Light-weight Server-assisted H-K Compression for Image-based Embedded Wireless Sensor Network

Younghoon Song, Hyungsik Shin, and Jeongyeup Paek

Abstract—Embedded wireless imaging systems based on wireless camera sensor network can be used to observe biological phenomena unobtrusively in various environmental monitoring applications. In our prior work, we have shown the feasibility and benefits of such a system through actual deployments. We have also shown that low-complexity data compression schemes can be employed to improve image transfer rate or to lower the energy costs of communication. In this article, we extend upon our prior work and propose a scheme called as H-K compression, a simple light-weight image compression algorithm combining the ideas of Huffman coding and K-means clustering. Specifically, H-K compression applies K-means clustering for pixel color grouping and Huffman coding for the group color encoding, but combines the two schemes into one algorithm which can reduce the computational cost. Using 100,000 images collected from our pilot deployments at James Reserve, we study the applicability and impact of the proposed algorithm. Our results suggest that the cost of running the learning steps on an embedded sensor node may outweigh the benefit of compression, but offloading the learning can provide significant energy gains. Evaluations using the dataset show that our proposed scheme compresses the data by $\sim 57\%$ and reduces power usage by $\sim 43\%$ when sending image updates from a bird nest periodically every 15 minutes.

Index Terms—Wireless sensor network, Image compression, K-means clustering, Huffman code, Embedded systems

I. INTRODUCTION

With improved and miniaturized low-power wireless platforms, embedded wireless imaging systems that are based on wireless camera sensor networks can be used to observe biological phenomena unobtrusively. Among various applications of those systems, there are many environmental monitoring applications such as vineyard [1], agricultural [2], and bird nest monitoring systems [3, 4]. Another example application is to install wireless camera sensors in buildings or heavy machineries that are hard to reach by human beings: for example, it will be easier to detect decrepitude from a distance, which can free the maintenance workload of manual checking.

In our prior work [5], we have shown the feasibility and usefulness of an embedded wireless imaging system through real world deployments and pilot studies. In particular, we used a custom-designed embedded wireless imaging system for monitoring biological events such as the status of bird nests.

During a deployment period of three months, we collected over one hundred thousand images from nineteen camera nodes, which were deployed in a region of 0.05 square miles under highly variable environmental conditions. A few sample images from our deployment are shown in Fig. 1. Many biologists found the on-line, near-real-time system very useful for answering various biological questions by using the obtained image data.

In these wireless imaging systems, it is evident that a well-designed data compression scheme can improve image transfer rate per node and/or can reduce the amount of energy consumed when the images are transferred wirelessly. However, implementing data compression schemes are often constrained by limited memory space and low processing power of embedded platforms. For example, although families of JPEG or DCT based image compression schemes guarantee high compression ratios, these techniques require large amount of resources and computation, which is not suitable for embedded devices [7]. Therefore, image compression algorithms for these systems must be simple enough to be implemented on such resource constrained devices.

In addition to the limitations in computational resources, typical embedded sensor nodes have also power consumption constraints because they often operate on battery power. This energy consumption constraint naturally introduces another challenge to the system design. As a result, we need to design a low-complexity algorithm that takes into account the limited computational and memory resources, the limited radio bandwidth, and the power constraints under which these devices operate.

To address the challenges, our previous work suggested K-means clustering based data compression scheme for image-based wireless sensor networks with low-power embedded devices [8]. Specifically, we used the K-means clustering algorithm for image compression by clustering colors present in an image into similar color groups so that only the $K$ most representative colors (centroids within each group) are used for representing the image. We showed that shrinking
the number of colors to only that of the most common ones reduces the computational complexity significantly on the embedded platforms. With this scheme, we acquired \( \sim 49.9\% \) compression ratio without significant loss of image quality (low distortion), resulting in \( \sim 37.5\% \) reduction in energy usage per image transfer (including the cost of compression, assuming 1-hop transmission).

In this article, we develop and extend upon our prior work to propose H-K compression, an image compression scheme combining the ideas of Huffman coding and K-means clustering, for image-based wireless sensor networks of low-power embedded devices. Specifically, we use K-means clustering to group pixels based on similarity of colors, and encode the centroid colors using Huffman coding to compress the original image. We combine the two schemes to reduce computation times and to improve compression ratios. We inherit the benefits of K-means clustering from our prior work, and exploit the advantages of Huffman coding (i.e. optimal lossless encoding) through the reduced number of colors in a K-clustered color set. This achieves better compression ratio than Huffman or K-means only schemes with identical image quality as the K-means clustering based compression while using less energy.

Using one hundred thousand images collected from our pilot deployments at James Reserve\(^1\), we study the applicability and impact of the proposed H-K compression algorithm. Our results suggest that the energy cost of running the learning step (K-means clustering and Huffman code building) on a wireless sensor node may outweigh the benefit of data compression, but offloading the learning step to the server-side (server-assisted) and performing only the compression step on the sensor node can provide significant energy gains. Evaluation with real-world dataset shows that our proposed scheme achieves compression ratio of \( \sim 57\% \) and reduces power usage by \( \sim 43\% \) with low image distortion, which is a significant improvement from our prior work.

The remainder of this paper is structured as follows. We first discuss the prior related work in Section II. In Section III, we present our application scenario and the design of our proposed scheme on a systems perspective. We then briefly review the use of K-means clustering algorithm for image compression in Section IV. Using Section V we describe the specific details on, and investigate the effectiveness of, applying the Huffman-encoding for image compression. In particular, we compare the Huffman-encoding based i compression with the K-means clustering based one. Section VI is the main part of this manuscript where we propose H-K compression algorithm that combines the two distinct compression algorithms together into one compression scheme. In this section, we also evaluate the compression ratios and energy gains of H-K compression for stand-alone and server-assisted approaches, and compare it against Huffman only and K-means only schemes. Finally, we conclude our work in Section VII.

