Adaptive Uplink Data Compression in Spectrum Crowdsensing Systems

Yijing Zeng, Roberto Calvo-Palomino*, Domenico Giustiniano§, Gerome Bovet‡, Suman Banerjee
UW-Madison, *Universidad Rey Juan Carlos, §IMDEA Networks, ‡armasuisse
{yijingzeng, suman}@cs.wisc.edu, *roberto.calvo@urjc.es, †domenico.giustiniano@imdea.org, ‡Gerome.Bovet@armasuisse.ch

***Abstract*—Understanding spectrum activity is challenging when attempted at scale. The wireless community has recently risen to this challenge in designing spectrum monitoring systems that utilize many low-cost spectrum sensors to gather large volumes of sampled data across space, time, and frequencies. These crowdsensing systems are limited by the uplink bandwidth available for transmitting to the backhaul network raw in-phase and quadrature (IQ) samples and power spectrum density (PSD) measurements needed to run a variety of applications. This paper presents FlexSpec, a framework based on the Hadamard-Walsh transform to compress spectrum data collected from distributed and low-cost sensors for real-time applications. We show that this transformation allows sensors to greatly save uplink bandwidth thanks to its inherent properties both when it is applied to IQ and PSD measurements. Additionally, by leveraging a feedback loop between an edge device and the sensor, FlexSpec carefully adapts the compression ratio over time, such that data size, application's performance, and spectrum variations are all considered. We experimentally evaluate FlexSpec in several applications. Our results show that FlexSpec is particularly suitable for IoT transmissions and for signals close to the noise floor. Compared with prior work, FlexSpec provides up to 7× more reduction of uplink data size for signal detection based on PSD data, and reduces up to 6× to 8× the number of undecodable messages for IQ sample decoding.**

I. Introduction

Measuring and understanding spectrum activity at scale — time, space, and frequency — is challenging. While traditionally a few high-end spectrum sensors have been used to understand such activity, in recent years the community has devoted effort in building low-cost spectrum sensors that can potentially be deployed at a much greater scale by many users, e.g. [12], [17], [9]. These crowdsourced spectrum sensing systems have an architecture where different types of spectrum data can be continuously obtained from a large number of very low-cost spectrum sensors (e.g., the $\sim$ \$25 RTL-SDR) and aggregated in cloud-hosted backends for further analytics by different applications. A high level illustration of the architecture is presented in Fig. 1, with low-cost sensors that transmit their data to edge devices, and several applications that can be selected by the end user. In particular, the low-cost nature of the sensor and its easy integration into an existing cloud-hosted backend often motivates more participants to join this effort for the greater good of the community.

The two common types of spectrum data of interest to various use cases are power spectral density (PSD) data and in-phase and quadrature (IQ) data. The former is useful to

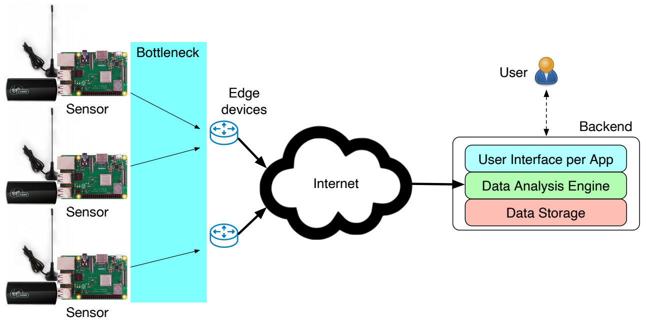


Fig. 1. Illustration of spectrum crowdsensing system supporting a variety of applications; the bottleneck is the backhaul connection from the sensors.

various applications in understanding spectrum utilization and occupancy, while the latter helps analyze wireless protocol behaviors and decode messages. For each data form, the sensor can quickly generate a large volume of data, which eventually needs to be backhauled and stored remotely. For instance, the RTL-SDR generates 2.4M IQ samples per second which results in a data rate of 153.6 Mbps per sensor, often beyond the capacity of any common backhaul network.

Prior efforts have proposed different spectrum data compression methods. Electrosense leverages lossless compression to pack data more compactly [12], [11]. Unfortunately, this approach is unable to compress much due to the nature of spectrum data. Airpress [18] and SparSDR [7] propose lossy compression methods for PSD and IQ data, respectively. Although they achieve greater compression ratio, they either (i) depend on setting a specific threshold to determine whether there is wireless activity and whether such samples should be uploaded or (ii) use a fixed compression ratio that does not allow the application to adapt to changes in the received spectrum data. As we will show in § II, this causes missing important information in the compressed spectrum data, and lack of adaptation with respect to the application needs, which is especially relevant both for Internet of Things (IoT) wireless transmissions and signals close to the noise floor.

Adaptive Compression of Spectrum Data: We propose a framework, FlexSpec, which performs adaptive compression of spectrum data that are uploaded from distributed spectrum sensors in such crowdsensing systems. FlexSpec addresses the limitations of prior work and is based on two components, a lightweight real-time lossy compression method and an application-oriented compression adaption scheme. Our real-time lossy compression runs in the sensor with feedback received from the edge device.

FlexSpec's compression method is based on fast Walsh-Hadamard transformation (FWHT). FWHT has been applied in a variety of other fields. It received large attention in the past because its low-computational cost and simplicity of transform bases. For spectrum data compression, our key idea is that we *apply this transformation on single spectrum measurement* (either PSD or IQ) and only *retain significant readings in the transferred domain*. To the best of our knowledge, we are the first to apply FWHT on spectrum data compression and show why it works well on both PSD and IQ data. The core reason is that the transferred domain of either PSD or IQ data captures the features of the data. The bases of FWHT capture the bandwidth limited feature of signals in PSD data, and the transferred domain for the IQ data is a variant of the frequency domain. In addition, this method has a highly adjustable compression ratio, meaning the more appropriate choice is available when adapting the compression ratio.

