Trimming the Tree: Tailoring Adaptive Huffman Coding to Wireless Sensor Networks

Andreas Reinhardt[1], Delphine Christin[2], Matthias Hollick[2], Johannes Schmitt[1], Parag S. Mogre[1], and Ralf Steinmetz[1]

[1] Multimedia Communications Lab, Technische Universität Darmstadt
Rundeturmstr. 10, 64283 Darmstadt, Germany
{areinhardt, jschmitt, pmogre, ralf.steinmetz}@kom.tu-darmstadt.de
[2] Secure Mobile Networking Lab, Technische Universität Darmstadt
Mornewegstr. 32, 64293 Darmstadt, Germany
{delphine.christin, matthias.hollick}@seemoo.tu-darmstadt.de

Abstract

Nodes in wireless sensor networks are generally designed to operate on a limited energy budget, and must consciously use the available charge to allow for long lifetimes. As the radio transceiver is the predominant power consumer on current node platforms, the minimization of its activity periods and efficient use of the radio channel are major targets for optimization. Data compression is a viable option to increase the packet information density, resulting in reduced transmission durations and thus allowing for an optimized channel utilization. The computational and memory demands of many current compression algorithms however hamper their applicability on sensor nodes.

In this paper, we present a novel variant of the adaptive Huffman coding algorithm, operating on reduced code table sizes and thus significantly alleviating the resource demands for storing and updating the code table during runtime. An implementation for tmote sky hardware proves its adequacy to the capabilities of sensor nodes, and we present its achievable compression gains and energy requirements in both simulation and real world experiments. Results anticipate that overall energy savings can be achieved when transferring packets of reduced sizes, even when increased CPU utilization is incurred.

1 Introduction

In general, energy budgets of nodes in wireless sensor networks (WSNs) are tightly limited [1], thus necessitating the design of applications with increased awareness to their energy consumption. As radio transmissions are an inherent and crucial characteristic of WSNs, but current radio transceivers, such as the widely employed CC2420 device, still expose power consumptions of tens of milliamperes [2], permanent operation of the radio transceiver leads to quick depletion of the battery in both transmission and reception mode. This problem can be approached in several ways, reaching from energy-aware medium access

control (MAC) protocols to highly application-specific means of data compression. In this paper, we focus on compressing packet payloads, targeting to reduce the transmission duration and thus the energy required to exchange data. We investigate the achievable energy savings while disregarding the influence of MAC protocols in our analysis, as reduced packet transmission durations always correspond to savings in transmission energy. The share of the overall radio energy consumption however depends on the selected MAC protocol and its features like duty cycling and low-power listening [3]. The presented solution is designed to remain compatible with both existing header compression schemes as well as energy-aware MAC protocols. In fact, our approach is even capable of compressing both packet payloads and headers.

While data processing and compression mechanisms specifically tailored to an application may provide optimal compression results, they require individual adaptation to sensor data and packet structures and thus place an additional load on the application developer. In contrast, generic data compression solutions, as known from desktop computers, often greatly exceed the capabilities and available resources of embedded sensing systems. In this paper, we pursue the strategy to adapt a generic compression algorithm to the capabilities of sensor nodes. The resulting generic and application-agnostic solution allows to compress data without necessitating additional programming efforts. Opposed to existing approaches, which buffer multiple packets of data prior to compression, our approach targets applications that rely on immediate transmissions; i.e. each packet is compressed individually prior to its transmission.

We focus on the adaptation of a lossless adaptive data compression algorithm, based on adaptive Huffman coding (AHC), where literals in the input sequence are replaced by binary codes with a length reciprocal to the frequency of their occurrence [4]. Our analysis of the existing adaptive Huffman coder implementation for WSNs by Guitton et al. in [5] however revealed that on a TelosB platform, more than 62% of both program Flash and RAM are consumed to maintain a single compressed unicast radio connection. Instantiating more than one connection has not been possible at all due to the memory requirement for storing the corresponding Huffman code table. We address this limitation by making use of Huffman code trees with a limited number of entries, greatly reducing computational and memory consumption at the possible cost of slightly degraded compression ratios. By comparing the achievable compression gains and energy requirements, we prove the applicability and benefits of the proposed approach considering the data-oriented characteristics of traffic in many deployments.

The contributions of this paper are as follows:

1. We analyze the characteristics of WSN traffic from different deployments and prove that compression gains can be achieved when only a subset of the contained symbols are encoded.
2. We present a modification to the adaptive Huffman coding algorithm, which operates on code trees with a limited number of elements.

3. We prove its adequacy to sensor networks through an evaluation of its compression gain and energy demand as well as its applicability on real hardware.

In a first step, we present existing approaches towards data compression in WSNs in Sec. 2. We describe selected data traces taken from real sensor network deployments and estimate their compression gain when encoding only a subset of symbols in Sec. 3. In Sec. 4, we present our modifications to the AHC algorithm. Simulation results for both compression gain and energy consumption are presented in Sec. 5, followed by the results from a real-world experiment. We conclude this paper in Sec. 6 and provide an outlook on prospective future work.