\(^1\)http://www.acnr.org/reserves/james-san-jacinto-mountains-reserve.html

II. PRIOR AND RELATED WORK

A number of previous work have proposed and examined to utilize compression schemes for wireless sensor network systems in order to maximize system-level efficiency.

The work in [7] explored energy tradeoffs involved in JPEG compression [9, 10] on resource-constrained platforms. They showed that the energy consumption of JPEG image compression is significantly high on embedded platforms: the long latencies of the computation and processing lead to higher energy consumption.

JPEG-LS [11–13] is a lossless/near-lossless compression standard for continuous-tone images, developed with the aim of providing a low-complexity image compression standard that could offer better compression efficiency than lossless JPEG. While JPEG-LS is generally much faster than JPEG2000 and much better than the original lossless JPEG standard thanks to its impressive coding efficiency, its computational overhead to achieve such efficiency is very high. Therefore, processing our images through JPEG-LS was not possible when using typical resource-limited platforms in WSNs [14, 15] due to memory constraints in both of RAM and ROM. Even if the software containing the operations were to fit on our target platform, the computational power limitations can cause long latencies for the image processing.

Besides the JPEG compression approach, there are two recently-proposed image compression schemes for sensor network systems, Tiny block-size image coding (TBSC) [16] and Optimal Zonal 2x2 BinDCT [17]. While both of them are effective compression schemes, they require much more memory than what most of recent WSN platforms (i.e. motes) provide. We agree that newer platforms with more memory hold the resources to support such complex algorithms [18]. These platforms, however, tend to consume considerable energy, so they were not satisfactory for our deployment purposes due to energy usage constraints. We note that an alternative approach such as the one proposed by Kaddachi et al. [19], which is a hardware-based image compression, can also be a solution. However, the hardware-based solution incurs relatively high device manufacturing cost.

Another well-known low-complexity lossy image compression technique for greyscale images is Block truncation coding (BTC) [20]. It divides the original images into blocks and then uses a quantizer to reduce the number of grey levels in each block whilst maintaining the same mean and standard deviation. Using sub-blocks of 4x4 pixels gives a compression ratio of 4 to 1 assuming 8-bit integer values are used during transmission. Larger blocks allow greater compression, but image quality deteriorates as the block size increases due to the nature of the algorithm.

Based on its low-complexity and high compression ratio (75%) characteristics, we thought that BTC might be a good fit for our application and considered it for use in our system. However, we found that BTC is not suitable mainly for the following two reasons. First, even though the complexity of the algorithm is low (lower than Huffman) and runs fast on a regular computer, its running time was in fact quite high on our embedded platform. Specifically, on our TelosB [14]...
platform with MSP430 processor and mspgcc compiler, the running time of BTC compression for a single image was 8.68 seconds. This is because BTC algorithms require floating point and square root operations while our embedded processor does not have a co-processor for efficient execution of those operations. Second, and more importantly, when we compressed the images in our dataset using BTC, the quality of images were lower than our application requirement of MSE being less than 10. The average mean square error (MSE) was 22.97 with standard deviation of 4.28, whereas our domain scientists (i.e. biologists) concluded MSE below 10 as acceptable threshold for our application [5, 8]. Fig. 2 shows four example images (bottom row) compressed and decompressed using BTC algorithm (top row are the originals), and we will later show a summary of all the results in Table VIII.

In order to overcome the shortcomings of the prior work, our compression algorithm focuses on a light-weight software-based solution, which is cost effective and applicable to most resource-limited low-power platforms. More detailed review of the prior and related work can be found in our previous work [8].

III. APPLICATION AND COMPRESSION SCENARIO

Our system is an image-based wireless sensor network, where several camera sensor nodes are deployed in the target environment. Fig. 3 illustrates our basic application system architecture. In our experiments, we deployed nineteen nodes to the bird nest boxes. The deployed sensor nodes communicate with a server over multihop wireless network.

An embedded sensor node consists of a Cyclops camera [21], and a Mica2 or MicaZ [15] mote as the processing core and radio transceiver, both of whom have an 8-bit microcontroller. Each sensor node captures a 200×200 pixel gray-scale image every $T_{\text{image}}$ minutes, where each pixel value is represented as an 8-bit unsigned integer. Therefore, an image can have up to 256 different gray-scale intensities.

A dedicated application server sends tasks (i.e. commands) to all sensor nodes every $T_{\text{image}}$ minutes so that each sensor node can capture images, read sensor values, perform computation, and return the processed data in a near synchronized manner. For our experiments, we have used the Tenet software stack [22] along with the rate-controlled reliable transport (RCRT) protocol [23] which guarantees 100% packet delivery under reasonable link and deployment conditions. However, other multihop networking mechanism (e.g. [24–26]) can be used as well. Also, the image transfer interval $T_{\text{image}}$ can be selected based on the achievable throughput of the underlying network. We point the readers to our previous work for the full details on the actual field deployment of the bird nest monitoring application and its hardware specifications [5].

Given the image-based wireless sensor network architecture described above, we explore two scenarios to compress and transmit images.

Scenario 1 (self-contained) : In the first scenario, each sensor node captures an image every $T_{\text{image}}$ minutes (e.g. 15 minutes), extracts (learns or computes) the information needed for compression on each image, applies it to compress that image, and transmits the compressed image to the server over multihop wireless network together with the information needed to decompress that image. This process is repeated for every captured image.

