Compression of Short Text on Embedded Systems

Stephan Rein, Clemens Gühmann, Frank Fitzek*
Technical University of Berlin, Dept. of Electronic Measurement and Diagnostic Technology
*Aalborg University, Dept. of Telecommunication Technology
{rein, guehmann}@tu-berlin.de ff@kom.aau.dk

***Abstract*— The paper details a scheme for lossless compression of short data series larger than 50 Bytes. The method uses arithmetic coding and context modeling with a low-complexity data model. A data model that takes 32 kBytes of RAM already cuts the data size in half. The compression scheme just takes a few pages of source code, is scalable in memory size, and may be useful in sensor or cellular networks to spare bandwidth. As we demonstrate the method allows for battery savings when applied to mobile phones.**

***Index Terms*— Arithmetic coding, context modeling, prediction by partial matching (PPM), short message compression, embedded system, mobile phone, sensor network**

I. Introduction

Lossless data compression, also referred as text compression, is widely employed on personal computers to allow for efficient computer data storage on hard drives or shared media and for sharing data via the Internet. Text compression may also be useful for wireless systems like cell phones or sensor networks, thus achieving bandwidth savings and reduced costs, or even enabling the wireless medium to be shared by more devices. These devices often employ micro-controllers and demand for low-complexity data compression algorithms that cope with memory and computational constraints. The data to be compressed may be short messages to be exchanged between phone users or signaling data in wireless networks. In sensor networks, a text compressor also may be useful for specific sensor data and program/source code update.

There exist two main classes for text compression, the dictionary and the statistical coding technique (there also exist methods for images and waveforms - i.e.,sounds, but these are not discussed in this paper). A dictionary coder searches for a match between a part of the text and a string in its dictionary. If a match is found, the text is substituted by a reference to the string in the dictionary. Most of the dictionary coders are based on the *Ziv-Lempel* methods (LZ77,LZ78) and are widely employed to compress computer data. The statistical coders encode each symbol separately taking their context - i.e., their previous symbols into account. They employ a statistical context model to compute the appropriate probabilities. The probabilities are coded with a Huffman or an entropy coder. The more context symbols are considered, the smaller are the computed probabilities and thus this may result in a better compression. The statistical coders give better compression performance than the dictionary coders, however, they generally require large amounts of random access memory (RAM).

In this paper we discuss the statistical coding technique *prediction by partial matching* (PPM), which employs a statistical context model and an arithmetic coder. We intend to develop and to apply a low-complexity version of PPM that a) demands for low memory, b) is conceptually very simple - that is, it can be implemented with a few pages of source code, and c) can compress very short data sequences starting from 50 Bytes. We have selected the PPMC version of PPM from [1] as a basis for our research investigations because it gives good compression while it is conceptually simple. This version uses a specific scheme to signal *Escape* probabilities (the so called method C) and an upper bound on memory by rebuilding the model when there is no memory left. We develop a library for the design of low-complexity context models and apply the library to design and evaluate a context model for embedded systems. The so derived model already achieves reasonable compression results for short messages with 32 kByte RAM when appropriate statistical data is preloaded.

The paper is organized as follows. In the next subsection, we review related literature. In section II, we explain the principle of PPMC and give performance results using a scalable data structure with constrained memory. The obtained results lead to the design of a low-complexity context model for compression of short messages as detailed in section III and evaluated in section IV, which we implement to run on Java- and Symbian-based mobile phones. As we demonstrate, the application not only pays of in terms of costs for the user, but also in terms of battery savings, as less data is transceived and the compression algorithm only consumes little energy. In section V, we summarize our findings.

A. Related Work

The method of context modeling - that is, prediction by partial matching - has been extensively covered in the lit-

This paper is based on "Low-Complexity Compression for Short Messages " by S. Rein, C. Gühmann, and F. Fitzek, which appeared in the Proceedings of the IEEE Data Compression Conference'05, Utah, USA, March 2006. © 2006 IEEE.

Parts of this project have been financed by the Danish Goverment within the X3MP project.

erature, see for instance [2]–[5]. These achievements concern the improvement of compression performance while reducing the computational requirements for personal computers. In [6], an optimal statistical model (SAMC) is adaptively constructed from a short text message and transmitted to the decoder. Thus, such an approach is not useful for short message compression, as the overall compression ratio would suffer from the additional size of the context model. In [7], the scheme PPM with *information inheritance* is described, which improves the efficiency of PPM in that it gives compression rates of 2.7 Bits/Byte with a memory size of 0.6 MBytes, which makes PPM readily applicable to low-cost computers. The memory requirements for ZIP, BZIP2, PPMZ, and PPMD were shown to vary from 0.5 to more than 100 MByte.

Data compression techniques for sensor networks are surveyed in [8]. Most of the sensor data compression techniques exploit statistical correlations between the typically larger data of multiple sensors as they all observe the same phenomenon, see for instance [9], [10]. The method proposed in this paper can help sensors to efficiently exchange sensor specific signaling data.