2 Related Work

Pottie and Kaiser have determined in [1] that the energy demand to transfer one kilobyte of data over a distance of one hundred meters in a WSN is the same as required for executing three million CPU instructions. Later, this observation was confirmed by Sadler and Martonosi, who determined that the one-hop transmission of a single byte consumes energy equivalent to performing several thousand instructions on an MSP430 microcontroller [6]. In the same work, the authors propose the RT-LZW (retransmission LZW) algorithm, which achieves compression gains up to a factor of 2.5x when operating on aggregated data blocks of 528 bytes each. It relies on retransmissions of lost packets to ensure that data required to construct the code dictionary is present at both parties, possibly resulting in energy expenses for these additional transmissions.

Guitton et al. have analyzed the applicability of adaptive data compression in WSNs in [5]. They have extended the AHC algorithm by fault-tolerant mechanisms, which groupwise acknowledge transfers of encoded data and adapt the dictionaries to the successfully received data only. They do however not measure achievable compression gains or the energy consumption of their algorithm. When packet structures can be statically defined prior to node deployment and some fields are known to remain constant or only change incrementally, the EasiPC packet compression scheme by Ju and Cui [7] can also be used to transmit changed fields only.

In [8], Tsiftes et al. have focussed on compressing firmware updates that are transferred over the radio, and designed the SBZIP algorithm, a derivative of BZIP2, adapted to the requirements present in sensor networks. However, the implementation of SBZIP on sensor nodes does not target to compress application-generated data, but is instead used to decompress application code updates. Chou et al. present means to reduce an overall network's energy consumption by exploiting the Slepian-Wolf coding theorem in a low-complexity implementation in [9]. Hereby, no inter-node communication overhead is required as long as the correlation between the data is known. Targeting to reduce the overall number of packet transmissions, the approach is orthogonal to our concept of reducing the sizes of packets and can be used supplementary.

In [10], we have presented the Squeeze.KOM compression layer as an architectural element for sensor network nodes. Using a differential coding module,

compression gains of up to 35% can be achieved at low computational cost and overall energy savings. Additionally, we have presented a feasibility study of data compression on WSN nodes in [11]. Focused on the energy gains of application-specific compression means for a wearable sensor, we have determined overall platform energy savings of up to 5% in a realistic application setting.

We are however not aware of any previous work that discusses the energy efficiency of adaptive compression algorithms in detail while providing an extensive analysis of their applicability on current WSN hardware.

3 Analyzing the Traffic in Existing Sensor Networks

In the last decade, a variety of WSNs have been deployed in a wide range of scenarios, including wildlife surveillance [12, 13], object tracking [14], or environmental monitoring [15]. In most of the WSN deployments, network traffic follows a convergecast scheme; all data is routed out of the network using a collection tree or equivalent means, rooted at one or more sinks [16]. Especially when the packet payload is comprised of environmental data, transfers often take place at a regular interval. Timely message delivery is not essential in such scenarios, but the loss of a series of packets is often interpreted as a node failure, hence regular successful transmissions are essential to determine the state of the network.

For our analysis, we have considered four exemplary data sets from existing WSN deployments: PermaSense [15], Glacsweb [17], and two series taken from the Porcupines [18]. For PermaSense, we have used 19,730 packets of 30 byte payload each transmitted by node 2036 from 15 November to 15 December 2008, taken from the project website[3]. From the Glacsweb deployment, we have used all 523 available packets of 52 byte payload, and in case of the Porcupines, we have selected two representative phases of 2.203 packets of 42 bytes each, where the first one was recorded during wearer activity (termed *activity phase*) and the second one when the wearer was asleep (*sleep phase*). While the two former data sets are physical measurements from sensors deployed for environmental monitoring, with readings changing smoothly over time, the latter are taken from motion sensors attached to a human and thus reflect both phases of sudden motions and steadiness. Representative excerpts of the four data sets are plotted in Fig. 1 for reference. It should be noted at this point that only five different symbols are present in the entire data stream in the Porcupine sleep phase, whereas the active Porcupine data set is composed of 89 different values. Glacsweb makes use of 185 different symbols, and PermaSense spans the entire input symbol range of 256 values.

To attain an estimate for the compressibility of the data sets, we show the analysis of their symbol distributions in Fig. 2, showing that the occurrence frequencies of the used symbols are not distributed evenly over the data set. In contrast, the data sets rather expose a number of subset of symbols with significantly greater occurrence numbers. The cumulative distribution function