In FlexSpec, the amount of compression to be applied depends specifically on the application of interest. Consider an application like spectrum utilization measurements over a wide area which operates over PSD data. The samples in this setting can be compressed in a lossy manner quite substantially because the results can gracefully degrade with loss in fidelity of the data, i.e., if the uplink capacity is low and the energy measurements have reduced fidelity, the application can still provide reasonable estimates from them. However, if we consider another application — decoding broadcast ADS-B messages from aircraft transponders that allow them to be tracked — it might not be equally amenable to reduce fidelity of information; therefore, aggressively lossy compression might not be suitable in this case. Consequently, FlexSpec aims to support dynamic selection of the compression ratio to support a variety of spectrum applications according to the user's interests and is suited to work with live streaming of spectrum data (consumed by the running application).

FlexSpec introduces an application-oriented compression ratio adaptation scheme to enable dynamic compression ratio adaptation for different applications as well as changes in the captured spectrum. The essence of this scheme is that, given the desired application performance metric(s), we *introduce a feedback loop between an edge device and the sensor to jointly decide the compression ratio based on the target application's performance on current compressed spectrum data*. As a result, FlexSpec can consistently optimize the desired application performance but send as a little amount of data as possible in a varying spectrum band, which is impossible for prior work.

We evaluate FlexSpec through extensive experiments. The results show that, for signal detection based on PSD data, FlexSpec misses less than 1 FM channel (out of 9 total) in the FM radio band when there is spectrum activity, and has 7× more reduction in uplink data size when there is no spectrum activity compared with prior work. For IQ decoding, FlexSpec reduces 6× to 8× the number of undecodable messages compared with prior work with a similar compression ratio.

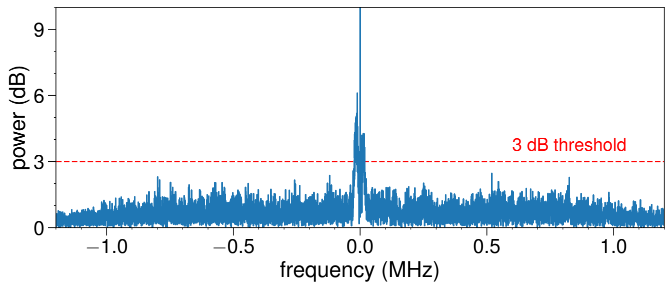


Fig. 2. PSD of a sample ADS-B signal. The bandwidth of the signal determined by the 3 dB threshold above the noise level is 80 KHz, but important information is lost with this threshold.

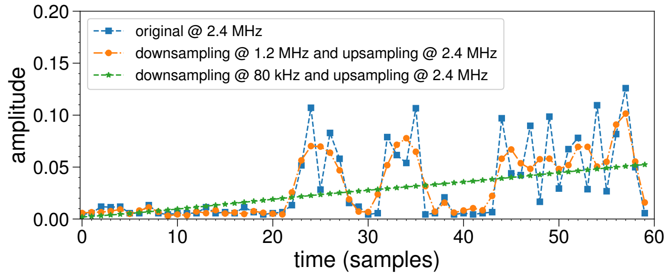


Fig. 3. Amplitude of ADS-B signal over time. Downsampling causes detail information loss, which makes ADS-B signal hard to be decoded.

II. Challenges

Lossy compression of spectrum data on low-cost sensors, e.g. RTL-SDR [4] with Raspberry Pi 4 (RBPi-4) [2], could allow to save large data volume, but raises the following challenges to the spectrum data compression techniques.

Limited computing resources. Low-cost sensors usually have a relatively small amount of memory (8 GB for RBPi-4) and only a few CPU cores (4 cores for RBPi-4). Therefore, unlike prior work with high computational cost (e.g. [16]), the compression technique in our case should have low time/space complexity such that it can compress streaming spectrum data in the resource-limited low-cost sensors in real-time.

Spectrum occupancy and wideband signals. Wireless signals have bandwidth B that can vary largely. In contrast, low cost-sensors have limited support in terms of sampling rate S. This has implications on the compression ratio strategy. For narrowband signals ($S \geq B$), methods that estimate *spectrum occupancy* using a power threshold in the frequency domain have been used to select the active frequency bins to be backhauled, as done in prior work by SparSDR [7]. While this idea is simple, measuring the spectrum occupancy is not effective in a number of cases. For instance, the report from the International Telecommunication Union (ITU) on spectrum occupancy suggests to declare the channel as busy if it is 3-5 dB above the noise level [14]. Fig. 2 shows the PSD of a ADS-B signal trace sent by an aircraft and captured by RTL-SDR with 2.4 MHz sampling rate. Following the above ITU report, its spectrum occupancy would be 80 KHz. However, the signal cannot be decoded at all with this 80 KHz band. The reason is that the modulation used in ADS-B causes leakages outside the 80 kHz band and near the noise floor, yet fundamental for message decoding. Besides, even in those cases when spectrum occupancy measurement in SparSDR is effective on narrowband signals, the output sequence could be further compressed depending on the requirements of the application, a chance ignored by past work. On the other hand, the captured signals can be wideband ($S < B$) for numerous cases, as S is relatively small in low-cost sensors (e.g. $S = 2.4\,\text{MHz}$ in RTL-SDR). In this case, spectrum occupancy is not effective at all, as all frequency bins must be backhauled. In summary, a different approach is needed that does not depend on spectrum occupancy measurements.