The recent paper in [11] uses syllables for compression of short text files larger than 3 kBytes. A related work in the field of very short text files is the study in [12], where a tree machine is employed as a static context model. It is shown that zip (Info-ZIP 2.3), bzip-2 (Julian Seward version $1.0.2$), rar (E. Roshal, version 3.20 with PPMII), and paq6 (M. Mahoney) fail for short messages (compression starts for files larger than 1000 Bytes). In contrast to our work, the model is organized as a tree and allocates 500 kBytes of memory, which makes the proposed method less feasible for a mobile device. In addition, our method is conceptually less demanding, which is a key requirement for low-complexity devices.

This paper is different from our work in [13] in that the compression for the smaller data models were improved by using a modified hash function. Furthermore, a methodology for the design and analysis of low complexity data-models together with extended performance results are given, for which we make parts of our implementation publicly available in [14].

II. Library for Context Modeling

A. PPM compression

Context modeling is a technique that can be employed for data compression when combined with an entropy coder. In PPM, each symbol is coded separately taking its context with n symbols into account, where n denotes the current model order. If a symbol with context length n is not in the context model, the model iteratively switches down to the order $n-1$. Otherwise, the context model returns the symbol probability that is a *left*, a *right*, and a *total* probability on the *probability line*, as illustrated in figure 1. There exists a separate probability line for each context on which the appendant set of symbols (the symbols that belong to the given context) is stored in alphabetical order. The arithmetic coder returns a binary bit stream for each given set of symbol probabilities. Similar, the arithmetic decoder returns a number that lays in the section of the appropriate symbol on the probability line when decoding the single Bits. The context model estimates the symbol and the exact borders of the section that are the left and the right probability. These probabilities are needed by the arithmetic decoder to decode the next number. Figure 1 also illustrates that a data model is part of the context model. The data model returns the symbols count for each string (which is a symbol with its given context). Note that the figure simplifies the interaction between the two functions *GiveSymbProb* and *DecodeSymbol* and the data model. An extended data model is able to provide the counts of all existent symbols for a given context. Detailed information on PPM and a comparison to dictionary coding techniques are given in [15].

B. Hash Table Data Structure

PPM has larger memory requirements than the dictionary-based compression schemes. Thus, it is advisable to carefully design the data structure for the data and the context model, respectively. Teahan and Cleary propose a *trie*-based data structure for fixed-order models [16]. In computer science, a *trie* or a prefix tree is an ordered tree data structure to store an associative array where the keys of the nodes are strings, see [17] for a survey on data structures. A trie requires functions for traversing it to access the queried data. A hash-table technique can be faster in case of a smart hash-function that avoids collisions. For our primary evaluations, we use a hash table to manage the string data and resolve collisions by linked lists. A table element refers to a trie-node and contains a *key*, a *count*, a *total count*, and a *bit mask* entry. The key is a pointer to the string data the element refers to. The counts are needed for the on-the-fly probability calculation, see [14] for details. The total count helps the probability calculation not to traverse the rest of the symbols on the probability line if the searched symbol is found. The bit mask is a specific binary data structure that indicates the existent context for the probability calculation - that is, the set of successors of each node.

For the design and evaluation of lossless text compression algorithms with statistical context modeling, we have developed a testbed in c++, which consists of an arithmetic coder [18] and a context model library, for which the functionality is given in figure 2. The context model uses the data structure illustrated in figure 3 with linked lists for resolving collisions. To simplify statistical model refinements we use an interface to the free high-level language software *Octave*. Our complete methodology for the design of low-complexity compression algorithms is depicted in figure 4.

When the context is modeled adaptively the number of possible collisions is not yet known when the compression starts and thus the memory for the collision items would

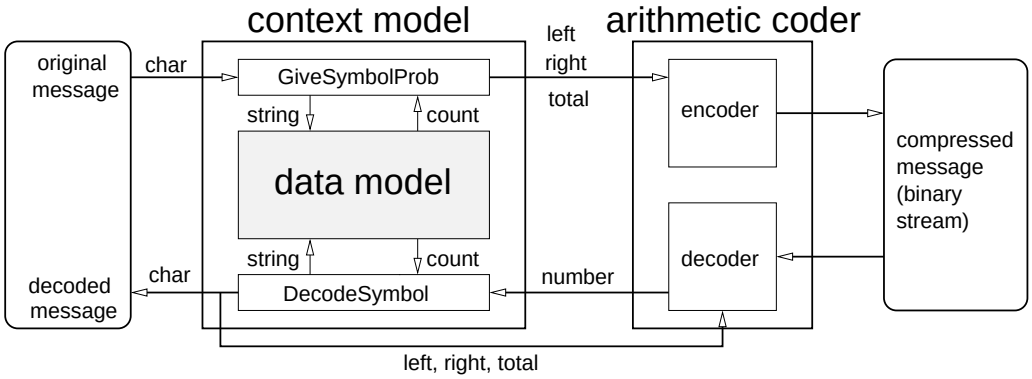


Figure 1. Principle of the PPM compression. A context model calculates the symbol probabilities for the arithmetic coder.