Scenario 2 (server-assisted) : In the second scenario, the server-assisted strategy is used. Each sensor node still captures an image every $T_{\text{image}}$ minutes, but transfers the raw uncompressed image to the server only at an interval of $T_{\text{learn}}$, where $T_{\text{learn}} \gg T_{\text{image}}$ (e.g. 1 day or 1 month). Except those intervals, each sensor node transfers compressed images. When a raw uncompressed image is received, the server will extract (learn or compute) the information needed for compression, and return this extracted information back to the sensor node. Note that the data volume of this information is far less than a raw image. Then the sensor node will use this information to compress the subsequent images, and transmit the compressed images to the server. This process is depicted in Fig. 4.
A primary advantage of the second scenario with the server-assisted strategy is that a sensor node need not spend energy in extracting the compression information from an image (e.g., K clustering), which can be the dominant factor of energy consumption. In addition, the server already has the information to decompress the image which relieves the sensor nodes from the overhead of transferring this information for every compressed image. Compressing the total amount of bits transmitted over the radio will evidently reduce the energy consumption on the sensor nodes in the long term despite the occasional raw image transfers. We also emphasize that this energy gain will increase as the image data travels over multiple hops since the amount of data to transmit at every relaying hop is significantly less than without compression.

The specifics of what the “information needed for compression/decompression” are, and what it means to “extract (learn or compute)” this information depend on the compression scheme used, which we elaborate in the following sections.

IV. K-MEANS CLUSTERING AND IMAGE COMPRESSION

In this section, we provide a brief overview of our prior work which applies K-means clustering algorithm to image compression [8]. Note that our current work is an extension of the previous work.

The K-means clustering algorithm is a well-known unsupervised machine learning algorithm, and it automatically clusters data points into groups. The algorithm partitions the data into K “clusters,” where points in a cluster are close/similar to each other. The mathematical mean of all the points in a cluster is called a centroid of the cluster.

In our prior work [8], we utilize the K-means clustering algorithm for image compression by reducing the number of colors of an image to only those that are representative of the given image. Specifically, suppose that we use 8 bits to represent a pixel value of an image, so an image can have 256 different values for a pixel. Then, we reduce 256 colors into K colors by clustering all the pixel values of the image into K clusters. After clustering, we replace each pixel value into the corresponding centroid value. These replacements reduce the number of bits required to describe a pixel from 8 to \( \log_2 K \) at the cost of less accurate color representation.

For an entire image, we do need a small additional overhead for storing the K centroid colors, where each centroid color is encoded with 8 bits. Therefore, if \( K = 16 \) and the image size is \( 200 \times 200 \), the total number of bits to represent a K-means clustering applied image is \( 16 \times 8 \) (overhead) + \( 200 \times 200 \times \log_2 16 = 160128 \) bits, whereas a raw image would require \( 8 \times 200 \times 200 = 320000 \) bits. This reduces the required number of bits in almost two-fold (49.96%) compared to the uncompressed case. This compression ratio is the same for all images, and it is a major advantage of utilizing K-means clustering algorithm for image compression; it provides deterministic, constant, and high compression ratio regardless and independent of the data within the image to be compressed.

When applying K-means clustering algorithm, there are three critical parameters to be designed. The first is the \( K \) value, which determines the image quality (measured in terms of MSE, PSNR, or SSIM [27]), compression ratio, and memory requirements. The second is the number of learning iterations within a run of K-means clustering algorithm to update and improve the quality of clustering. Finally, the third is the number of trials of the K-means algorithm to run with random initializations to avoid falling into bad local optima.

In general, more iterations and trials of the K-means algorithm result in better learning of the centroids and lower optimization cost; thus, less distortion (e.g., MSE) can be expected in compressed images. However, the optimization cost tends to converge to a local minimum after a certain point, and increasing further the number of iterations or trials may no longer improve the quality of compressed images.

Note that the latter two among the three parameters have direct impact on the running time of the algorithm, which strongly affects the energy usage of the system, as well as the accuracy. Specifically, while the complexity of K-means clustering algorithm is \( O(N) \), where \( N = \text{width} \times \text{height} \) is the size of the image, the number of clusters \( K \), the number of trials \( N_t \), and the number of iterations \( N_i \) dominate the proportional factor of the computation complexity. Thus it is important to minimize those parameters, while the application requirements are satisfied with acceptable image quality. In our prior work, we have experimented with varying parameters, and based on a dataset collected from our deployment, we selected ‘4’ as the number of learning iterations, ‘4’ as the number of trials, and \( K = 16 \) for our application. These values were selected based on the trade-off between acceptable image quality and energy cost of computation.

TABLE I lists the experimental results of average running times and their corresponding energy usage for the K-means clustering based image compression scheme from our prior work [8], and we provide it here to help readers compare with our current work. We used the aforementioned parameters for K-means clustering algorithm, and processed 200×200 gray scale images on a TelosB mote with an MSP430 microcontroller and CC2420 radio.

From the experimental results, we can see that the K-means
clustering based compression actually increases (contrary to our goal of reducing) the overall energy consumption from 311.8 mJ to 1026.0 mJ due to high computational cost of the learning step when performed on a sensor node. Thus the goal of this work is to design an alternative method for image compression on resource limited (or sensitive) embedded platforms such as our WSN scenarios, which can provide better compression ratio and lower energy usage while maintaining the image quality.

V. HUFFMAN CODE AND IMAGE COMPRESSION

In this section, we first discuss the use of Huffman coding as the image compression scheme in our image-based wireless sensor network. Then, we evaluate its performance under the first self-contained strategy scenario and compare it against the K-means clustering based image compression scheme. We also discuss how Huffman coding can be used for image compression when server-assisted strategy is used.

Huffman coding [28], proposed by David A. Huffman in 1951, is one of the most popular and basic entropy coding techniques widely used for lossless data compression. The algorithm derives a variable-length code table from the frequency (or probability) of each possible value of the source symbol, and represents more common symbols using fewer bits than less common symbols. The code assignment is guaranteed optimal among many methods that encodes symbols separately. Despite the fact that lots of compression techniques have been developed ever since, Huffman coding is still widely used thanks to its low-complexity ($O(N \log N)$), efficiency, optimality, losslessness, and lack of patent coverage [29, 30].