[3] `http://tik42x.ee.ethz.ch:22001`

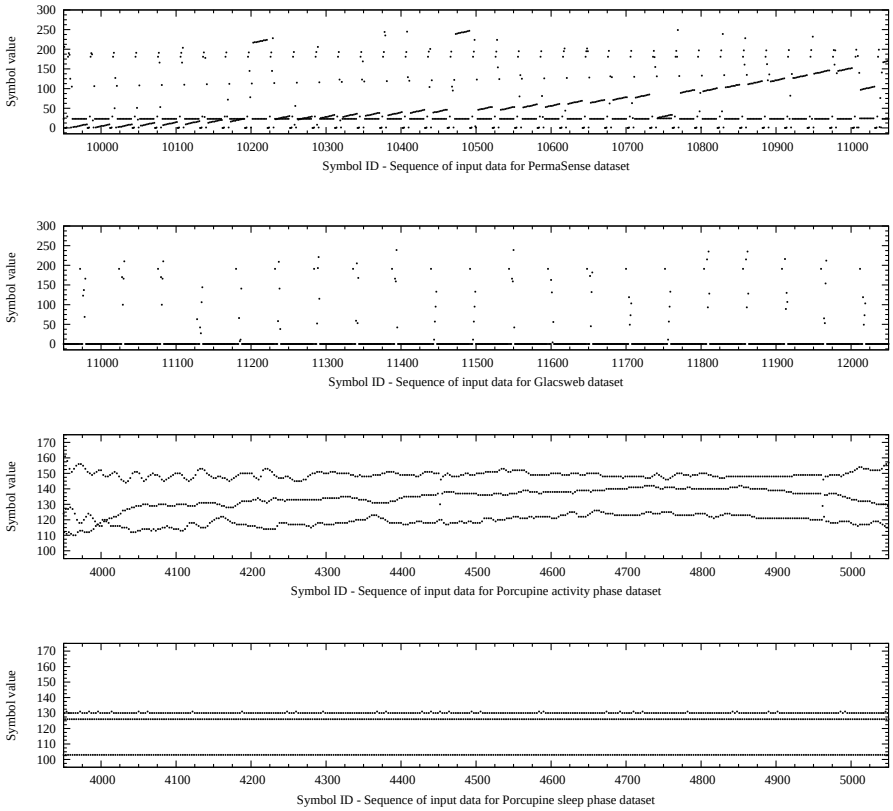


Fig. 1. Representative excerpts of the used data sets

of the symbols, which is also shown in the figure, also indicates that only a fraction of the contained symbols show frequent occurrences, while the remaining symbols have almost negligible occurrence numbers.

3.1 Huffman Coding Revisited

The foundation of Huffman coding is the assignment of codes to input symbols, with their length being reciprocal to their occurrence frequency within the input stream. In static Huffman coding [19], the input sequence is analyzed prior to encoding, and occurrence frequencies of all contained symbols are determined. On completion of this process, a tree is constructed, containing mappings for all input symbols to their corresponding Huffman code. This tree must be sent to the receiver before the actual data is transmitted to ensure both parties operate on the same dictionary. This represents additional overhead, which is however generally encountered by a near-optimal adaptation to the input sequence. The

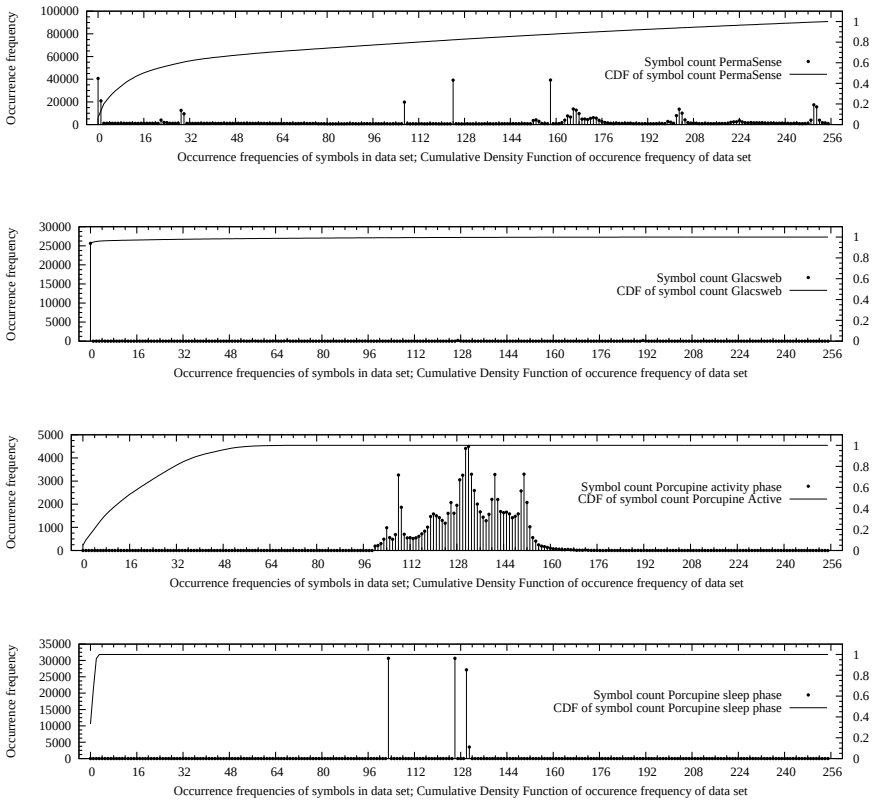


Fig. 2. Symbol distributions for the used data sets

major drawback when using static Huffman coding is the required full knowledge of the data, which strongly limits its applicability in sensor networks, where sensor readings become available periodically. In such case, the algorithm needs to operate on individual packets, and thus transmit the code table in each of them.

Adaptive Huffman coding is based on the maintenance of a the code table in a dynamic way [4]. In contrast to static Huffman coding, where this table is generated prior to the actual encoding step, AHC assigns (and possibly modifies when occurrence frequencies change) the code tree during runtime. To allow for these dynamic adaptations to occur, a dedicated placeholder symbol for an input symbol not yet encountered (NYE) is part of the code tree. This symbol is always maintained with an occurrence frequency of zero and thus always assigned one of the longest codes. Whenever a symbol not yet present in the Huffman table needs to be transferred, the NYE symbol is transmitted, followed by the unencoded representation of the symbol. The symbol is then added to the code tables of both parties, so its newly assigned code can be used on its next occurrence.