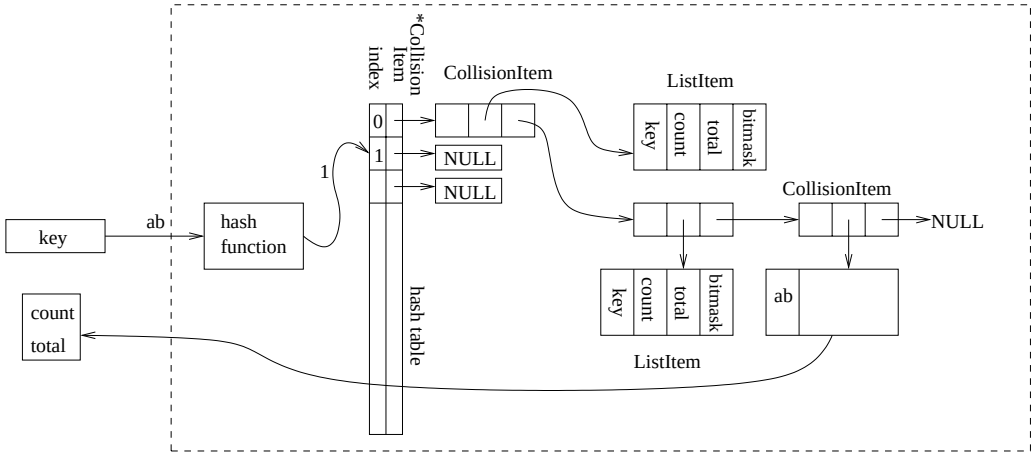


Figure 3. The data structure for the more complex data model that is employed for gathering the statistics for the low-complexity data model. Collisions are resolved by linked lists of *CollisionItems*, each of them pointing to a *ListItem*.

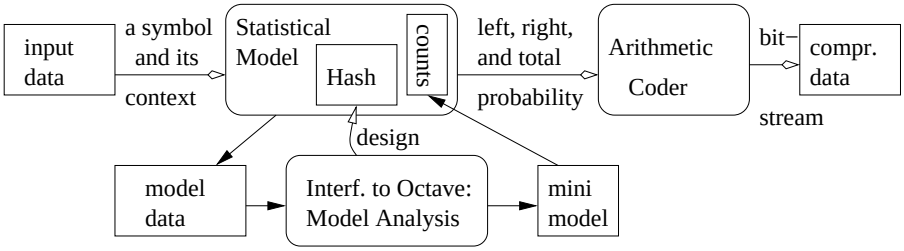
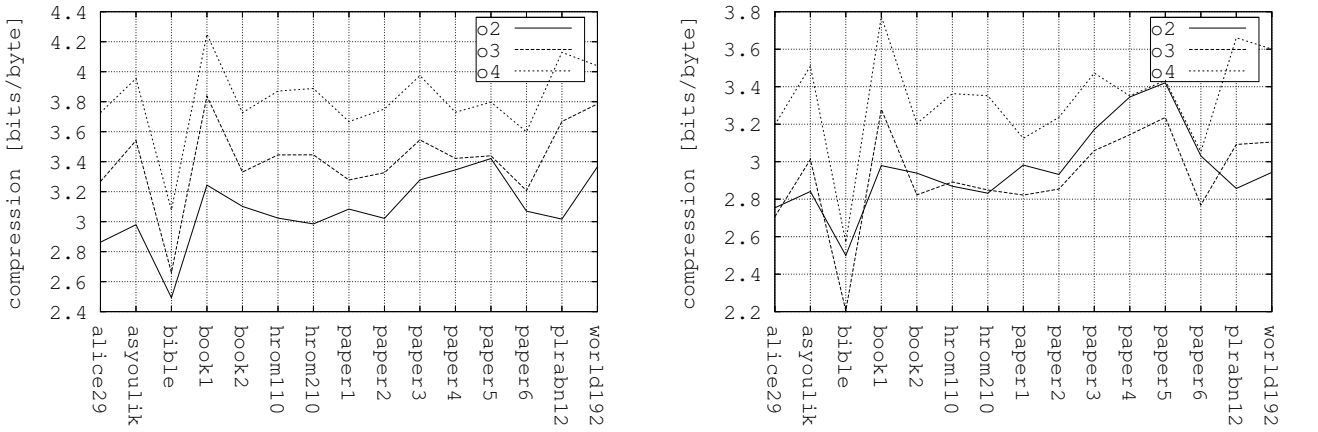


Figure 4. The testbed for designing lossless compression schemes. The statistical data is interfaced to the high-level language *Octave* for model refinements. We plan to make the Octave functions available in [19].

function	description
read model	read a model from disc
write model	write a model to disc
collisions	set the maximum number of collisions
static modeling	model is not updated throughout the compression
model order	works with each order
hash table size	should be power of 2
model state	keeps track of the type of nodes