In our deployment case, each symbol is an 8-bit gray-scale color intensity of each pixel, and the frequency of occurrence is calculated over the total of 40000 pixels ($200 \times 200$ image). Then we assign a variable-length-bit code to each color, where more popular colors are represented using fewer bits than less popular colors.

When building Huffman codes, however, there is one design decision to be made, which is whether to assign Huffman codes for zero frequency colors or not. Binary codes for zero frequency colors are not necessary if the Huffman code extracted from an image is used only for decoding that image as in the self-contained scenario. However, if we were to use the same Huffman code for compressing (and thus decompressing) other images as well, as is the case for the server-assisted scenario, all colors should be encoded including the ones with zero occurrence in one particular image because zero frequency colors of an image may appear in other subsequent images.

To investigate the impact of the above decision in terms of overhead and compression ratios, we implemented three different methods for transmitting Huffman code information for decompression when using self-contained strategy.

Method A: We assign code for all possible 256 ($0 \sim 255$) colors. When transmitting the Huffman code table, we employ the well-known tree structure sending method [31] depicted in Fig. 5 a), which shows how to transform a Huffman code tree into a bitstream. We search the tree in preorder and assign ‘0’ when we meet a node that is not a leaf node. We assign ‘1’ when we encounter a leaf node, and append the 8 bit color. With this method, Huffman code table can be encoded compactly always with a constant number of bits, which is $256 \times 8 + 256 \times 2 - 1 = 2559$ bits, regardless of the color distribution.

Method B: We do not assign code for zero frequency colors. The purpose of this method is to minimize the overhead of transmitting the information needed for compression/decompression. It uses the same tree structure sending technique as above. However, the total amount of bits will differ depending on the color distribution of the image that generated the Huffman code since the number of zero frequency colors will differ.

Method C: In this method, we do not assign code for zero frequency colors. Furthermore, instead of using the tree structure sending method as with Methods A and B, we transmit the Huffman code table using simple length-color-code encoding. Since each Huffman code varies in length, additional length information is required. Fig. 5 b) shows how this scheme encodes Huffman code table in the form of a sequence of length-value-code triplets. Methods A and B were designed to examine the difference of overhead between assigning code for zero frequency colors or not. Methods B and C were designed to compare the

<table>
<thead>
<tr>
<th>Operation</th>
<th>Avg. Running Time</th>
<th>Est. Energy Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning step (K-means clustering)</td>
<td>153.99 sec</td>
<td>831.58 mJ</td>
</tr>
<tr>
<td>Compress image (given K colors)</td>
<td>7.13 sec</td>
<td>38.52 mJ</td>
</tr>
<tr>
<td>Transmit compressed image w/ K=16</td>
<td>2.67 sec</td>
<td>155.9 mJ</td>
</tr>
<tr>
<td>Total</td>
<td>163.79 sec</td>
<td>1026.0 mJ</td>
</tr>
<tr>
<td>Transmit raw 200x200 image</td>
<td>5.33 sec</td>
<td>311.8 mJ</td>
</tr>
</tbody>
</table>

**Fig. 5.** How to send Huffman code information in a bitstream; a) using the Huffman code tree structure, and b) using the Huffman code table as length-value-code triplets.
difference between the tree sending technique and the code sending technique for Huffman code in terms of overhead and computation time.

TABLE II shows the resulting statistics of compression ratios for the three different methods, and it shows that there are very small differences between the methods. The tree sending technique without zero frequency colors is minutely better (in terms of overhead) than the code sending technique, and the differences in running times were negligible. Furthermore, it turns out from our measurements that these results are similar for both self-contained and server-assisted strategies. For this reason, we have chosen to use Method B for our evaluation of the Huffman coding based compression.

However, we have not yet discussed the challenge of how to use Huffman compression under the server-assisted strategy without codes for zero frequency colors that may appear in other subsequent images. Obviously, we could apply Method A, but the problem of the Method A is that Huffman-tree is constructed into a significantly biased tree due to zero-frequency colors. As a result, if the subsequent images do not match well with the image that the compression information was extracted from, compression ratios of the subsequent images drop dramatically. In some cases, compressed data size became even larger than the raw image.

To resolve this problem and apply Huffman coding to server-assisted strategy, we designed a new method B* where a special ‘NULL’ color that does not exist in the image (denoted as SP(0) in Fig. 5) is used to represent all zero-frequency colors. This can help to encode any new colors that appear in subsequent images. Specifically, when extracting information from an image on the server, we identify a color that did not appear in the image, and define it as the special ‘NULL’ color SP(0). Then the Huffman coding scheme will naturally assign a code for the special color.

After extraction, the server sends tree information along with special color information SP(0) which takes only 1 more extra byte. A sensor node receives this data and re-constructs the Huffman-tree as normal. When performing image compression of subsequent images on the sensor node, if a new image has a color that is absent from the Huffman-tree, the sensor node will write the special code for SP(0) along with the pixel’s actual 8 bits color. By applying this method, the longest code for zero-frequency color will have its length to be the sum of the length of the Huffman code for SP(0) and 8 in bits.

**Huffman coding vs. K-means clustering:** Compressing images using Huffman coding has several different aspects from compressing with the K-means clustering. Those aspects are the compression ratios, the computational complexity, and the degree of image distortion.

First of all, K-means clustering based compression always guarantees ~50% compression ratio (with \( K = 16 \)) because each pixel color is represented using 4 bits instead of 8 bits. However, Huffman code based compression does not guarantee constant compression ratio because it is strictly based on frequency of actual color appearances and thus the compression ratio is highly dependent on the images. For example, if all 256 colors appear with the same frequency in an image, then the Huffman code based compression will not compress the image at all. On the other hand, an image with only few colors will have high compression ratio.