3.2 Estimation of Compression Gains

In Fig. 2, the cumulative distribution functions for the studied real-world sensor data indicate that the full range of input symbols is dominated by symbols with few occurrences within the data stream, whereas only a subset of symbols with high occurrence frequency is present. To estimate the compressibility of the data, we evaluate the resulting output sizes when only a subset of symbols is being compressed while all remaining symbols are sent unencoded.

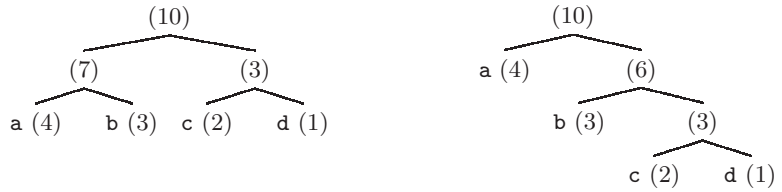
Let us assume that a compression algorithm can encode n symbols of the size of a byte, leaving the remaining $256-n$ symbols uncompressed. We furthermore assume that γ_i represents the number of occurrences of the byte value i in the input sequence, and that $f(i)$ is the function that assigns a code length (in bits) to this symbol. In case of an uncompressed transmission, $f(i)$ would statically be assigned a value of eight bits. Given these definitions, the length l of the output sequence resulting from the data compression step can be calculated as shown in Eq. 1, which sums the lengths of each symbol's code multiplied by the number of its occurrences within the input sequence.

$$l = \sum_{i=1}^{256} f(i) * \gamma_i \tag{1}$$

Symbol-oriented compression schemes, such as Huffman coding, create the code length function $f(i)$ from the state of their code table. To assess if compression with a reduced number of entries in the code tree is feasible, we have used two approximation functions for code lengths; while f_e in Eq. 2 assumes an equal length for the symbols that are encoded, f_f in Eq. 3 assigns the lengths of the output codes to follow the symbol's rank $r(i)$ within the occurrence frequency list. Code trees for both functions are also depicted in Fig. 3.

$$f_e(i) = 1 + \begin{cases} \lceil ld(n) \rceil & \text{if} \quad i <= n \\ 8 & \text{if} \quad i > n \end{cases} \tag{2}$$

$$f_f(i) = 1 + \begin{cases} r(i) & \text{if} \quad i < n \\ n-1 & \text{if} \quad i = n \\ 8 & \text{if} \quad i > n \end{cases} \tag{3}$$



(a) Code tree following f_e distribution (b) Code tree following f_f distribution

Fig. 3. Trees for f_e and f_f with $n = 4$, resulting from the input sequence `aaaabbbccd`

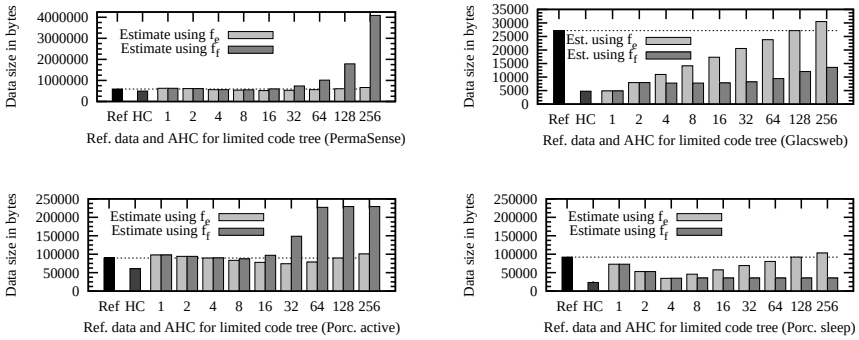


Fig. 4. Compression gain estimates for the data sets using f_e, f_f, and Huffman coding

When only a subset of the possible input symbols is present within the table mapping from input symbol to corresponding code, an additional indicator is required to mark the following bits as plaintext or encoded symbol. We have selected a one bit prefix to allow for this distinction, which is also reflected in the two functions. The results for this preliminary analysis are shown in Fig. 4, which additionally indicates the compression gains when using static Huffman coding to put the results into perspective. Although clearly indicating that savings can be achieved even when using the presented non-ideal code length distributions, the compression gain shows a strong dependence on the used data set.

As the Glacsweb and Porcupine (sleep mode) data sets only expose a small number of symbols with high occurrence frequency, the f_f function presents a better basis to achieve high compression gains, as very short codes are assigned to the most frequently occurring symbols. This way, gains of 82% are achieved for Glacsweb (at n=1), and up to 62% for the Porcupines (at n=4). In contrast, the active Porcupine and PermaSense data sets contain a larger number of frequent symbols, which are not covered well by the ranking performed in f_f. When applying f_e instead, compression gains of 17.3% (at n=32) for the active Porcupine phase, and 12% for PermaSense (at n=16) can be determined.