Detail information is important for IQ data. Airpress compresses the PSD data with the goal of spectrum summarization [18]. It applies Haar wavelet transform to get the approximation and detail coefficients, and assumes that detail coefficients are not important for spectrum summarization. Although this assumption is valid for PSD data, we find that it becomes invalid for IQ data. For example, if we would like to decode IQ samples of FM radio signal with reasonable SNR, the minimal possible compression ratio of 2 will directly lead to noisy audio that cannot be perceived by human. Therefore, for IQ data, we need a method that balances approximation and detail during the compression. Note that a byproduct of completely omitting detail coefficients in Airpress is that it can only support exponential compression ratios in the form of 2^i ($i = 0, ..., n$), given a PSD data with $N = 2^n$ readings. The limited number of choices in compression ratios makes fine-grained compression ratio adaptation impractical. By balancing approximation and detail, we can potentially support linear compression ratios in the form of N/i ($i = 1, ..., N$) for fine-grained compression ratio adaptation. In addition, downsampling, a naive approach to compress IQ data, also loses the detail information. For instance, Fig. 3 shows the amplitude of reconstructed ADS-B signal after downsampling/upsampling, and compares with the original signal. We can see that after downsampling to 1.2 MHz, the preambles needed to decode ADS-B signal, which last from time 20 to 40, are already distorted. The four spikes becomes two. If the signal is downsampled to 80 kHz as the spectrum occupancy result suggested, not only the preambles are distorted, but the signal duration that carries data bits become meaningless.

Difficulty in deciding the compression ratio. Lossless compression does not save much data volume due to its inability to capture the properties of the noisy spectrum data. Our experience with Gzip (used in Electrosense [12]) shows that it can only reduce data size by 10% for IQ data. Lossy compression can instead allow to balance the information loss and compression ratio of the lossy compression algorithm. However, how much information loss is tolerable highly depends on the specific application that is streamed at the current time. For example, if the application is to know whether a channel is occupied based on PSD data, it tolerates relatively high information loss. If the application is to decode aircraft messages from IQ samples, we may only tolerate little information loss. In addition, significant utilization change of the spectrum band e.g. from idle to active, and discrepancy of activities in different bands may also result in the difference of tolerable information loss. Prior work did not study this problem.

III. Compression Algorithm

In this section, we present FlexSpec's spectrum data compression algorithm. Our method applies a low-cost transformation, fast Walsh-Hadamard transform (FWHT), on single spectrum measurement and only keeps significant readings in the transferred domain, where both the features of PSD and IQ data can be captured. We first review the principles of Walsh-Hadamard transform, and then present our compression method and explain the ideas behind its efficacy for spectrum data compression.

A. *Primer on Walsh-Hadamard transform*

Walsh-Hadamard transform [13] is a generalized form of Fourier transform. Different from Fourier transform, which uses complex sine and cosine functions as the orthogonal basis, Walsh-Hadamard transform uses real Walsh functions as the orthogonal basis, which contains only -1 and 1. Formally, let H_{2^n} denote the Hadamard matrix with row/column size equals 2^n and $H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, we have the following relationship between $H_{2^{n+1}}$ and H_{2^n}:

$$H_{2^{n+1}} = \begin{bmatrix} H_{2^n} & H_{2^n} \\ H_{2^n} & -H_{2^n} \end{bmatrix}, n = 1, 2, 3, \ldots . \quad (1)$$

Therefore, each row/column of H_{2^n} is a Walsh function, which contains only -1 and 1, and is orthogonal to each other. Let the data with $N = 2^n$ readings as a row vector $\mathbf{x}$, its Walsh-Hadamard transform is defined as

$$\mathbf{X} = \mathbf{x} \cdot H_{2^n}. \quad (2)$$

The corresponding inverse transform is computed as

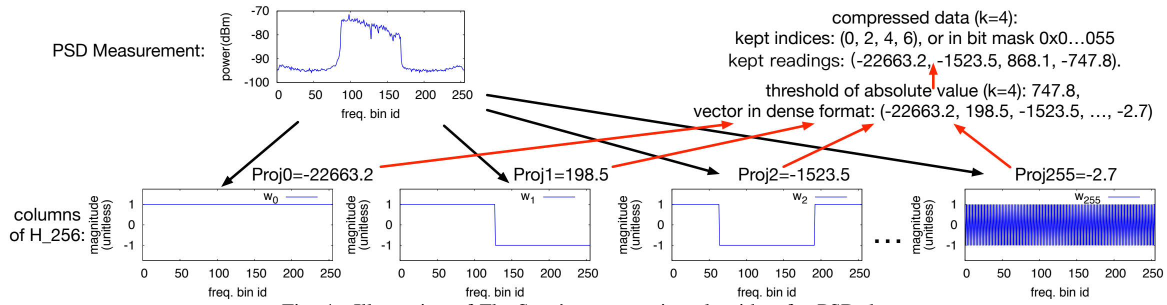
$$\mathbf{x} = \frac{1}{N} \mathbf{X} \cdot H_{2^n}. \quad (3)$$

Similar to Fourier transform, which has a fast butterfly algorithm with $O(N\log N)$ time complexity, i.e. fast Fourier transform (FFT), Walsh-Hadamard transform also has a fast butterfly algorithm with $O(N\log N)$ time complexity, i.e. FWHT. The difference is that FFT involves $N\log_2 N$ complex add operations and $\frac{1}{2}N\log_2 N$ complex multiplication operations, but FWHT only involves $N\log_2 N$ real add operations.

B. *PSD Data Compression*

Given PSD data $\mathbf{x}$ of length N and the number of kept readings k, we can compress $\mathbf{x}$ as follows. The idea is that we ignore the readings in the transferred domain close to 0 and encode this dense vector in the transferred domain as a sparse vector[1]. More specifically, we compute the FWHT of the PSD data and then record the absolute values in the transferred

[1]A dense vector is an array that contains mostly non-zero values, and a sparse vector is an array that contains mostly zeros. For the representation, a dense vector presents all values of the vector, while a sparse vector is described as two parallel arrays, indices and values.