This fluctuation in compression ratios can be noticed in TABLE II as well as in Fig. 6 which shows the cumulative distribution function (CDF) of the compression ratios for Huffman coding. Both of the table and the figure show that the differences between the maximum and the minimum compression ratio range up to 25% while the average is around 36%. Thus, Huffman coding based image compression is highly image dependent, and the compression ratio is significantly lower than the K-means clustering based image compression.

Secondly, the computational complexity and thus the running times for information extraction and image compression are significantly different. TABLE III presents the detailed average running times and estimated energy consumption of the entire Huffman coding based image compression procedure. It shows that wireless communication consumes most of the energy, while the whole Huffman coding procedure is negligible compared to it. Total energy consumption of compressing an image using Huffman coding and transmitting the compressed image is \( \sim 203.24 \text{ mJ} \), which is lower by \( \sim 108.26 \text{ mJ} \) than transmitting a raw image without compression. This implies that even the self-contained approach of compressing each image (with Huffman coding) before sending the image is more effective than sending the raw image. Now, if we compare this with the energy usage of K-means clustering based compression in TABLE I (1026 mJ), we see significant energy gains. This provides us a good reason to consider Huffman-coding despite its disadvantage of being image-dependent and lower compression ratios.

Finally, one simple but indisputable advantage of Huffman-coding over K-means clustering is that it is “lossless” compression; unlike K-means clustering algorithm, Huffman coding scheme does not distort the image, which maintains the full image quality. Considering the above discussed aspects, our goal is to develop an image compression scheme for

<table>
<thead>
<tr>
<th>Operation</th>
<th>Average Running Time</th>
<th>Estimated Energy Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct Huffman-tree</td>
<td>0.358 sec</td>
<td>1.93 mJ</td>
</tr>
<tr>
<td>Compress image</td>
<td>0.335 sec</td>
<td>1.81 mJ</td>
</tr>
<tr>
<td>Transmit compressed image</td>
<td>3.41 sec</td>
<td>199.5 mJ</td>
</tr>
<tr>
<td>Total</td>
<td>4.10 sec</td>
<td>203.24 mJ</td>
</tr>
<tr>
<td>Transmit raw 200x200 image</td>
<td>5.33 sec</td>
<td>311.8 mJ</td>
</tr>
</tbody>
</table>
In this section, we propose **H-K compression**, a simple light-weight image compression scheme combining the ideas of Huffman coding and K-means clustering. Specifically, we use K-means clustering to group pixels based on similarity of colors and encode those colors using Huffman coding to compress the original image, but we combine the two schemes into one to reduce the computational cost. **H-K compression** improves the compression ratio over both K-means clustering and Huffman coding based image compression while maintaining the same image quality level as the K-means based compression with negligible additional energy cost.

A more detailed description of the **H-K compression** algorithm is given as follows. The first step of **H-K compression** is to perform K-means clustering to find the K centroids, the K representative colors in our case. Then, the second step is to replace the colors of all pixels with those K representative colors (centroids). This step effectively compresses the image into K colors; however, we have not reduced the number of bits yet. We refer this step as the "**1st compression w/ K centroids**" step (see TABLE IV). With these color modified pixels, the third step is to count the frequency of the appearances of those K colors and build the Huffman code using K colors. The final and fourth step is to compress the image using the Huffman code built from the previous step. This is the step that actually reduces the total number of bits of the image below the original image size, and since there are only K colors left in the image, the average Huffman code length must be less than or equal to \( \log_2 K \). We refer this step as the "**2nd compression w/ Huffman**" step.

Note that most of the third step could be done simultaneously during the second step while finding the closest centroid of each pixel. This is important because all three steps require repeated iterations over all the pixels in the whole image (200×200 = 40000 pixels), so reducing the number of iterations reduces the running time of the algorithm significantly. Therefore, it saves more time and thus more energy than applying K-means and Huffman coding schemes separately.

**VI. PROPOSAL: H-K compression Algorithm**

**TABLE IV** presents the resulting average running times and estimated energy consumption when applying the compressing 

<table>
<thead>
<tr>
<th>Operation</th>
<th>Avg. Running Time</th>
<th>Est. Energy Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute K centroids (K = 16)</td>
<td>153.99 sec</td>
<td>831.58 mJ</td>
</tr>
<tr>
<td>1st compression w/ K centroids</td>
<td>7.13 sec</td>
<td>38.52 mJ</td>
</tr>
<tr>
<td>Construct Huffman-tree w/ K centroids</td>
<td>0.2610 sec</td>
<td>1.13 mJ</td>
</tr>
<tr>
<td>2nd compression w/ Huffman</td>
<td>0.295 sec</td>
<td>1.59 mJ</td>
</tr>
<tr>
<td>Transmit compressed image</td>
<td>2.27 sec</td>
<td>132.6 mJ</td>
</tr>
<tr>
<td>Total</td>
<td>163.92 sec</td>
<td>1006.92 mJ</td>
</tr>
<tr>
<td>Transmit raw 200×200 image</td>
<td>5.33 sec</td>
<td>311.8 mJ</td>
</tr>
</tbody>
</table>

**TABLE IV** Running time and estimated energy usage of **H-K compression** algorithm based image compression on TelosB platform for 200×200 gray-scale images when images are compressed individually (self-contained scenario).

**Fig. 6. CDF of image compression ratios for Huffman-coding, K-means clustering, and H-K compression algorithm based image compression when images are compressed individually (self-contained scenario).**

**A. Evaluation: Self-contained Scenario**

**TABLE IV** presents the resulting average running times and estimated energy consumption when applying the entire **H-K compression** algorithm based image compression procedure on TelosB platform for 200×200 gray-scale images. This is the case when images are compressed individually (self-contained scenario). It shows that the running times and the energy usage of **H-K compression** is clearly less than the sum of K-means and Huffman coding schemes applied separately.