4 Adaptive Huffman Coding in Sensor Networks

As outlined in Sec. 3.1, a Huffman code tree must be stored for each communication link, with each of the nodes in the tree containing information about the symbol it represents, its occurrence frequency, its status (e.g., root, leaf, or NYE) as well as the identities of its children nodes and its parent. As $2n-1$ nodes are required to allow for n code entries in a tree, 511 nodes must be stored within the tree to allow for mappings of 256 input symbols. This number requires nine bits to be represented and thus two bytes on any byte-aligned microcontroller. As each tree node needs to store six bytes for its parent and child identities as well as the input symbol it represents, its frequency and status information, a

minimum of nine bytes are consumed. In summary, this results in a demand of more than four kilobytes of RAM for a Huffman tree storing 256 symbols. Besides the tree itself, a table for the occurrence frequencies of input symbols must be maintained, consuming another 256 bytes at least. This theoretical analysis also confirms the behavior observed in Guitton's implementation [5], where the memory consumption of the code tree disallowed us to instantiate more than one connection. Additionally, whenever a packet is sent or received, the Huffman tree must be updated according to its new occurrence frequency by a number of swap operations, which pose computational overhead.

The analysis of the resource demands of AHC has shown its limited applicability in WSNs due to the excessive resource demands, but also resulting from the lack of dynamic memory allocation schemes in TinyOS [20]. When operating on statically assigned memory, worst case behavior needs to be assumed for the assignment of memory during compile time, i.e. memory needs to be reserved for all symbols, including those that never occur within the input sequence.

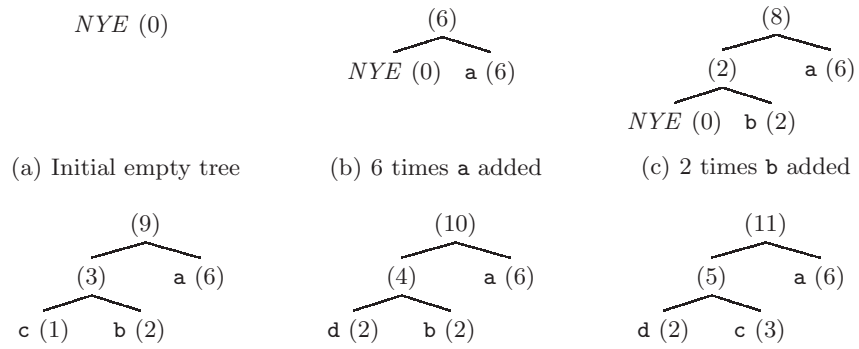
4.1 Trimming the Tree

Our observations show that the memory consumption and thus the applicability of the AHC implementation on WSN nodes is mainly limited by the number of symbols that are stored in the Huffman tree. However, as discussed in Sec. 3.2, the symbol occurrence frequencies of traffic in current WSNs are often strongly biased towards a small subset of symbols, while the remaining input characters might only rarely or never be part of the input string. Our preliminary estimations of the achievable compression gain, as shown in Fig. 4, confirm that packet size reductions are possible when only a subset of symbols are stored within the Huffman tree, while the remaining ones are transferred unencoded. The selected estimation functions were however neither adaptive to the traffic (i.e., a priori knowledge about the whole data set was required), nor did they match the characteristics of the traffic precisely.

As the memory consumption of the code table is linearly dependent on the number of entries stored within the table, keeping only a subset of input symbols in the tree can significantly reduce its memory requirement. Besides, when a smaller number of node IDs must be stored, their size can also be reduced (an 8 bit wide node ID field is sufficient to store up to 128 symbols in the tree). As a third benefit, the time to restructure the tree when changes in the occurrence frequencies are encountered also depends on the number of entries, and can in consequence be improved by reducing the tree size. In the following, we analyze the effects of confining the Huffman code tree to a limited number of entries.

4.2 Populating the Tree

The main difference between our proposed approach and conventional adaptive Huffman coding lies in the process of populating the tree. While in AHC, the NYE node is always present to attach unknown symbols to the tree, the limitation of the number of tree nodes in our algorithm can lead to situations where



(a) Initial empty tree (b) 6 times `a` added (c) 2 times `b` added

(d) 1 `c` added, NYE replaced (e) 2 `d` added, `c` replaced (f) 2 `c` added, `b` replaced

Fig. 5. Populating a tree with capacity for 3 symbols with the sequence `aaaaaabbcddcc`

the NYE node, with its assumed occurrence frequency of zero, is being replaced by a symbol. We encounter this situation by keeping track of the occurrence frequencies of the symbols stored in the tree, and replacing the element with the smallest occurrence frequency in case a more frequent symbol is encountered.

We depict the operation of the proposed implementation in Fig. 5, where an input sequence of `aaaaaabbcddcc` and a tree capacity of 5 nodes (equalling 3 symbols) is assumed. The nodes in the tree are labeled with the symbols they represent as well as their occurrence counter. In the initial phase (Fig. 5a–c), updates to the code tree are performed identical to AHC, i.e. either the counter of a symbol present in the tree is incremented, or a new symbol is added to the tree through the NYE node. In Fig. 5(d) however, the new input symbol `c` is encountered in the input sequence, while the limited number of nodes disallows the NYE to create a new tree node for the symbol. In contrast to AHC, our approach replaces the NYE by the symbol node; the tree thus loses the inherent capability of being extended through the NYE node. To still adapt to the input sequence during runtime, we follow the approach of replacing the node with the smallest counter value when a symbol with greater counter is present, such as shown in Fig. 5(e) and 5(f). To allow for this, we keep track of all symbol occurrence frequencies during runtime. All resulting codes are prefixed by a single bit indicating if the following bit sequence should be interpreted as a code from the Huffman tree or as an unencoded symbol. Assuming the tree state depicted in Fig. 5(f), the letter `c` would thus be encoded as the binary code `101`, where the `1` bit indicates that the following bits are taken from the code table, and the `01` bits refer to the branches taken to reach the value (`0`: left, `1`: right). Similarly, symbols not contained in the table, like the numeric digit `2` can be represented as `000100010`, where the first `0` bit indicates that it is followed by an unencoded symbol, and the `00100010` bits contain the ASCII representation of the digit.