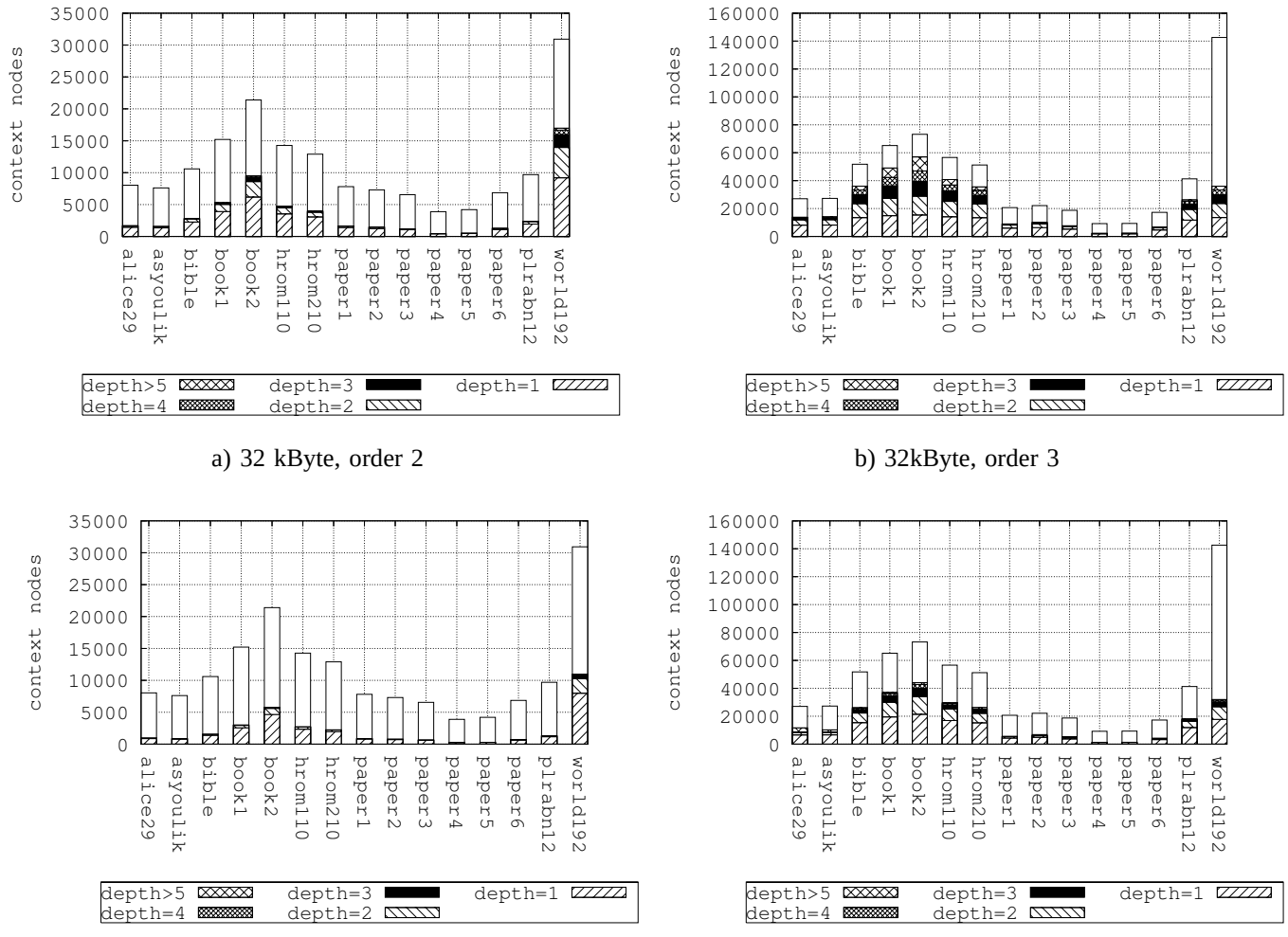
Figure 2. Some options of the context model library in C++, which allows for *adaptive, static,* and *semistatic* compression of files. The models can be modified with the software *Octave*, thus allowing a software engineer to design an optimal model for the required application.

have to be allocated on the fly. As this can be time-consuming there exist three data pools for the table elements and the string items. The memory is constrained by the maximum number of collisions. Figure 5 shows the compression for orders two and three, a table size of 16384 elements, and a maximum number of 1000 or 5000 collisions, respectively. We employ English text files from the *Canterbury* corpus [20] (*alice29, asyoulik, plrabn12*), the *project Gutenberg* [21] (*hrom110, hrom220*), the *Calgary* [22], and the *large corpus* (*bible, world192*, available at *http://corpus.canterbury.ac.nz/descriptions*). For the order two model, the compression varies from 2.5 to 4.2 Bits/Byte. Adaptive modeling gives reasonable results



a) 1000 collisions b) 5000 collisions

Figure 5. Compression results for adaptive modeling with a table of 16384 elements and model orders 2-4. If the maximum number of a) 1000 and b) 5000 collisions is attained the model is flushed. The adaptive technique gives promising results even with scarce memory, however, it is not able to compress a very short data file.