Also note that, since we find the representative K colors in the 1st K-means compression step, the problem of zero-frequency colors has disappeared in the 2nd Huffman compression step. In addition, since there are only K colors left to be compressed using Huffman coding, it also reduces the size of Huffman tree/table, and thus the energy consumed for constructing Huffman-tree is much less than Huffman-coding alone without K-means compression. This can be observed by comparing **TABLE IV** (for **H-K compression** and **TABLE III** (for Huffman coding) where ‘Construct Huffman-tree w/ K centroids’ and ‘2nd compression w/ Huffman’ operations in **TABLE IV** is faster than ‘Construct Huffman-tree’ and ‘Compress image’ operations in **TABLE III**, respectively. Furthermore, since the compression ratio of **H-K compression** is better than the other two algorithms (explained below), the total number of bytes to be transmitted is lower, and thus the time and energy it takes to ‘Transmit compressed image’ has improved as well.

**TABLE V** summarizes the statistics of compression ratios for K-means clustering, Huffman coding, and **H-K compression** algorithm based image compression schemes when images are compressed individually (self-contained scenario).
The following observations can be made from this table. First of all, the average compression ratio of \( H-K \) compression is significantly better than K-means or Huffman code based compression. Since \( H-K \) compression performs K-means clustering first, the compression ratio is guaranteed to be larger than or equal to 49.96%. This is shown in TABLE V where its minimum compression ratio is above 49.96%.

Another observation is that \( H-K \) compression shows less deviation of compression ratios for different images than the Huffman-coding-only compression. This is due to the fact that the number of colors (and thus the variances in their relative frequency) has reduced. TABLE V shows that the standard deviation of compression ratios of \( H-K \) compression is almost half of that of Huffman-coding, and the difference between maximum and minimum compression ratio is only~10%, which is less than half of that in Huffman-only scheme. This can also be seen in Fig. 6, which plots the CDF of image compression ratios for Huffman-coding, K-means clustering, and \( H-K \) compression algorithm based image compression. \( H-K \) compression's compression ratios are more concentrated around its mean than Huffman-coding based, while K-means has a fixed constant guaranteed compression ratio.

Finally, the image quality of \( H-K \) compression in terms of mean square error (MSE) is obviously identical to the K-means clustering based compression. This is because Huffman coding is ‘lossless’ and thus has zero MSE by definition. In other words, since \( H-K \) compression combines the K-means clustering algorithm with Huffman coding and since Huffman coding is ‘lossless’ by nature, the image quality and thus the MSE is identical to the K-means clustering based compression. Detailed analysis of MSE (statistics, variation depending on parameters, application requirement, etc.) for the K-means clustering based compression can be found in [8], and here we only provide a summary comparison of the overall average in TABLE V (and also later in TABLE VIII).

However, despite all the advantages of \( H-K \) compression, we still have the problem that \( H-K \) compression actually spends more energy in total, which is contrary to our goal of saving energy, compared to sending raw images. This is due to high computational cost consumed when a sensor node performs \( H-K \) compression in the self-contained strategy. Specifically, compressing an image with \( H-K \) compression and transmitting the compressed image consumes total of 1006.92 mJ, but transmitting a raw image requires only 311.8 mJ, resulting in increase of energy usage by an additional 698.12 mJ. Even though there are extra energy gains from transmitting compressed data over multiple wireless hops [32], the computational cost is too high to justify the use of the \( H-K \) compression algorithm without modification. This result is in accordance with [7], which found that JPEG energy consumption is actually higher on low power platforms due to the longer times needed for these platforms to perform the computation tasks to the desired precision. This motivated us to propose a server-assisted compression scheme as we discuss in the next section.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Avg. Running Time</th>
<th>Est. Energy Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct Huffman tree from bitstream</td>
<td>0.011 sec</td>
<td>0.06 mJ</td>
</tr>
<tr>
<td>+ Code Assignments + extract centroids</td>
<td>7.705 sec</td>
<td>41.63 mJ</td>
</tr>
<tr>
<td>Compress image</td>
<td>0.335 sec</td>
<td>1.81 mJ</td>
</tr>
<tr>
<td>Transmit compressed image</td>
<td>2.267 sec</td>
<td>132.6 mJ</td>
</tr>
<tr>
<td>Total</td>
<td>10.318 sec</td>
<td>176.1 mJ</td>
</tr>
<tr>
<td>Transmit raw 200×200 image</td>
<td>5.33 sec</td>
<td>311.8 mJ</td>
</tr>
</tbody>
</table>

**TABLE VI**

**RUNNING TIME AND ESTIMATED ENERGY USAGE OF SERVER-ASSISTED \( H-K \) COMPRESSION ALGORITHM BASED IMAGE COMPRESSION ON TELOSB PLATFORM FOR 200×200 GRAY-SCALE IMAGES.

**B. Server-assisted \( H-K \) compression**

As we mentioned in the previous subsection, when \( H-K \) compression compresses every image individually, extracting compression information from an image takes considerable amount of time and consumes more energy than just sending a raw image. To overcome this problem, we propose server-assisted strategy together with \( H-K \) compression.

Recall that in the server-assisted scenario, each sensor node captures an image every \( T_{image} \) minutes, but transfers the raw uncompressed image to the server only at an interval of \( T_{learn} \) where \( T_{learn} \gg T_{image} \) (e.g. 1 day or 1 month). We call this \( T_{learn} \), the “learning rate”. The server will extract the information needed for compression using the raw image, and return this extracted information back to the sensor node. Then the sensor node will use this information to compress the subsequent images and transmit the compressed images to the server. The primary advantage of this scenario is that a sensor node need not spend energy for extracting the compression information from an image, which can be the dominant factor of energy consumption. In addition, the server already has the information to decompress the image, which relieves the sensor nodes from transferring the overhead information for every compressed image.

