) perspective, HAEST is somewhat expensive as it consumes approximately 50% of its compute resources. Most of this computational cost is from event detection, the building block of applications like human activity recognition, elderly fall detection, etc. And we can offset some of the HAEST's computational cost by sharing event detection with such applications albeit with a caveat i.e. our event detection approach has high false positive rates. While some upstream applications may not use it "out-of-the-box", their task would still be cut, as they do not need to process a continuous stream of sensor data but only the event candidates (to weed out false positives).

We also note that there are few upfront costs for setting up HAEST with new types of sensors, and use cases (determining event detection window N, drift window 10 min and k=20). However, these costs are akin to engineering efforts involved in any product development and should diminish for repeat deployments as there are finite types of sensors and use cases.

VII. Conclusion

We introduce sensing based time-sync approach for heterogeneous platforms called HAEST that relies on ambient events to align clocks without explicit exchange of time information, thus HAEST is not affected by communication delay uncertainty and moves overheads from the IoT devices to the gateway. Our smart home and WBAN evaluations show applicability of our design to a variety of platforms and deployments.

Extensibility. HAEST is a universal time synchronization approach for ubiquitous IoT devices. Though our evaluations are based on a variety of platforms with three most prevalent sensors, our approach is not dependent on any sensor type, rather sensors such as WiFi RSSI, power meters, and thermal cameras can be integrated into our system.

Acknowledgements

We thank the anonymous RTAS reviewers for their insightful comments and feedback. This research is supported by NSF grants 2237485 and 2230143.

References

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

[2] K. Lee, J. C. Eidson, H. Weibel, and D. Mohl, "Ieee 1588-standard for a precision clock synchronization protocol for networked measurement and control systems," in *Conference on IEEE*, vol. 1588, 2005, p. 2.

[3] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *SenSys, Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004.

[4] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 147–163, Dec. 2003. [Online]. Available: https://doi.org/10.1145/844128.844143

[5] M. T. Nyamukuru and K. M. Odame, "Tiny eats: Eating detection on a microcontroller," in *2020 IEEE Second Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML)*, 2020, pp. 19–23.

[6] L. Fridman, D. Brown, W. Angell, I. Abdić, B. Reimer, and H. Noh, "Automated synchronization of driving data using vibration and steering events," *Pattern Recognition Letters*, vol. 75, 10 2015.

[7] S. S. Sandha, J. Noor, F. M. Anwar, and M. Srivastava, "Time awareness in deep learning-based multimodal fusion across smartphone platforms," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2020, pp. 149–156.

[8] F. Mokaya, R. Lucas, H. Noh, and P. Zhang, "Burnout: A wearable system for unobtrusive skeletal muscle fatigue estimation," 04 2016, pp. 1–12.

[9] M. Meyer, T. Farei-Campagna, A. Pasztor, R. D. Forno, T. Gsell, J. Faillettaz, A. Vieli, S. Weber, J. Beutel, and L. Thiele, "Event-triggered natural hazard monitoring with convolutional neural networks on the edge," ser. IPSN '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 73–84. [Online]. Available: https://doi.org/10.1145/3302506.3310390

[10] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE network*, vol. 18, no. 4, pp. 45–50, 2004.

[11] ——, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, 2004.

[12] S. Karthik and A. A. Kumar, "Challenges of wireless sensor networks and issues associated with time synchronization," in *Proceedings of the UGC sponsored national conference on advanced networking and applications*, 2015, pp. 19–23.

[13] C. A. Chin, G. V. Crosby, T. Ghosh, and R. Murimi, "Advances and challenges of wireless body area networks for healthcare applications," in *2012 International Conference on Computing, Networking and Communications (ICNC)*, 2012, pp. 99–103.

[14] R. Nabiei, M. Najafian, M. Parekh, P. Jančovič, and M. Russell, "Delay reduction in real-time recognition of human activity for stroke rehabilitation," in *2016 First International Workshop on Sensing, Processing and Learning for Intelligent Machines (SPLINE)*, 2016, pp. 1–5.

[15] M. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, vol. 2, 2003, pp. 1266–1273 vol.2.