The limited code tree size reduces the algorithm's resource demands significantly, as only codes for the most frequently occurring input symbols are stored, and less memory and computation time is required when reorganizing the table.

Especially, as each sensor node needs to maintain a Huffman table for each connection, the proposed reduction in terms of memory consumption is essential to successfully apply AHC in WSNs. Still, the adaptive character is maintained, allowing for high compression gains.

5 Analysis and Evaluation

Concluding from the compression gain estimates presented in Sec. 3.2, it is apparent that size reductions can already be achieved when using simplified code length approximations while limiting the number of entries within the tree. In consequence, we have presented the design of an adaptive Huffman coding algorithm that operates on a limited code tree size. In this section, we analyze its compression gains when applied to the data sets introduced in Sec. 3. Secondly, we show the algorithm's applicability on sensor node hardware by evaluating both its resource and energy demands. In a third and final step, we verify the applicability of our algorithm and energy-efficiency in a real-world experiment.

5.1 Analysis of the Compression Gain

We have compressed the four presented data sets with the algorithm and varied the parameter n, indicating the number of symbols that can be stored in the tree. We show the sizes of the compressed sequences in Table 1 in comparison to the uncompressed data, which we use as reference for all following analyses.

Table 1. Output sizes in bytes (and ratio to input) for AHC with limited tree size

#Symbols in tree (n)	PermaSense	Glacsweb	Porcupines	
			active	sleep
Reference	591930 (1.0)	27144 (1.0)	89754 (1.0)	89754 (1.0)
1	625211 (1.06)	4903 (0.18)	91172 (1.02)	72816 (0.81)
2	595944 (1.01)	7929 (0.29)	88487 (0.99)	49835 (0.56)
4	567247 (0.96)	7794 (0.29)	84249 (0.94)	34504 (0.38)
8	539434 (0.91)	7766 (0.29)	79065 (0.88)	34940 (0.39)
16	517086 (0.87)	7759 (0.29)	74431 (0.83)	34940 (0.39)
32	510933 (0.86)	7772 (0.29)	70931 (0.79)	34940 (0.39)
64	519592 (0.88)	7807 (0.29)	71884 (0.80)	34940 (0.39)
128	537240 (0.91)	7869 (0.29)	71972 (0.80)	34940 (0.39)

Notably, the achievable compression gains show a strong correlation to the used data set and its characteristics. However, the number of entries in the code tree also has a major impact on the compression gain. While very small values for the symbol count n allow to encode predominant symbols in a very efficient way, the one bit prefix increases the encoded length of all other symbols. Especially in the PermaSense and active Porcupine data sets with many different contained symbols, this even leads to size increases of the output for certain configurations

Table 2. Resource consumption of AHC with limited tree size compared to reference

#Symbols in tree	Ref	1	2	4	8	16	32	64	128	256
Flash (bytes)	22800	23838	23932	23936	23936	23936	23936	23936	23926	23918
	46.3%	48.5%	48.7%	48.7%	48.7%	48.7%	48.7%	48.7%	48.7%	48.7%
RAM (bytes)	5086	6122	6138	6170	6234	6362	6618	7130	8154	10202
	49.7%	59.8%	59.9%	60.3%	60.9%	62.1%	64.6%	69.6%	79.6%	99.6%

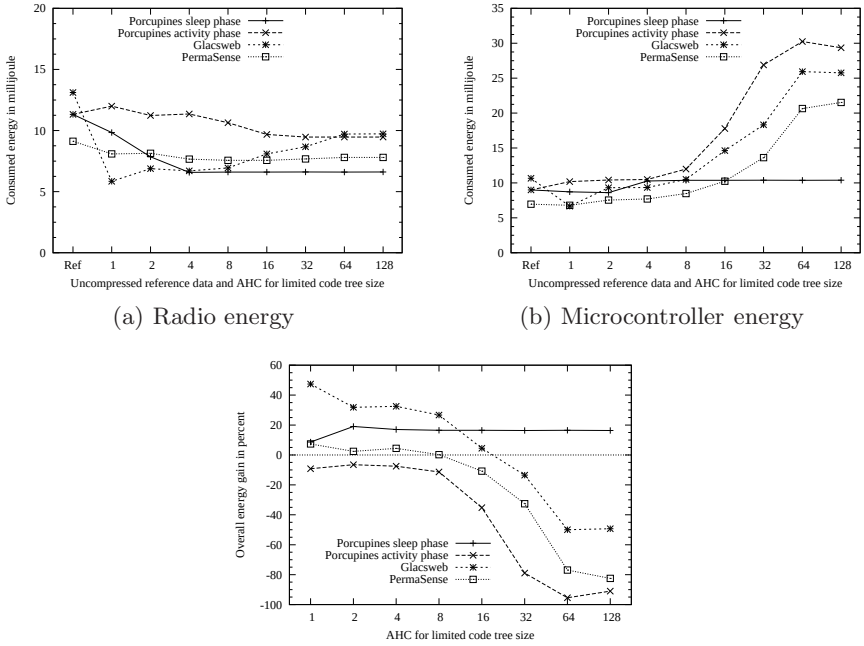
of n. In contrast, if too large values for n are chosen, the compression gain slightly degrades as a result of the longer code lengths of rarely occurring symbols.