a) 32 kByte, order 2 b) 32kByte, order 3

c) 64 kByte, order 2 d) 64 kByte, order 3

Figure 7. Type of the collision context nodes within the hash table data structure with linked lists for the order two and three *book2* models with 32 and 64 kBytes. The complete length of a bin gives the total number of context nodes in the model. A *depth 4* beam area refers to the fourth collision in the list. The not-hatched area refers to the non-collision nodes.

even with constraint memory, however, compression is not possible for very short files where only scarce statistics can be gathered. The idea is now to inspect the hash table data structure for the adaptively gathered models and to derive a low-complexity data structure from it. As we want to apply the context model to an embedded system, we analyze the collision avoidance performance of the hash function. We employ the *One-at-a-Time* hash function introduced in [23], where it is evaluated to perform without collisions for mapping a dictionary of 38470 English words to a 32-bit result. The source code is given in figure 6. Figure 7 depicts the total number of

```
unsigned int HashFunction(char* key,
                  unsigned char len,
                  unsigned int TableLen,
                  unsigned int bits){
unsigned int hash,i;
for (hash=0,i=0;i<len;i++){
  hash+=key[i];
  hash+=(hash << 10);
  hash^=(hash >> 6);
}
hash += (hash<<3);
hash^=(hash>>11);
hash+=(hash<<15);
unsigned int mask=( (unsigned int) 1<<bits ) - 1;
unsigned int index=(hash & mask);}
```

Figure 6. Source code of the *One-at-a-Time* hash function.

collision nodes categorized by their depth in the linked list for 16384 and 32768 table elements with order two and three. The bars give the total number of nodes in the data structure. Collision nodes are hatched in dependence of their depth in the linked list. For order two there exist few collisions and the data structure performs fairly well. For order three the smaller hash table is exhausted. Generally, enlarging the hash table size relaxes the hash table. With respect to the total number of nodes for order three that make collisions unavoidable the hash function performs well for each order. Furthermore, there are less collisions in the deeper linked lists. We use the data structure for our further analysis and derive a heuristic scheme.

III. Low-Complexity Modeling

A. Static Modeling

Besides *adaptive* context modeling there exist *semi-adaptive* and *static* context modeling. In semiadaptive modeling the sender first transmits the codebook and then the encoded message. In static modeling sender and recipient use the same static model for each message. In adaptive context modeling the first few hundred Bytes are needed for gathering the statistics. For a short message no reasonable compression can be achieved. The semiadaptive technique is not useful either because of the larger model to be transceived. Teahan et. al use the PPM compression scheme to show that the compression decreases when the model is incrementally constructed [16]. We apply a similar idea in that we preload a model that is built by compressing a training file and preload the so obtained data to compress other text data. With such a

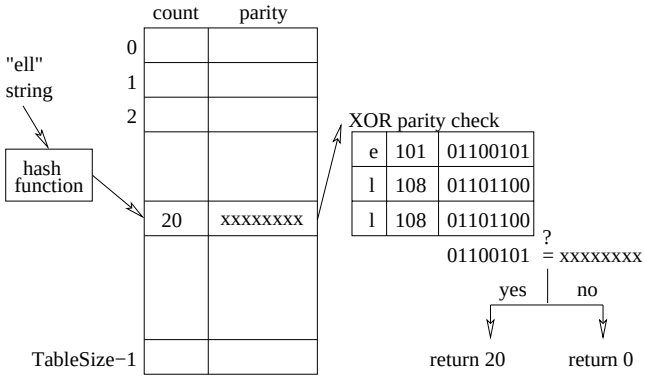


Figure 8. Elementary parts of the data model, a hash table each element containing a count and a parity, a hash function, and a parity check function. Reprinted from [13] with permission of the IEEE.

preloaded model, it should be possible to compress very short data. A set of these models can remain on a cell phone or wireless sensor as a fixed data base. Instead of transmitting the model, the sender just needs to signal which of the models should be used for decoding. In contrast to the technique in [16] we do not consider to update the model throughout the compression, which is motivated as follows:

- A short data sequence is not able to statistically update a larger data model.
- A sender may send different messages to different recipients (there would be need to keep track of the different updates and assign them to the correct entity;)
- If the model shall not be updated throughout the compression less source code is required and the compression is faster.

In the next section the context model to be applied on an embedded system is described.

B. Data Model

As illustrated in figure 7, the percentage of the collisions is fairly low compared to the total number of context nodes. We design a low-complexity context model that consists of a data model and two functions for statistical calculations. The data model returns a symbol count to any string it is queried. It works as illustrated in figure 8. First, the string is given to the hash function, which transfers the string of characters to an integer number within the interval $[0, TableSize - 1]$, where $TableSize$ denotes the total number of elements in the hash table. The estimated hash table element contains a count and a parity Byte. Then, the parity of the queried string is computed and compared with the parity in the hash table. If both parities are the same, the data model returns the count in the hash table element. Otherwise, a zero count is returned indicating that the string is not in the model. The parity check is essential as during the encoding/decoding process a large amount of strings are queried the data model for the on the fly estimation of