[16] (2023) Matter 1.0 core specification. [Online]. Available: https://csa-iot.org/all-solutions/matter/

[17] M. C. Dinescu, J. Mazza, A. Kujanski, B. Gaza, and M. Sagan, "U.S. Patent No. 61/276,266," 2010, https://patents.google.com/patent/US9002044.

[18] C. Chen, S. Rosa, C. X. Lu, N. Trigoni, and A. Markham, "Selectfusion: A generic framework to selectively learn multisensory fusion," *arXiv preprint arXiv:1912.13077*, 2019.

[19] J. Clemente, F. Li, M. Valero, and W. Song, "Smart seismic sensing for indoor fall detection, location, and notification," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 2, pp. 524–532, 2020.

[20] S. Sandha, J. Noor, F. Anwar, and M. Srivastava, "Time awareness in deep learning-based multimodal fusion across smartphone platforms," 04 2020, pp. 1–8.

[21] S. Patel, K. Lorincz, R. Hughes, N. Huggins, J. Growdon, M. Welsh, and P. Bonato, "Analysis of feature space for monitoring persons with parkinson's disease with application to a wireless wearable sensor system," *Conference proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 2007, pp. 6291–4, 02 2007.

[22] K. Lorincz, B.-r. Chen, G. Challen, A. Roy Chowdhury, S. Patel, P. Bonato, and M. Welsh, "Mercury: A wearable sensor network platform for high-fidelity motion analysis," 01 2009, pp. 183–196.

[23] Z. Yu, C. Jiang, Y. He, X. Zheng, and X. Guo, "Crocs: Cross-technology clock synchronization for wifi and zigbee," ser. EWSN '18. USA: Junction Publishing, 2018, p. 135–144.

[24] N. Shivaraman, P. Schuster, S. Ramanathan, A. Easwaran, and S. Steinhorst, "Cluster-based network time synchronization for resilience with energy efficiency," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 149–161.

[25] A. Rowe, V. Gupta, and R. Rajkumar, "Low-power clock synchronization using electromagnetic energy radiating from ac power lines," 01 2009, pp. 211–224.

[26] Y. Li, R. Tan, and D. K. Yau, "Natural timestamping using powerline electromagnetic radiation," in *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2017, pp. 55–66.

[27] Z. Yan, Y. Li, R. Tan, and J. Huang, "Application-layer clock synchronization for wearables using skin electric potentials induced by powerline radiation," 09 2017.

[28] J. V. Jeyakumar, L. Lai, N. Suda, and M. Srivastava, "Sensehar: A robust virtual activity sensor for smartphones and wearables," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, ser. SenSys '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 15–28. [Online]. Available: https://doi.org/10.1145/3356250.3360032

[29] K. J. Oh and K. jae Kim, "Analyzing stock market tick data using piecewise nonlinear model," *Expert Systems with Applications*, vol. 22, no. 3, pp. 249–255, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417401000586

[30] A. Hasan, W. Ning, and A. K. Gupta, "An information-based approach to the change-point problem of the noncentral skew t distribution with applications to stock market data," *Sequential Analysis*, vol. 33, no. 4, pp. 458–474, 2014. [Online]. Available: https://doi.org/10.1080/07474946.2014.961842

[31] J. Reeves, J. Chen, X. L. Wang, R. Lund, and Q. Q. Lu, "A review and comparison of changepoint detection techniques for climate data," *Journal of Applied Meteorology and Climatology*, vol. 46, no. 6, pp. 900 – 915, 2007. [Online]. Available: https://journals.ametsoc.org/view/journals/apme/46/6/jam2493.1.xml

[32] K. Yamanishi and J.-i. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 676–681. [Online]. Available: https://doi.org/10.1145/775047.775148

[33] M. Basseville, I. V. Nikiforov *et al.*, *Detection of abrupt changes: theory and application*. prentice Hall Englewood Cliffs, 1993, vol. 104.

[34] R. Zhang, Y. Hao, D. Yu, W.-C. Chang, G. Lai, and Y. Yang, "Correlation-aware unsupervised change-point detection via graph neural networks," 2020.