**TABLE VI** presents the detailed average running times and estimated energy consumption of the \( H-K \) compression algorithm based image compression procedure on a TelosB sensor node when server-assisted strategy is used. As the table shows, it takes very little time (on the sensor node) to reconstruct the Huffman tree and extract K centroids from the bitstream containing the information sent by the server. Furthermore, the actual “Compress Image” step (similar to the \( 2^{nd} \) compression w/ Huffman” step in TABLE IV for self-contained scenario) is very fast. The dominant time-consumer is the “Find nearest centroids” step, which corresponds to the “1st compression w/ K centroids” step in TABLE IV for self-contained scenario. However, although the total running time spent on an image (10.318 sec) by server-assisted \( H-K \) compression is greater than transmitting a raw image (5.33 sec), the total energy usage has reduced from 311.8 mJ to 176.1 mJ, a 135.7 mJ reduction for every image that is transmitted after compression. This is due to the fact that radio transmissions have higher power consumption than CPU computations; Since the compression ratio has improved, the total number of bytes to be transmitted over the radio has reduced, and thus the time and energy it takes to ‘Transmit
TABLE VII

<table>
<thead>
<tr>
<th></th>
<th>raw</th>
<th>hourly</th>
<th>daily</th>
<th>weekly</th>
<th>monthly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average compression ratio(%)</td>
<td>56.23</td>
<td>57.93</td>
<td>57.04</td>
<td>55.92</td>
<td></td>
</tr>
<tr>
<td>Standard deviation</td>
<td>1.65</td>
<td>1.65</td>
<td>3.32</td>
<td>4.09</td>
<td></td>
</tr>
<tr>
<td>Max compression ratio(%)</td>
<td>65.52</td>
<td>61.93</td>
<td>62.09</td>
<td>60.98</td>
<td></td>
</tr>
<tr>
<td>Min compression ratio(%)</td>
<td>47.56</td>
<td>50.01</td>
<td>24.23</td>
<td>23.04</td>
<td></td>
</tr>
<tr>
<td>Est. Energy Usage per image (mJ)</td>
<td>210.0</td>
<td>177.5</td>
<td>176.3</td>
<td>176.1</td>
<td></td>
</tr>
<tr>
<td>Energy Reduction (%)</td>
<td>32.6</td>
<td>43.1</td>
<td>43.4</td>
<td>43.5</td>
<td></td>
</tr>
</tbody>
</table>

TABLE VIII

**COMPRESSION RATIO STATISTICS, ESTIMATED ENERGY USAGE PER IMAGE TRANSFER, AND ENERGY REDUCTION RELATIVE TO NON-COMPRESSED RAW IMAGE TRANSFER FOR SERVER-ASSISTED H-K COMPRESSION AT VARIOUS LEARNING INTERVALS (HOURLY, DAILY, WEEKLY, MONTHLY) WHEN AN IMAGE IS TAKEN EVERY 15 MINUTES (T_{image} = 15 MIN).**

Fig. 7. CDF of compression ratios for server-assisted H-K COMPRESSION algorithm at various learning rate (hourly, daily, weekly, monthly)

Compressed image quality has improved. Thus, the result proves that using the server-assisted strategy with H-K compression is beneficial in terms of energy gains as long as T_{learn} > T_{image}.

A key challenge in using H-K compression with the server-assisted strategy is to decide how frequently the server assists in the process. In other words, deciding the (learning) time interval T_{learn} of extracting the (de)compression information from an image is a major design problem. If we set the time interval T_{learn} too short, embedded devices have to send raw images often, which results in low energy gain. On the other hand, if the interval between extracting compression data is too long, the image quality and/or compression ratio might drop because the images may differ significantly over time due to changes of the deployment/environment or the monitored objects such as birds growing up in a bird nest.

To investigate this challenge and quantify the impact of learning update rates on the compression ratios, we have taken an experimental approach with our dataset. Fig. 7 plots the CDF of the compression ratios for different learning interval T_{learn} when H-K compression learning operates (1) every hour, (2) once per-day using the first image of each day, (3) once per-week using the first image of the week, and (4) once per-month using the first image of the month.

TABLE VII presents the statistics of the same experimental result, along with the estimates of the average energy usage per image transfer for various learning rates when assuming image transfer rate of every 15 minutes (i.e. T_{image} = 15min). The estimated energy usage includes the energy used for transferring raw images every T_{learn}. The results show that, although the average and maximum compression ratios do not differ significantly with T_{learn}, the minimum compression ratio may do if T_{learn} is longer than one day. This shows that the energy gain from compression is significant up until T_{learn} is one day (43.1%), but increasing T_{learn} further has diminishing gains on-wards mainly due to the fact that the fraction of energy used for raw image transfer every T_{learn} becomes negligible compared to the compressed image transfer every T_{image}.

Based on these results, we were able to conclude that a learning interval T_{learn} of one day is reasonable for our target dataset. It has not only the best average compression ratio of 57.9% but also the best minimum compression ratio. More importantly, it is at the knee of the energy gain curve, achieving 43.1% energy reduction. Note that the impact of T_{learn} on image quality was discussed in [8], and H-K compression has identical image quality characteristics as K-means algorithm since Huffman coding is lossless. Thus, by choosing T_{learn} to be one day, our image-based system can provide domain scientists with images of the target environment with acceptable quality, while maintaining a low power usage profile to lengthen the system lifetime using image compression. While we are careful in generalizing the results to other various applications, we believe that our methodology will suit similar image-based environmental monitoring application scenarios very well.

As a side note, we have also tried using the server-assisted strategy with the Huffman coding (only) based compression. However, the improvement in energy gain was negligible compared to the self-contained Huffman compression results. This is because, unlike self-contained Huffman-coding compression, server-assisted Huffman-coding has to check for special color and write non-existing color.