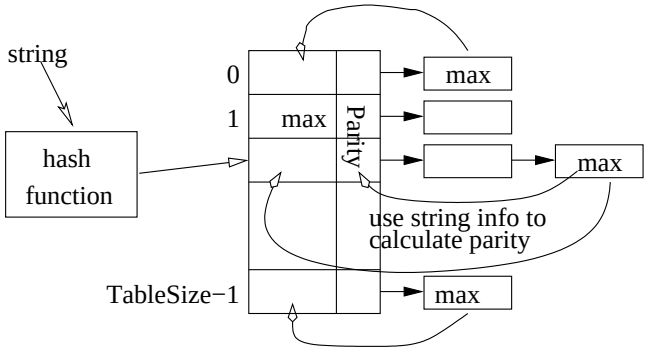


Figure 9. The figure illustrates the hash table of the original, more complex data model with collision resolving linked lists, which is employed to train the low-complexity model. Reprinted from [13] with permission of the IEEE.

file	ram [kB]	size [Byte]
o2tb-32	32	30278
o2tb-64	64	47606
o2tb128	128	63060
o2tb256	256	73260

a) order 2

file	ram [kB]	size [Byte]
o3tb-32	32	32762
o3tb-64	64	64752
o3tb128	128	117106
o3tb256	256	176490

b) order 3

Figure 10. Size of the compressed data models. The larger low-complexity models can be compressed; thus the embedded system may manage a set of different data models. The sender has to signal the receiver which model to use.

the symbol statistics. It is a collision detection technique and does not resolve collisions. We note that the lossless compression/decompression with the data model even works without the parity check, however, the compression performance is significantly deteriorated. Let N denote the model order. For each order, the hash function maps the space of 256^{N+1} different strings to an array of $TableSize$ elements, resulting in hash function collisions which are not resolved. The task of the hash function is to only map the statistically significant context nodes.

The model is built using the more complex data structure in section II. First, a model is loaded into the hash table with linked lists. For each linked list, the element with the maximum statistical count is copied to the first entry of the hash table, as illustrated in figure 9. A parity Byte of a table element is calculated by a logical exclusive-OR operation of the single symbols of the string. The remaining linked lists are deleted.

Such a data model requires the probability calculation functions of the data model to traverse all the possible symbols of a given context including the symbols that are not on the probability line. If the symbol unit is one Byte 256 symbols have to be checked. As the scheme is designed for compression of a small number of Bytes the main focus is on low memory requirements at the cost of computational speed. Each node of an arbitrary key length in the data trie is represented by just 2 Bytes. The strings are not stored anymore. With such a data model it is not possible to traverse its content like it can be done with a book. Instead, it can just can be checked for given strings.

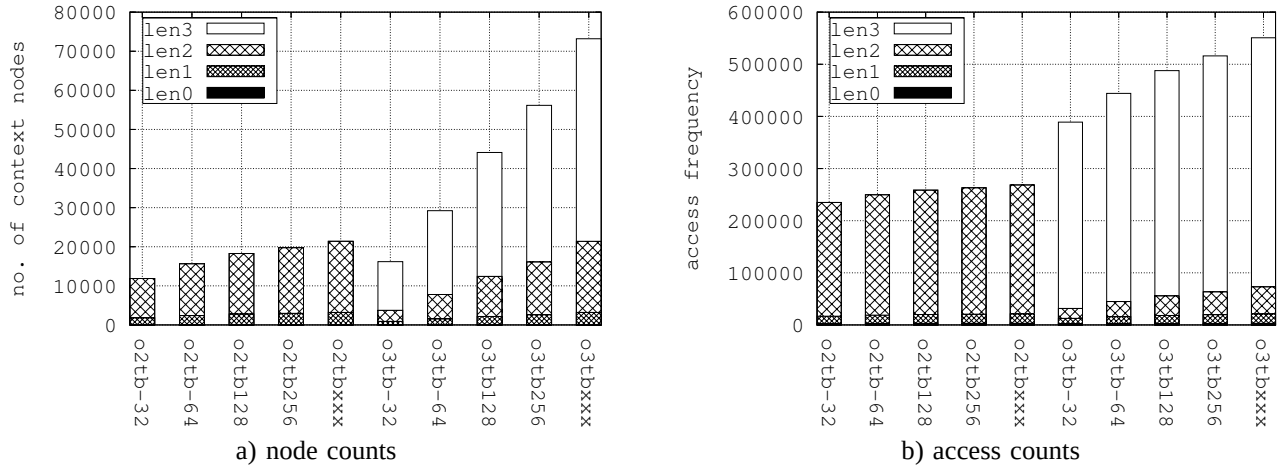
C. Compression of the model

A mobile device does not need to permanently retain the statistical model in its RAM memory. When the compression program is not needed, the statistics can be kept in the program memory. We propose to compress the array of counts and parities for efficient storage in the program memory. For our measurements, we use models with the hash table lengths 16384, 32768, 65536, and 131072 elements. As each element requires 2 Bytes (symbol count and parity), the statistical models allocate 32, 64, 128, and 256 KBytes of RAM memory, respectively. The models are constructed from the text file *book2* from the Calgary corpus [22].