[35] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long- and short-term temporal patterns with deep neural networks," 2018.

[36] T. Chowdhury, M. Aldeer, S. Laghate, and J. Ortiz, "Cadence: A practical time-series partitioning algorithm for unlabeled iot sensor streams," *arXiv preprint arXiv:2112.03360*, 2021.

[37] J. Oostvogels, S. Michiels, and D. Hughes, "One-take: Gathering distributed sensor data through dominant symbols for fast classification," in *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2022, pp. 337–349.

[38] L. Klingbeil and T. Wark, "A wireless sensor network for real-time indoor localisation and motion monitoring," in *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, 2008, pp. 39–50.

[39] F. Gustafsson, "The marginalized likelihood ratio test for detecting abrupt changes," *IEEE Transactions on Automatic Control*, vol. 41, no. 1, pp. 66–78, 1996.

[40] M. Sugiyama, S. Nakajima, H. Kashima, P. Buenau, and M. Kawanabe, "Direct importance estimation with model selection and its application to covariate shift adaptation," *Advances in neural information processing systems*, vol. 20, 2007.

[41] S. Bickel, M. Brückner, and T. Scheffer, "Discriminative learning for differing training and test distributions," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 81–88.

[42] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, "Covariate shift by kernel mean matching," *Dataset shift in machine learning*, vol. 3, no. 4, p. 5, 2009.

[43] M. Sugiyama, T. Suzuki, and T. Kanamori, *Density ratio estimation in machine learning*. Cambridge University Press, 2012.

[44] W.-H. Lee, J. Ortiz, B. Ko, and R. Lee, "Time series segmentation through automatic feature learning," *arXiv preprint arXiv:1801.05394*, 2018.

[45] J. Hester, N. Tobias, A. Rahmati, L. Sitanayah, D. Holcomb, K. Fu, W. P. Burleson, and J. Sorber, "Persistent clocks for batteryless sensing devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 4, pp. 1–28, 2016.

[46] S. S. Sandha, J. Noor, F. M. Anwar, and M. Srivastava, "Exploiting smartphone peripherals for precise time synchronization," in *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, 2019, pp. 1–6.

[47] C. G. Ramirez, A. Sergeyev, A. Dyussenova, and B. Iannucci, "Longshot: long-range synchronization of time," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, 2019, pp. 289–300.

[48] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 81–94.

[49] X. Guo, X. Zheng, and Y. He, "Wizig: Cross-technology energy communication over a noisy channel," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[50] X. Guo, Y. He, X. Zheng, L. Yu, and O. Gnawali, "Zigfi: Harnessing channel state information for cross-technology communication," *IEEE/ACM Transactions on Networking*, vol. 28, no. 1, pp. 301–311, 2020.

[51] M. Buevich, N. Rajagopal, and A. Rowe, "Hardware assisted clock synchronization for real-time sensor networks," in *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 2013, pp. 268–277.

[52] S. Viswanathan, R. Tan, and D. K. Yau, "Exploiting power grid for accurate and secure clock synchronization in industrial iot," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 146–156.

[53] Z. Li, W. Chen, X.-Y. Li, and Y. Liu, "Flight: clock calibration using fluorescent lighting," 08 2012.

[54] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[55] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz *et al.*, "A public domain dataset for human activity recognition using smartphones." in *Esann*, vol. 3, 2013, p. 3.

[56] A. Mesaros, T. Heittola, E. Benetos, P. Foster, M. Lagrange, T. Virtanen, and M. D. Plumbley, "Detection and classification of acoustic scenes and events: Outcome of the dcase 2016 challenge," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 2, pp. 379–393, 2018.