PSD Measurement:
power(dBm)
freq. bin id
compressed data (k=4):
kept indices: (0, 2, 4, 6), or in bit mask 0x0…055
kept readings: (-22663.2, -1523.5, 868.1, -747.8).
threshold of absolute value (k=4): 747.8,
vector in dense format: (-22663.2, 198.5, -1523.5, …, -2.7)
Proj0=-22663.2
Proj1=198.5
Proj2=-1523.5
Proj255=-2.7
columns of H_256:
magnitude (unitless)
w0
w1
w2
w255
Fig. 4. Illustration of FlexSpec's compression algorithm for PSD data.

domain. Next, we get the kth largest absolute value of readings in the transferred domain using Quick Select [6] algorithm, where k is selected by the running application. Last, we get all the kept readings in the transferred domain and encode them as a sparse vector. Note that we use bit mask to represent the indices of kept readings when compression ratio is smaller than 16, which saves even more space. To decompress, one need to first convert the compressed data into the dense format and then compute the inverse transform using Eq. (3).

C. IQ Data Compression

We extend our compression algorithm on PSD data to IQ data by applying the algorithm to in-phase (I) reading sequence and quadrature (Q) reading sequence separately. Other options are possible too, e.g., directly use complex FWHT, convert I and Q coordinates to amplitude and phase polar coordinates first then apply the algorithm to amplitude reading sequence and phase reading sequence separately. Although the best option in terms of compression ratio varies for different modulation scheme of the signal (e.g. if the signal is purely phase modulated, convert I and Q coordinates to amplitude and phase polar coordinates is very desirable), our method (1) has lower time complexity so that it can work on low-cost sensors in real-time because it involves real number addition and subtraction only, and (2) the transferred domain is a variant of the frequency domain, called sequency domain [5], so it captures the features of IQ data in this analogous frequency domain.

D. Suitability for Spectrum Compression

Although FWHT is commonly applied in data compression for example image compression [15] and video compression [8], our main contribution is to show why it also works well on spectrum data, for the first time. At a high level, FWHT can capture the features of both PSD and IQ data. In fact, the bases of the transferred domain for PSD data represent one or more (frequency domain) bandwidth-limited signals (c.f. Fig. 4), and the transferred domain of IQ data is a variant of the frequency domain.

PSD. Fig. 4 illustrates how FlexSpec's compression algorithm captures the features of a single PSD measurement with 256 frequency bins. In Fig. 4, we reorder the columns of H_{256} so that each column/Walsh function, denoted as $\mathbf{w}_i, 0 \leq i \leq 255$, has i sign changes. We notice that, $\mathbf{w}_0$ and the corresponding Projection0 represent the average power of the PSD measurement, which highly depends on the noise floor value. Starting from $\mathbf{w}_1$, each Walsh function equals to 1 for half of the frequency bins and equals to -1 for the other half. Therefore, if the projection on $\mathbf{w}_i$ is bigger than 0, then it means there are probably notable signal(s) in the frequency bins whose corresponding values are 1. Otherwise, there are probably notable signal(s) in the frequency bins whose corresponding values are -1. For example, if Projection1 is bigger than 0, then there are probably notable signal(s) from frequency bin 0 to 127. If Projection2 is smaller than 0, then there are probably notable signal(s) from frequency bin 64 to 191. In both cases, the bases represent one or more bandwidth-limited signals no matter whether the projection is positive or not. Moreover, because we keep the projections that have largest absolute values rather than keep the projections whose corresponding Walsh functions have a fewer number of sign changes, our compression algorithm can not only preserve relatively wideband signals, e.g. ones captured by $\mathbf{w}_1$ and $\mathbf{w}_2$, but also narrowband signals, e.g. ones captured by $\mathbf{w}_{255}$.

Compared with Airpress [18], our compression algorithm supports more possible compression ratios (any value in the form of $N/i, i = 1, \ldots, N$), which is better suited for adaptive compression scenario. Our FWHT based compression on PSD data also brings two additional advantages over Airpress. First, because in Airpress, all approximation coefficients are retained but all detail coefficients are ignored after Haar wavelet tranform, our method can capture more detailed information when the same compression ratio is used, i.e., less information loss. Second, our algorithm is more friendly for narrowband signals. The reason is that in Airpress each PSD reading is approximated by the corresponding approximation coefficient, which is the average of consecutive 2^l (l is the level of transformation) PSD readings, so the bandwidth of narrowband signals can be significantly enlarged but the power of them can be drastically reduced.

IQ. Besides capturing features in the sequency domain, we consider our compression algorithm as a good methodology because of the following reasons. First, as FWHT's orthogonal bases are square waves rather than sine waves, our algorithm will retain fewer projections for signals whose baseband waveform is more similar to square waves, compared with FFT-based compression. Second, compared with complex number FFT, FWHT on I and Q readings respectively saves more than half number of real valued operations (including addition or multiplication), which means it has higher time

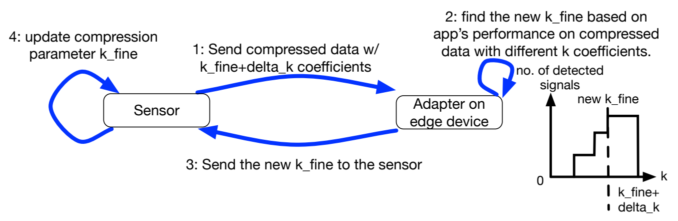

4: update compression parameter k_fine
1: Send compressed data w/ k_fine+delta_k coefficients
2: find the new k_fine based on app's performance on compressed data with different k coefficients.
Sensor
Adapter on edge device
3: Send the new k_fine to the sensor
no. of detected signals
new k_fine
k
0
k_fine+ delta_k

Fig. 5. Illustration of FlexSpec's application-oriented compression ratio adaptation scheme.

efficiency and is a better fit for low-cost devices. Last, we offer an opportunity to compress I and Q readings separately. Although I and Q readings are usually equally important in traditional modulation schemes, recent advances in deep learning based communication design show that in a learning based modulation scheme, the constellation points can be asymmetric around the origin point [10], so the importance of I and Q readings can differ from each other. In addition, the relative importance of each basis within I and Q readings can also differ from each other, as modulation schemes may exploit I and Q readings differently. In these cases, compressing I and Q readings separately is more beneficial compared with compressing them together.