A simple yet effective method for compression of the statistical data can be achieved due to the fact that even with a good hash, many of the elements of the hash table are empty and not employed for the statistical model. An empty element is defined by a zero count. If an element is empty, there is no need for storage of a parity. To store the counts and the parities the hash table is traversed from its first to the last element. For each element, the count and the parity is sequentially stored. In case of a zero count, a zero Byte and a Byte for the total number of zero elements in a stream is written on disk. Thereby, large streams of zero elements are stored by just two Bytes. Typically, there exist many streams of zero elements in the data model. The maximum stream size that can be denoted is 256 empty elements. Figure 10 gives the size of different compressed data models in Bytes for the orders 2 in table a) and 3 in table b). We use the so constructed models for our cell phone software to let the user manage a set of statistical models.

IV. Results

For construction of a static model we use the text file *book2* with the model sizes $32, 64, 128,$ and 256 kByte and evaluate them with the text files given in section II-B. To achieve a better compression it is advisable to build a model from multiple files, as Teahan and Cleary propose in [16]. Figure 11 shows that statistical data is collected throughout the complete length of the file. This trend is expected to proceed for consecutive files. Only the statistical relevant data could be collected from the so constructed model for a low- complexity data model. However, such a construction is related to a specific application where the statistical properties of the data are known in advance. We here just address the construction and application of a model in principal and leave the sophisticated model construction to the software engineer. (Nevertheless, we achieve similar compression results as detailed in section IV for English short messages written by real users of our cell phone application using the *book2* training data.)



a) node counts b) access counts

Figure 12. A context model represents the nodes of a data tree, each node referring to a different symbol with a certain context. Figure a) classifies the nodes by their context length, figure b) depicts the frequency the nodes of a certain context length are accessed. The low-complexity models retain the statistically important data.

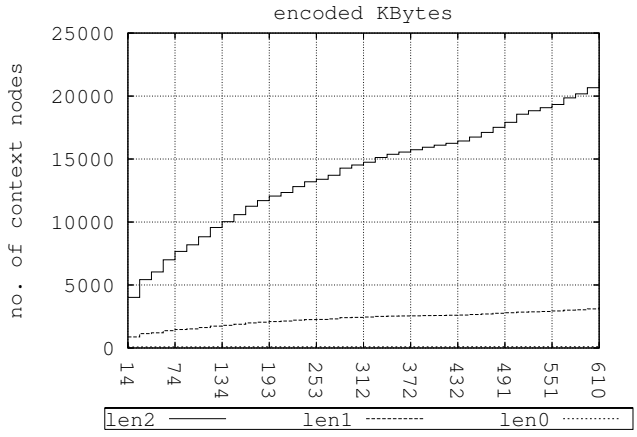


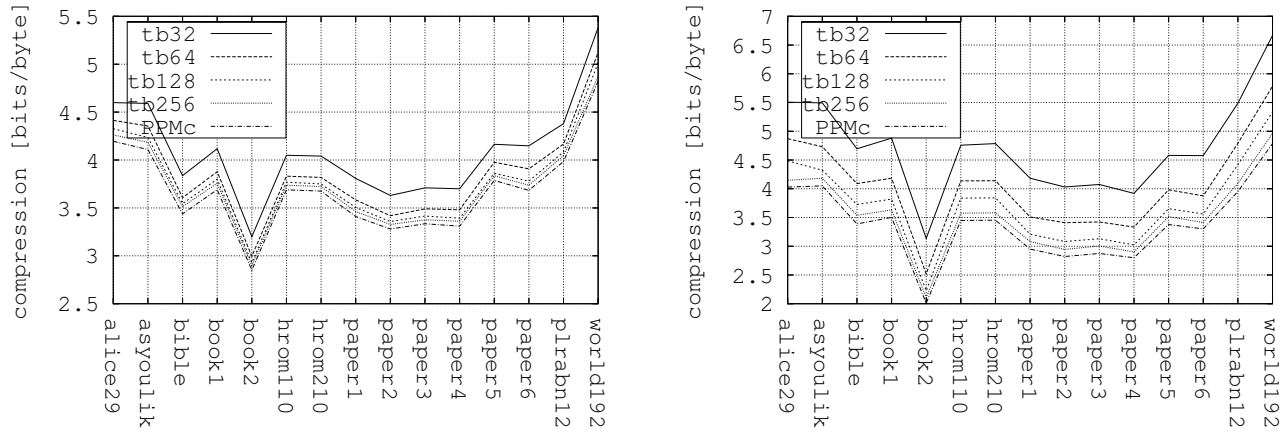
Figure 11. The *book2* file delivers statistical data over its complete length of 610 KBytes. The different node lengths (*len0*- *len1*) indicate the order of a node - i.e., a node of order 0 refers to a key with one Byte.