TABLE VIII summarizes and compares all results together. In this table, compression ratios and MSE results are averages over our dataset, and energy results are estimates solely based on the average compression ratios and running times excluding the energy used for occasional raw image transfer. Run-length encoding (RLE), PackBits, and K-means results are from our prior work [8], and other results are from this work. K-means* and H-K compression* (with mark *) are results using server-assisted scenario. The table shows that server-assisted H-K compression achieves high compression ratio (57.93%) and low energy usage per image (174.67 mJ) while having acceptable image quality (MSE below 10). Although BTC has the best compression ratio among the compared, the image quality (i.e. MSE > 20) is not satisfactory. Huffman is lossless (zero MSE), but has higher energy usage and large variation of compression ratio. To summarize, the advantages of server-assisted H-K compression algorithm are:

- Improved average compression ratio
- Lower variance in image dependent compression ratios
- Low complexity
- Identical image quality as K-means based compression
- Lower energy used per image compression and transfer

where the first four features come from combining the K-means clustering and Huffman coding algorithms into one, and the final advantage comes from using the server-assisted strategy.
<table>
<thead>
<tr>
<th>Raw</th>
<th>RLE</th>
<th>PackBits</th>
<th>BTC</th>
<th>K-means</th>
<th>K-means*</th>
<th>Huffman</th>
<th>H-K compression</th>
<th>H-K compression*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20.3</td>
<td>33.8</td>
<td>75</td>
<td>49.96</td>
<td>49.96</td>
<td>36.61</td>
<td>57.47</td>
<td>57.93</td>
</tr>
<tr>
<td>0</td>
<td>4.4</td>
<td>4.4</td>
<td>22.97</td>
<td>4.41</td>
<td>4.41</td>
<td>2.34</td>
<td>4.41</td>
<td>4.41</td>
</tr>
<tr>
<td>311.8</td>
<td>248.50</td>
<td>77.95</td>
<td>872.82</td>
<td>38.51</td>
<td>3.74</td>
<td>872.82</td>
<td>43.5</td>
<td></td>
</tr>
<tr>
<td>311.8</td>
<td>248.50</td>
<td>77.95</td>
<td>872.82</td>
<td>38.51</td>
<td>3.74</td>
<td>872.82</td>
<td>43.5</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE VIII**

**SUMMARY COMPARISON OF ALL RESULTS TOGETHER. ALL VALUES ARE AVERAGES OVER OUR DATASET. RUN-LENGTH CODING (RLE), PACKBITS, AND K-MEANS RESULTS ARE FROM OUR PRIOR WORK [8], AND ALL OTHER RESULTS ARE FROM THIS WORK. K-MEANS* AND H-K COMPRESSION* (WITH MARK *) ARE RESULTS USING SERVER-ASSISTED SCENARIO.

VII. CONCLUSION

This paper presents a simple light-weight image compression scheme called H-K compression for low-power resource-constrained embedded devices in image-based wireless sensor networks. H-K compression combines the ideas of Huffman coding and K-means clustering into one algorithm to compress the image data collected on these platforms while reducing computational cost. Using 100,000 images collected from our pilot deployments at James Reserve, we study the applicability and impact of the proposed H-K compression algorithm. Our results suggest that the cost of running the learning steps on an embedded sensor node may outweigh the benefit of compression, but a server-assisted strategy can provide significant energy gains without loss of compression ratio or image quality.

Evaluations using the actual deployment dataset show that our proposed scheme with learning interval of one day compresses the data by ∼57% and reduces power usage by ∼43% when transmitting images from a sensor node periodically every 15 minutes. Our results suggest that with these efforts, the H-K compression for image data can be highly effective for low-power embedded platforms, given that the computational overhead of our proposed scheme is minimal, while an efficient compression performance with minimal quality loss can be easily achieved.

ACKNOWLEDGMENTS

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03031348).

REFERENCES


Younghoon Song is currently an undergraduate student at the School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea. He is also a research assistant at the Networked Systems Laboratory (NSL) led by Dr. Jeongyeup Paek.

Jeongyeup Paek received his B.S. degree from Seoul National University in 2003 and his M.S. degree from University of Southern California in 2005, both in Electrical Engineering. He then received his Ph.D. degree in Computer Science from the University of Southern California (USC) in 2010 where he was a member of the Networked Systems Laboratory (NSL) led by Dr. Ramesh Govindan. He worked at Deutsche Telekom Inc. R&D Labs USA as a research intern in 2010, and then joined Cisco Systems Inc. in 2011 where he was a Technical Leader in the Internet of Things Group (IoTG), Connected Energy Networks Business Unit (CENBU, formerly the Smart Grid BU). At Cisco, he was one of the lead engineers for CG-Mesh system and WPAN software. In 2014, he was with the Hongik University, Department of Computer Information Communication as an assistant professor. Jeongyeup Paek is currently an assistant professor at Chung-Ang University, School of Computer Science and Engineering, Seoul, Republic of Korea since 2015. He is an IEEE and ACM member.

Hyungsik Shin received his B.S. degree in Electrical Engineering from Seoul National University in 2003. He received his M.S. degree in 2005 and Ph.D. degree in 2011 from Stanford University in Electrical Engineering, where he was a member of the Information Systems Laboratory (ISL). He worked at Docomo Communications Laboratories USA, Inc. as a research intern in 2010, and then joined Docomo Innovations, Inc. in 2011 where he was a Research Engineer in the Open Service Innovation Group. At Docomo Innovations, he was one of the lead engineers of machine learning and natural language processing projects. Hyungsik Shin is currently an assistant professor at the School of Electronic and Electrical Engineering, Hongik University, Seoul, Republic of Korea.