IV. System Overview

FlexSpec is designed to efficiently compress data transmitted in real-time by the sensor and used by the application specified by the user. We define the user as any person with access to the applications implemented by the system. In our view, the user will have the control of the sensor[2], and dynamically select a specific application to run. Although FWHT based compression could be applied to historical data in the backend too[3], for live consumption of data, additional optimization in compression ratio is possible. We observe that *the tolerable information loss varies for different applications, and changes in spectrum utilization also lead to variations of tolerable information loss,* which is the basis of compression ratio adaptation. Previous works do not explicitly consider adaptive compression because they are designed for specific applications. In this section, we show how compression ratio can be dynamically adapted in FlexSpec so that the information loss can be adjusted to the variations of spectrum utilization and satisfy the requirement of different applications.

Overall, FlexSpec's application-oriented compression ratio adaptation scheme works as follows. *We introduce a feedback loop between an adapter and the sensor to jointly decide the compression coefficient k based on the application's performance on current compressed data.* The adapter decides and sends the appropriate compression coefficient k_{fine} to be used by the sensor given the received compressed data (we will explain how to decide it shortly). Then, in order to be aware of changes in the spectrum utilization, the sensor adds some redundancy to the uplink data, i.e., it uploads the compressed data with $k_{fine} + \Delta k$ coefficients. The amount redundancy added is rich enough to affect application's performance when the utilization of the spectrum is changed. Given the compressed data with $k_{fine} + \Delta k$ coefficients, the adapter decides the new k_{fine} based on the application's performance. It outputs the smallest k, such that the difference between the application's performance on the compressed data with k coefficients and that on the compressed data with $k_{fine} + \Delta k$ coefficients is tolerable in terms of some performance metrics specified by the user, e.g., the number of detected signals for signal detection application. Fig. 5 illustrates this scheme. Note that naively applying feedback loop does not work. This is because the performance of the application on the compressed data with a larger compression coefficient k cannot be inferred based on the current compressed data, which means the feedback loop cannot decrease the compression ratio. As a result, adding a small redundancy in our proposed feedback loop is critical.

Initialization. During the initialization phase, if the uplink cannot handle uncompressed data, the sensor can be configured to compute the initial compression coefficient to be used, k_{coarse}, by itself. A simple example of computing k_{coarse} can be having at most 5 dB error for every frequency bin between reconstructed data and the uncompressed data for PSD. For IQ data, an example is to directly start with a compression ratio of 2.

Reduce computation cost at the adapter. When the number of sensors that an adapter controls is large, there is a need to further reduce the computation cost at the adapter. A simplified version of the adapter's logic is to only evaluate three options for the new compression ratio, which are $k_{fine} - \Delta k, k_{fine}$, and $k_{fine} + \Delta k$. This simplified logic can be parallelized easily.

Reduce fluctuations. The adapter's logic computes k_{fine} on every compressed data sent from the sensor. However, the result may fluctuate (usually towards a more aggressive compression coefficient) due to some random noise in the data. Therefore, we reduce the fluctuations (as well as the downlink traffic) and output the new k_{fine} on every m compressed data records (usually around 10). The adapter selects the largest one (most conservative one) from the m computed new compression coefficients because the logic at the adapter's is relatively aggressive (see the limitation below).

Deciding Δk. Having a large Δk needs a higher uplink data rate but it can handle bursty changes of spectrum utilization. One can determine Δk by replaying a synthetic data trace, which adds more significant variations to a real-world (uncompressed) data trace, to the system before supporting live streaming data, and choose from $1/2^i$ of the segment size. In practice, we find Δk within the range of 1/8 to 1/16 of the segment size works well in balancing the two sides for the applications we tested.

Multiple application performance metrics. It is straightforward to incorporate multiple application performance metrics. For example, we can easily add the center frequency, the

[2]Given the limited hardware and software capabilities of the spectrum sensors, we do not consider the extension to multiple users controlling the same sensor realistic.

[3]In case the same data was already compressed based on the application needs, storing this data for historical processing could use the same or higher ratio with respect to live consumption of data.

bandwidth of detected signal(s), etc. into the toy example of signal detection application in Fig. 5 by outputting the highest k_{fine} for different metrics. The computations of k_{fine} for different metrics are also parallelized.

Limitation. A limitation of this scheme is that there is no theoretical guarantee between the gap of the application's performance on compressed data and that on uncompressed data. In fact, the adapter can output a very aggressive compression coefficient such that the redundancy added is not enough. Further, uploading uncompressed data, even only for a small portion of all the traffic, is undesirable or, depending on the available bandwidth, even impossible. This means we do not have access to the ground truth of the application's performance to calibrate our system. If this assumption is relaxed, we can upload a small portion of data uncompressed to calibrate the application performance. However, we empirically find that the application's performance always converges to the application's performance on uncompressed data without a significant gap in our system, as long as Δk is configured as previously described. Furthermore, this method applies to live consumption of data only.

Benefits. The only previous work that involves a certain degree of compression ratio adaptation is SparSDR. Based on the assumption that the signal of interest is always sparse (in frequency domain), SparSDR applies a fixed energy threshold and filters out readings with insufficient energy. Consequently, it implicitly controls the compression ratio as it considers the activities in the spectrum. However, as discussed in § II, the sparsity assumption is not always valid, especially for spectrum measurement using low-cost sensors, which usually has a relatively narrow bandwidth. Furthermore, it completely misses the opportunity provided by the nature of different applications. We take a step forward by introducing a feedback loop that takes both sides in to consideration. Without additionally leveraging the nature of the application, there is not much room for compression of spectrum data that contains long-lived signals and is collected by low-cost sensors with a relatively low sampling rate. Therefore, it makes our adaptive compression generalizable not only to various applications, but also to a diverse group of sensors.