5.2 Applicability on WSN Hardware

Before analyzing the algorithm's overall energy consumption, its applicability on current node hardware has been investigated. We have selected the *tmote sky* platform as our reference, comprising a TI MSP430 microcontroller (MCU) with 48 kilobytes of program Flash and 10 kilobytes of RAM [21]. This platform also acts as the basis for all further analyses in this paper. To assess the resource consumption, we have implemented a simple application in the Contiki operating system [22], which periodically takes sensor readings and transmits them over the radio. We have compared our variant of the adaptive Huffman coder to the reference implementation without compression functionality. Results for the required amount of Flash and RAM are shown in Table 2 and indicate that the additional amount of resources required by our implementation stays within reasonable limits when less symbols need to be stored within the tree, even though an array containing all symbol frequencies is required. With less than an 1,150 bytes increase in the program memory consumption, and an overhead of 8 bytes per Huffman table node, the algorithm proves applicable on the used sensor node hardware, leaving sufficient resources available to the application.

5.3 Energy Analysis

If we consider the computational efforts required to process input symbols and accordingly restructure the code tree, possible size reductions of radio packets might be counterbalanced by additional expenses for the processing. To evaluate the algorithm's energy efficiency on real sensor node hardware, we have performed a detailed energy simulation using MSPsim and COOJA [23] with the corresponding NullMAC protocol implementation (i.e., the radio transceiver of the receiver node is always active, so the sender radio only needs to be switched on during packet transmissions). As discussed in Sec. 1, this particular choice of the MAC protocol has been made to evaluate the algorithm's energy demand independently of any additional effects introduced by the MAC protocol. The *sky* node type has been selected, as it also represents the platform we base our practical experiment on. To allow for reproducible results, we have statically supplied the data sets to the simulated application, and assumed a lossless wireless channel as a detailed analysis of the impact of real-world channel characteristics is



(a) Radio energy

(b) Microcontroller energy

(c) Overall energy gain

Fig. 6. Energy analysis for the adaptive Huffman coder with limited tree size

beyond the scope of this paper. Assuming a single-hop transmission at a rate of ten packets per second, we have analyzed the energy requirements of the sender node only, as only marginal changes occur to the receiver's energy consumption when its radio device is not duty-cycled. We have analyzed the algorithm's energy consumption and show the corresponding results in Fig. 6. Analog to [23], we use the current consumptions measured by Dunkels et al. in [24] for our analysis. We assume an operating voltage of 3V, and radio current consumptions of 20mA in listening, 17.7mA in transmission, and 21μA in the inactive state. For the remaining platform, we have assumed 1.8mA in the active, and 5.1μA in the sleep mode.

It is evident that the use of trees with a limited number of nodes can effectively lead to reductions in the packet sizes, as observed through the reduced amount of energy spent on radio transmissions in Fig. 6(a). It can be seen that savings in radio energy of more than 50% are achieved for the Glacsweb and Porcupine sleep data sets. In case of the PermaSense and both Porcupine data sets, the reduced packet sizes lead to a consistent decrease in radio energy. Only in case of Glacsweb data, the great number of input symbols with low frequency leads to the assignment of long codes, resulting in a degraded compression ratios when larger code tree sizes are used. On the contrary, an increase in MCU utilization occurs due to the additional processing needs, as shown in Fig. 6(b). Again, the Porcupine sleep data sets exposes behavior different to the other ones,

as only five symbols need to be placed in the tree. For the other data sets, a rise in the MCU energy demand is clearly visible, indicating the increased amount of energy required for for management and restructuring of the trees. The overall energy requirements, depicted in Fig. 6(c) however still prove that for the limited code tree size adaptive Huffman coder, energy gains can be observed for three of the four data sets when appropriate tree sizes, i.e. sizes in the range of 1 to 16 symbols, are chosen.

5.4 Real-world Experiment

To verify if the simulation results match the algorithm's real behavior, we have set up a real-world experiment using two tmote sky devices. The first node was configured as a sender node and supplied with the Glacsweb data set. Blocks of data were read from the Flash memory, compressed using the presented adaptive Huffman coder with limited code tree sizes, and transmitted over the radio. To limit the energy budget available to the node, we have connected its battery terminal to a boost converter powered by a supercapacitor. To allow for comparable measurements, we have put the same charge on the supercapacitor prior to each run of the experiment. A receiver node with no energy restrictions was also part of the experiment, and was used to count the number of transmitted packets in the used indoor environment. Both were configured to use NullMAC, thus allowing to compare the results to the previously performed analyses. The results of the real-world experiment with the Glacsweb data set are indicated in Table 3 and confirm that the algorithm's behavior on real hardware resembles the observed energy simulations for the given data.