[57] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[58] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[59] (2020) Ti cc2650stk. [Online]. Available: https://www.ti.com/tool/CC2650STK

[60] (2020) Sparkfun esp32 things. [Online]. Available: https://www.sparkfun.com/products/13907

[61] (2020) Adafruit flora. [Online]. Available: https://www.adafruit.com/product/659

[62] S. Rinaldi, D. Della Giustina, P. Ferrari, A. Flammini, and E. Sisinni, "Time synchronization over heterogeneous network for smart grid application: Design and characterization of a real case," *Ad Hoc Networks*, vol. 50, pp. 41–57, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157087051630097X

[63] R. Solis, V. S. Borkar, and P. R. Kumar, “A new distributed time synchronization protocol for multihop wireless networks,” in *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006, pp. 2734–2739.

[64] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, “Exploiting a natural network effect for scalable, fine-grained clock synchronization,” in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 81–94. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/geng

[65] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 138–149. [Online]. Available: https://doi.org/10.1145/958491.958508

[66] S. Birnbach, S. Eberz, and I. Martinovic, “Peeves: Physical event verification in smart homes,” 11 2019, pp. 1455–1467.

[67] R. S. Hallyburton, Y. Liu, Y. Cao, Z. M. Mao, and M. Pajic, “Security analysis of Camera-LiDAR fusion against Black-Box attacks on autonomous vehicles,” in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1903–1920. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/hallyburton

APPENDIX

The auto-encoder, as the name suggests learns its own encodings. The encoder takes in the input data and maps it to a latent vector and the decoder takes this latent vector and regenerates the input data. So, auto-encoders are trained in an unsupervised manner as they use the same data as their input and output.

For our base auto-encoder implementation, both the encoder and decoder consist of two hidden layers. LeakyReLU is the activation function for all layers except the last layer, which uses tanh instead. For the encoder, the input layer is the size of the sensor data window chosen for the training. The intermediate layer is half the size of the input layer while the last layer is one-tenth the size of the input. The decoder's intermediate layer is the same size as the encoder's intermediate layer. The decoder's input layer has a size equal to the encoder's output and the output layer's size is the same as the encoder's input. We use Adam optimizer with a learning rate of 10^{-5} to train the auto-encoder for 200 epochs.

To train the sensor-specific layer, we use a base auto-encoder trained on combined IMU and audio datasets. We freeze the base encoder and add an extra layer at the output with a size equal to half the base encoder's input. This is our sensor-specific layer. Similarly, we freeze the base decoder but this time we add the extra layer before the base decoder's input. Then this new auto-encoder is trained using the same settings that were used to train the base auto-encoder. Since the base encoder and decoders are frozen, the sensor-specific layer learns to predict sensor data encodings from the base encoder's output features. For HAEST to be lightweight, the sensor-specific layer should have a small size. To put our sensor layer's size in perspective, we compare its size to the base encoder's size. If our base encoder's input size is 2400, the intermediate layer size would be 1200. The first layer will have $2400x1200 = 2880000$ parameters to learn. The base encoder's last layer's size would be 240, which translates into $1200x240 = 288000$ parameters. The sensor-specific layer's size, in this case, would be 1200 making its learnable parameters the same as that of the base encoder's last layer. The sensor-specific layer's weights are only 9.1% of the base encoder's weights.

We evaluate our event detection approach using two public datasets; i) DCASE 2016 [56] synthetic audio event detection data and ii) UCI Human activity recognition data-set [55], which consists of 3-axial accelerometer and gyroscope measurements for participants performing different activities. An input size of 2400 timesteps for Dcase2016 and 400-time steps of the UCI dataset is used in training and inputs are normalized between $(-1, 1)$. Finally, we train auto-seconders on a server (Core i7-9700K+2080Ti) device equipped with GPU.

To measure the performance of our approach, we use the area under the receiver operating curve (ROC) as the metric. ROC is obtained by plotting the true positive rate (TPR) and false positive rate (FPR) which are calculated as $TPR = \frac{nCP}{nGT}$ and $FPR = \frac{nALL-nCP}{nALL-nCP+TN}$ respectively. Here, nCT and $nALL$ are the correct and total number of events predicted by our algorithm, respectively. nGT is the total number of events in the dataset, while TN are the true negatives. TN is determined by time-series length divided by N_w. A prediction is determined to be correct if the time difference between the predicted event and the ground truth event is less than τ. ROC curve is obtained by varying the values of τ.