V. Evaluation Results

We implement the compression algorithm using C++ on RBPi-4 with RTL-SDR and the compression ratio adaptation scheme using python on edge device Intel NUC. Due to the space limit, in this section, we omit the results of the compression algorithm with various applications but focus on FlexSpec's compression ratio adaptation scheme in two real scenarios, which are (i) channel detection based on PSD data, and (ii) signal decoding using IQ data.

A. Feedback loop

In this experiment we evaluate the performance of channel detection over FM radio band, which contains 9 active channels with varying energy over time, using compressed PSD data. We compare the performance of FlexSpec with

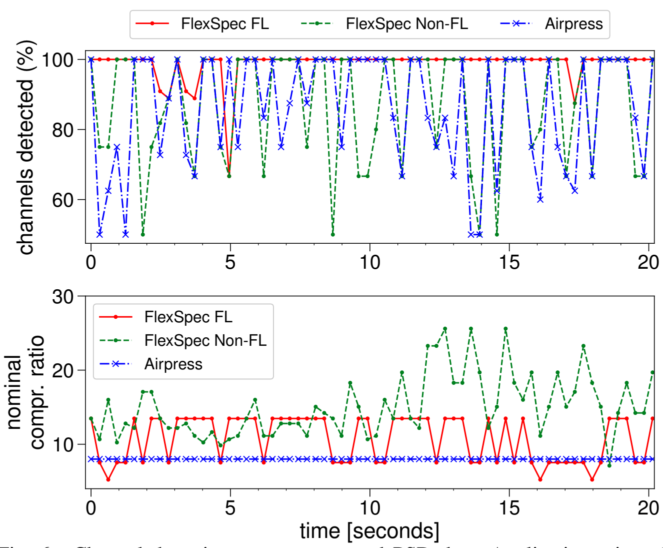


Fig. 6. Channel detection over compressed PSD data. Application-oriented compression ratio adaptation with Feedback Loop performs better than generic approaches.

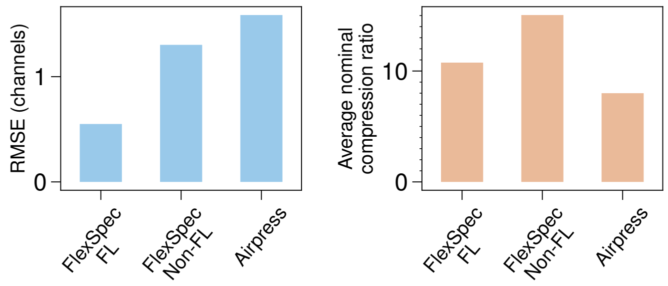


Fig. 7. RMSE of channels detected (left) and average nominal compression ratio for the different solutions evaluated over PSD data.

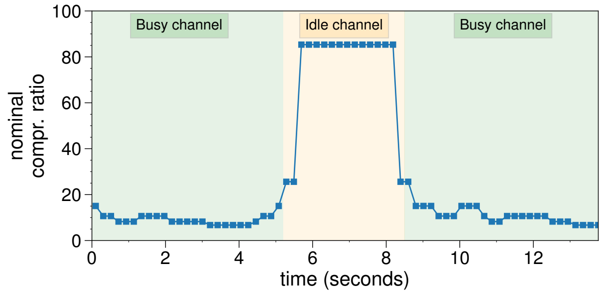


Fig. 8. Compression ratio is set adaptively in a busy-idle channel scenario to reduce data that are backhauled. Compression ratio adapted every 200 ms.

and without *Feedback Loop* (FL) and also compare with Airpress. Fig. 6 (top) shows the channels detected over PSD data for different methods. FlexSpec FL adapts dynamically the compression ratio so that the differences in the number of detected channels smaller than 1 is tolerated. FlexSpec Non-FL determines the new compression ratio by executing a very simple application on the sensor side (to not overload it). It consists of compressing the original signal until the reconstructed one introduces at most 5 dB error. This process is executed on the sensor side with no FL involved. For a fair comparison with Airpress, we select a nominal compression ratio (8) that gives a data compression ratio similar to the average one of FlexSpec FL. Fig. 7 shows that FlexSpecFL performs near perfect in terms of channels detected while it keeps the average compression ratio about 10. FlexSpec

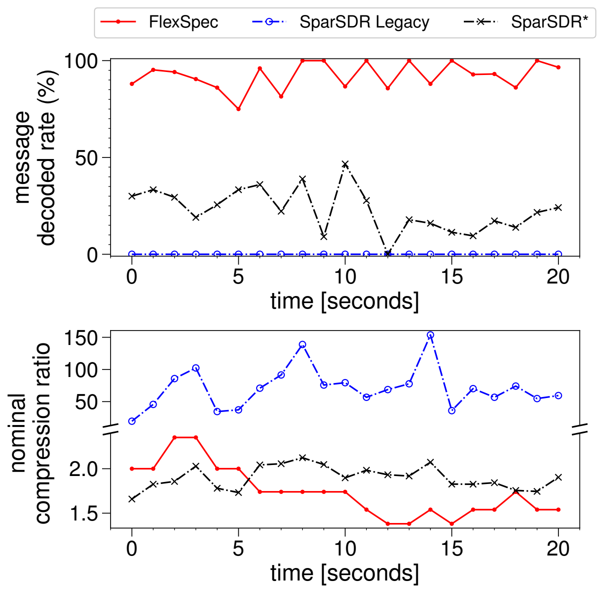


Fig. 9. ADS-B messages decoded and compression ratio used for FlexSpec and SparSDR.

Non-FL reports a higher nominal compression ratio but also introduces an RMSE of the channels detected larger than 1, which indicates that simple adaptation running on the sensor without involving FlexSpec's FL does not optimize applications' performance to get the tolerable performance loss. Lastly, Airpress, which used a fixed compression ratio, reports an even higher RMSE of channels detected, which shows adaptation is needed even in a relatively stable band.