Table 3. Number of packets transmitted using the AHC coder with limited tree size

#Symbols in tree	Ref	1	2	4	8	16	32	64	128
Sent packets	4733	6832	6668	6609	5991	5947	4979	4496	2581
Runtime gain	0%	44.3%	40.9%	39.6%	26.6%	25.6%	5.2%	-5.0%	-45.5%

6 Conclusion

In this paper, we have investigated the traffic characteristics of wireless sensor networks, and determined highly non-uniform symbol distributions in packet payloads; in all of our analyzed data sets, the better part of packets is comprised of a small number of different symbols only. We have shown that encoding these symbols in an efficient way, i.e. by applying adaptive Huffman coding, considerable compression gains can be achieved. To improve the applicability of existing adaptive Huffman coding algorithms on wireless sensor nodes, we have presented a lightweight version of the AHC algorithm, operating on Huffman code trees with a limited number of nodes. Our simulation results show that even when only a small number of symbols are stored in the code tree, overall energy gains

can be achieved while maintaining the algorithm's applicability on sensor nodes. Our observations from a real-world experiment confirm these simulation results.

When application level data needs to be compressed, solutions that target to compress large chunks of data at a time are often unsuited for WSNs. While compression solutions for a dedicated application might allow for significant savings, they require developers to spent time and efforts on the implementation and integration. To take this burden off the programmers, we have shown that generic solutions can be designed to yield high compression ratios while being energy efficient, even when the structure of data is unknown in advance.

It is common knowledge that links in WSNs are susceptible to packet losses and variable link qualities [25]. Those issues have been addressed by existing data compression mechanisms using retransmissions [6] or fault tolerance extensions [5]. Although not directly related to the algorithm design, we plan to integrate suitable means to cope with the characteristics of real radio channels.

Acknowledgment

We would like to thank Kristof Van Laerhoven for providing more than 200 megabytes of real porcupine data, and Kirk Martinez, who supplied us with the data sets from Glacsweb. Last, but not least, our thanks go to the PermaSense project, which offers its sensor data traces for download. This research has been supported by the German Federal Ministry of Education and Research (BMBF) and the Center for Advanced Security Research Darmstadt (CASED).

References

1. G. J. Pottie and W. J. Kaiser, "Wireless Integrated Network Sensors," *Communications of the ACM*, vol. 43, no. 5, 2000.
2. Texas Instruments Inc., "CC2420: 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)," 2007. [Online]. Available: http://www.ti.com/lit/gpn/cc2420
3. J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.
4. J. S. Vitter, "Design and Analysis of Dynamic Huffman Codes," *Journal of the Association for Computing Machinery*, vol. 34, no. 4, 1987.
5. A. Guitton, N. Trigoni, and S. Helmer, "Fault-Tolerant Compression Algorithms for Delay-Sensitive Sensor Networks with Unreliable Links," in *Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems (DCOSS)*, 2008.
6. C. M. Sadler and M. Martonosi, "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys)*, 2006.
7. H. Ju and L. Cui, "EasiPC: A Packet Compression Mechanism for Embedded WSN," in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2005.

8. N. Tsiftes, A. Dunkels, and T. Voigt, "Efficient Sensor Network Reprogramming through Compression of Executable Modules," in *Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2008.
9. J. Chou, D. Petrović, and K. Ramchandran, "A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2003.
10. A. Reinhardt, M. Hollick, and R. Steinmetz, "Stream-oriented Lossless Packet Compression in Wireless Sensor Networks," in *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2009.
11. A. Reinhardt, D. Christin, M. Hollick, and R. Steinmetz, "On the Energy Efficiency of Lossless Data Compression in Wireless Sensor Networks," in *Proceedings of the 4th IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp)*, 2009.
12. A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.
13. P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein, "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with Zebranet," in *Proceedings of the 10th Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002.
14. Y.-C. Tseng, S.-P. Kuo, H.-W. Lee, and C.-F. Huang, "Location Tracking in a Wireless Sensor Network by Mobile Agents and Its Data Fusion Strategies," in *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.
15. J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel, "PermaDAQ: A Scientific Instrument for Precision Sensing and Data Recovery in Environmental Extremes," in *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.
16. V. Annamalai, S. K. S. Gupta, and L. Schwiebert, "On Tree-Based Convergecasting in Wireless Sensor Networks," *IEEE Wireless Communications and Networking*, vol. 3, 2003.
17. K. Martinez, R. Ong, and J. Hart, "Glacsweb: A Sensor Network for Hostile Environments," in *Proceedings of the 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004.
18. K. van Laerhoven, H.-W. Gellersen, and Y. G. Malliaris, "Long-Term Activity Monitoring with a Wearable Sensor Node," in *Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, 2006.
19. J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "A Locally Adaptive Data Compression Scheme," *Communications of the ACM*, vol. 29, no. 4, 1986.
20. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Network Sensors," in *Proceedings of the 10th Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.
21. *tmote sky data sheet*, Moteiv corporation, 2006, http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf.

22. A. Dunkels, B. Grönvall, and T. Voigt, "Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the 1st IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, 2004.
23. J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, "COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks," in *Proceedings of the 2nd International Conference on Simulation Tools And Techniques For Communications, Networks And Systems (Simutools)*, 2009.
24. A. Dunkels, F. Österlind, N. Tsiftes, and Z. He, "Software-based On-line Energy Estimation for Sensor Nodes," in *Proceedings of the 4th Workshop on Embedded Networked Sensors (EmNets)*, 2007.
25. R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a Sensor Network Expedition," in *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN)*, 2004.