B. Busy-Idle state

In this experiment, we evaluate how the nominal compression ratio is set depending on the state of the spectrum and the application running in the edge device. We collect PSD data on the same FM radio band as the previous setting. In order to simulate the idle state, we disconnect the antenna from the radio receiver. Fig. 8 shows how the nominal compression ratio is set adaptively. We notice that during the idle channel state, FlexSpec increases the compression ratio (by decreasing k_{fine}), since there is no useful signal information to detect channels, reducing the the size of uplink data. We achieve up to 10× more nominal compression ratio during the idle state than Airpress, which uses a fixed compression ratio (8) as Fig. 6 shows. Considering the overhead of data format for FlexSpec and Airpress respectively, this translates to up to 7× more reduction in uplink data size.

C. IQ decoding

We evaluate FlexSpec over IQ data with the goal of maximizing the message decoded rate. For this experiment, we use ADS-B and IoT-433 signals as the example wideband and narrowband signals respectively. IoT-433 signals are signals from IoT devices at ISM 433 MHz band with 0.1 MHz bandwidth.

For the ADS-B signal, the compression is performed in segments of 2 ksamples and the message decoding is performed in

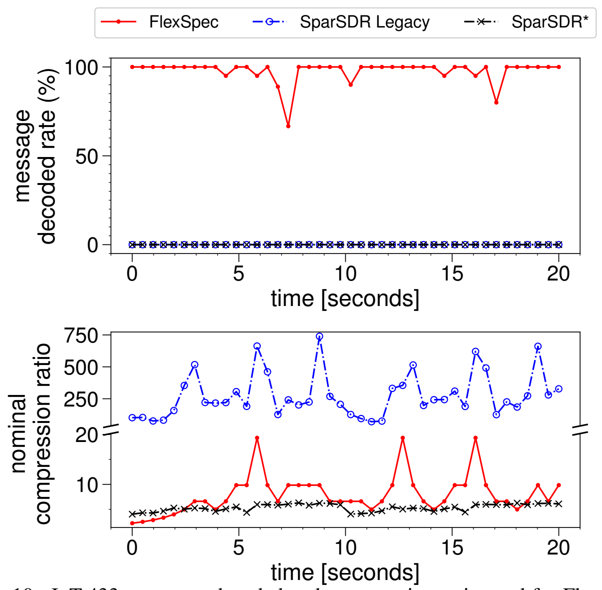


Fig. 10. IoT-433 messages decoded and compression ratio used for FlexSpec and SparSDR.

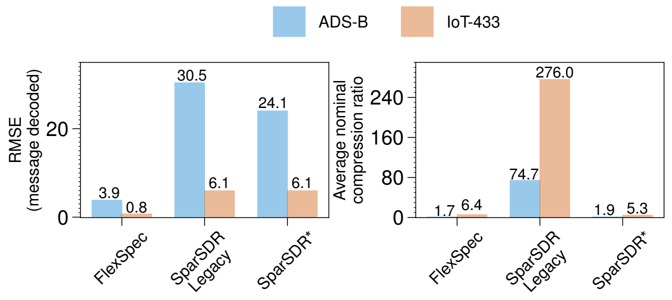


Fig. 11. Summary of IQ decoding performance and compression ratio using FlexSpec and SparSDR.

the adapter every 1 Msamples using *dump1090* [1]. The latter also means that every 1 Msamples FlexSpec takes a decision on the new compression ratio to be used. Fig. 9 compares FlexSpec and SparSDR with regard to their message decoded rate for ADS-B. For SparSDR legacy, we use a power threshold of 5 dB above the noise floor as suggested, but we find no message can be decoded at all. Therefore, we also tune the power threshold such that SparSDR outputs a similar nominal compression ratio as FlexSpec, denoted as SparSDR*. Fig. 11 shows that for ADS-B, FlexSpec has 6× less messages unable to be decoded compared with SparSDR*, at a very similar nominal compression ratio. Considering the data format of FlexSpec and SparSDR and the overlapping time window in SparSDR, FlexSpec's 6× less messages unable to be decoded comes with a 1.7× more reduction in uplink data size.

For the IoT-433 signal, we apply the same configuration as ADS-B but the *rtl_433* decoder [3] is used. Fig. 10 shows how FlexSpec outperforms SparSDR. Similar to ADS-B, if we use a power threshold of 5 dB above the noise floor (SparSDR legacy), no messages can be decoded at all. However, different from ADS-B, if we apply a power threshold that gives a similar nominal compression ratio as FlexSpec (SparSDR*), significant amount of messages still cannot be decoded. Fig. 11 shows that, for IoT-433 signal, FlexSpec

TABLE I
LATENCY OF FLEXSPEC'S FEEDBACK LOOP FOR DIFFERENT STREAMING APPLICATIONS EVALUATED ON BOTH SCENARIOS: RBPI-4 AND RBPI-4 + INTEL NUC.

Pipeline App	RBPi-4 (ms)	RBPi-4 + Intel NUC (ms)
Channel detection (PSD)	144	37
dump1090 (IQ)	264	115
IoT433 (IQ)	1080	220

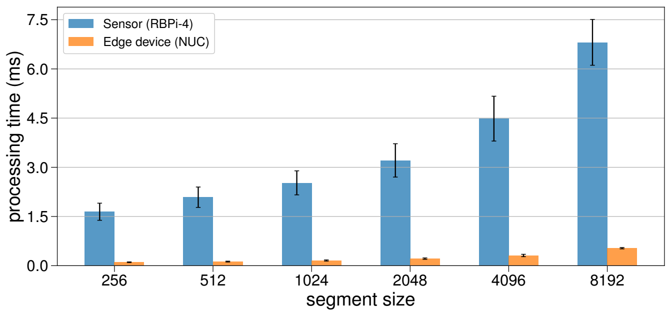


Fig. 12. FlexSpec compress/decompress latency performance in sensor (RBPi-4) and edge device (Intel NUC).

has 8× less messages undecodable compared with SparSDR*, with 1.2× more nominal compression ratio. If the data format of FlexSpec and SparSDR and the overlapping time window in SparSDR are considered, we have 2.2× more reduction of uplink data size.

D. Latency performance

The performance of our solution based on FWHT does not depend on the compression coefficent used but on the segment size of compressed/decompressed data. We evaluate the performance of the compression algorithm in different platforms (RBPi-4 and Intel NUC) for different segments sizes. Fig. 12 shows how the decompression processing time grows exponentially with the segment size. We can also observe that the compression/decompression process is 16× faster on the Intel NUC than on the sensor (RBPi-4) due to the computational limitations of the latter. For the reference segment size (2048) used in all the evaluations, compress/decompress functions take 0.20 ms in the edge device.

We also evaluate the latency of FlexSpec in terms of how long it takes to provide a new compression factor according to the application executed on decompressed data. Table I shows the results of the two different scenarios evaluated. In the first scenario (named as "RBPi-4") compressing/decompressing data and streaming application are executed on the sensor. The second one (named as "RBPi-4 + Intel NUC") performs the compressing data on the sensor, but decompressing data and streaming application are executed on the edge device (Intel NUC). Even assuming a transmission delay of 5 ms between the sensor and the edge device, we obtain a lower latency in the "RBPi-4 + Intel NUC" scenario, meaning that we can provide an optimal compression factor more often (in average 3.5× faster). The final latency of FlexSpec's feedback loop strongly depends on the streaming application executed to obtain the optimal compression factor.

VI. CONCLUSION

A key hurdle to build multiple spectrum applications with ubiquitous spectrum crowdsensing systems is the high uplink bandwidth needed as well as the resulting demands on centralized storage. We presented FlexSpec, a general framework of adaptive uplink data compression to remove this hurdle. FlexSpec solves the main drawbacks of prior work, namely the problem of important information missed in the spectrum compression technique and lack of adaptation with respect to the application needs. We have shown the benefits of our framework with several spectrum applications, from signal detection to IQ decoding. We envision that FlexSpec's techniques will be an integral component of large spectrum crowdsensing systems.

ACKNOWLEDGEMENT

We thank the anonymous reviewers for their valuable feedback. Yijing Zeng and Suman Banerjee are supported in part through the following US National Science Foundation grants: CNS-1719336, CNS-1647152, CNS-1629833, CNS-2003129, and an award from the US Department of Commerce with award number 70NANB21H043. The research work at IMDEA was sponsored in part by the NATO Science for Peace and Security Programme under grant G5461, and in part by the Madrid Regional Government through TAPIR-CM project S2018/TCS-4496.

REFERENCES

[1] dump1090. https://github.com/openskynetwork/dump1090-hptoa.
[2] Raspberry pi 4. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.
[3] rtl-433. https://github.com/merbanan/rtl_433.
[4] Rtl-sdr. https://www.rtl-sdr.com.
[5] HARMUTH, H. F. *Transmission of information by orthogonal functions*. Springer Science & Business Media, 2012.
[6] HOARE, C. A. Algorithm 65: find. *Communications of the ACM 4*, 7 (1961), 321–322.
[7] KHAZRAEE, M., GUDDETI, Y., CROW, S., SNOEREN, A. C., LEVCHENKO, K., BHARADIA, D., AND SCHULMAN, A. Sparsdr: Sparsity-proportional backhaul and compute for sdrs. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services* (2019), ACM, pp. 391–403.
[8] KWON, S.-K., TAMHANKAR, A., AND RAO, K. Overview of h. 264/mpeg-4 part 10. *Journal of Visual Communication and Image Representation 17*, 2 (2006), 186–216.
[9] LI, Y., ZENG, Y., AND BANERJEE, S. Enabling wideband, mobile spectrum sensing through onboard heterogeneous computing. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (2021), pp. 85–91.
[10] O'SHEA, T., AND HOYDIS, J. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking 3*, 4 (2017), 563–575.
[11] PFAMMATTER, D., GIUSTINIANO, D., AND LENDERS, V. A software-defined sensor architecture for large-scale wideband spectrum monitoring. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks* (2015), ACM, pp. 71–82.
[12] RAJENDRAN, S., CALVO-PALOMINO, R., FUCHS, M., VAN DEN BERGH, B., CORDOBÉS, H., GIUSTINIANO, D., POLLIN, S., AND LENDERS, V. Electrosense: Open and big spectrum data. *IEEE Communications Magazine 56*, 1 (2018), 210–217.
[13] RITTER, T. Walsh-hadamard transforms: A literature survey. *Research Comments from Cipers by Ritter, www. ciphersbyritter. com/RES/WALHAD. HTM,(10 pages)* (1996).
[14] SERIES, S. Spectrum occupancy measurements and evaluation - report itu-r sm.2256-1.

[15] Tu, C., Srinivasan, S., Sullivan, G. J., Regunathan, S., and Malvar, H. S. Low-complexity hierarchical lapped transform for lossy-to-lossless image coding in jpeg xr/hd photo. In *Applications of Digital Image Processing XXXI* (2008), vol. 7073, International Society for Optics and Photonics, p. 70730C.

[16] Zeng, Y., Chandrasekaran, V., Banerjee, S., and Giustiniano, D. A framework for analyzing spectrum characteristics in large spatio-temporal scales. In *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking* (2019), ACM.

[17] Zhang, T., Patro, A., Leng, N., and Banerjee, S. A wireless spectrum analyzer in your pocket. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (2015), ACM, pp. 69–74.

[18] Zheleva, M., Larock, T., Schmitt, P., and Bogdanov, P. Airpress: High-accuracy spectrum summarization using compressed scans. In *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)* (2018), IEEE, pp. 1–5.