A. Model Performance

Figure 12 gives statistics performance results for the *book2* models with different memory requirements. The models are compared with the data model of PPMC (denoted as *tbxxx*) that is employed without memory constraints. Figure 12 a) shows the number of context nodes each model contains, where *len0* to *len3* refer to the order of the key that is mapped (for instance, len1 refers to a key of 2 Bytes and the model order 1). The models o2tb-32 and o2tb-64 preserve more than half of the original model space, which is given by o2tbxxx as 21400. The models o2tb128 and o2tb256 tend to preserve the space of occurring context nodes, and therefore are expected to result into good compression performance. For order 3, the space of possible context nodes is given as 73191. The models o3tb-32 and o3tb-64 preserve less than 30000 of the context nodes, which is just a smaller percentage of the original model space. The models o3tb128 and o3tb256 preserve 60% and 77%, respectively.

A low-complexity data model consists of a subspace of the original data model, which should contain the context nodes that are of statistical importance. An indicator to assess the statistical quality of the retained nodes may be the frequency each node is accessed - that is, the sum of the node counts, which is illustrated in figure 12 b). Especially for the order 3 models, the rising trend of figure a) is not visible. The total count for *len3* context nodes is relatively large for the smaller models even though these models do not contain so many *len3* nodes. This indicates that these nodes are statistically more weighted in the smaller models.

Figure 13 illustrates the compression results with the static model *book2* for the orders two in figure a) and three in figure b) for the complete array of text files. The models are not updated throughout the compression. The performance of our low-complexity compression scheme, which we call from now *lcc*, is compared to PPMC with preloaded models without memory constraints. For the order two model, the improvement in compression is 0.24 Bits/Byte when the memory is enlarged to 64 kByte. Doubling the memory size only gives small improvements around 0.05 Bits/Byte. The model performance of lcc - that is the ability to maintain the statistics of the PPMC model - is similar to the PPM case if the models are larger than 32 kBytes. The 32 and 64 kByte models perform significantly better for order two than for order three. The 128 kByte order three model performs slightly worse than the 64 kByte order two model. This indicates that the smaller models are exhausted for the higher order. When using order three the allocated memory for lcc should be at least 256 kBytes, where the higher order starts to pay off. For order two, the 64 kByte lcc model achieves similar compression than the PPMC model. We note that the evaluation results from figure 13 were not obtained to prove the compression performance of lcc for longer text files, but to demonstrate that the lcc context models retain the statistical significant data of the PPMC model



a) order 2 b) order 3

Figure 13. Static context modeling using the training file *book2* for a) order two and b) order three. The larger models perform similar than PPMC without memory constraints. To achieve a better compression with a higher order the model should at least allocate 256 kBytes.

without memory constraints.

B. Compression of short text

Figure 14 shows the compression ratios for short text sequences of the file *paper4* for lcc with training data from *book2*. The data points are depicted together with the 95% confidence intervals. lcc is compared with the PPMC compression scheme using adaptive context modeling. Figure a) shows that for model order two the compression rates nearly stays constant for text sequences larger than 75 Bytes, which similarly holds true for the model order three in figure b) with the exception of the 32 kByte order three model, where the compression slightly rises for messages larger than 200 Bytes. The order two models of 32, 64, and 128 kByte give the compression rates 3.9, 3.55, and 3.5 Bits/Byte, respectively. Doubling the memory size once more does not result into compression improvements. The 64 and 128 KByte order three models perform with approximatively 3.7 and 3.3 Bits/Byte, respectively. The 256 kByte order three model gives the best performance of approximatively 3.1 Bits/Byte for the larger text sequences. The results show that especially the order two lcc model gives reasonable compression performance while preserving low computational requirements. For higher model orders, the size of the hash table has to be enlarged. Compared to our work in [13], we have reduced the compression rate by more than 0.4 Bits per Byte for the 32 and 64 kByte models. The improvements result from a modification in the hash function.

We note that in the study in [12], compression ratios ranging from 2.95 to 3.1 for compression of the file *hrom210* using training data from *book1* is achieved. The better compression performance is obtained with the cost of higher conceptual complexity and memory requirements of 512 kBytes. In contrast to this approach, lcc is designed as a scalable trade-off between complexity and compression performance.

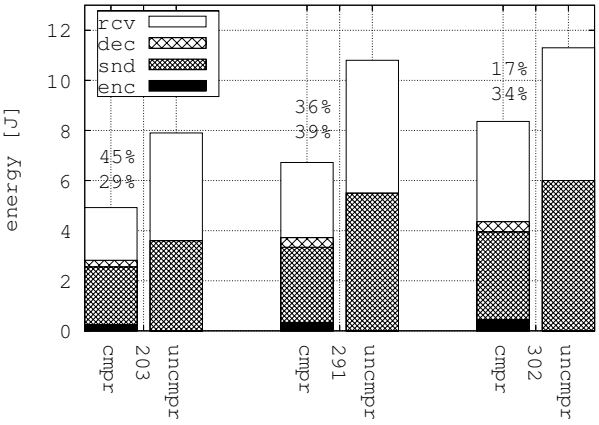
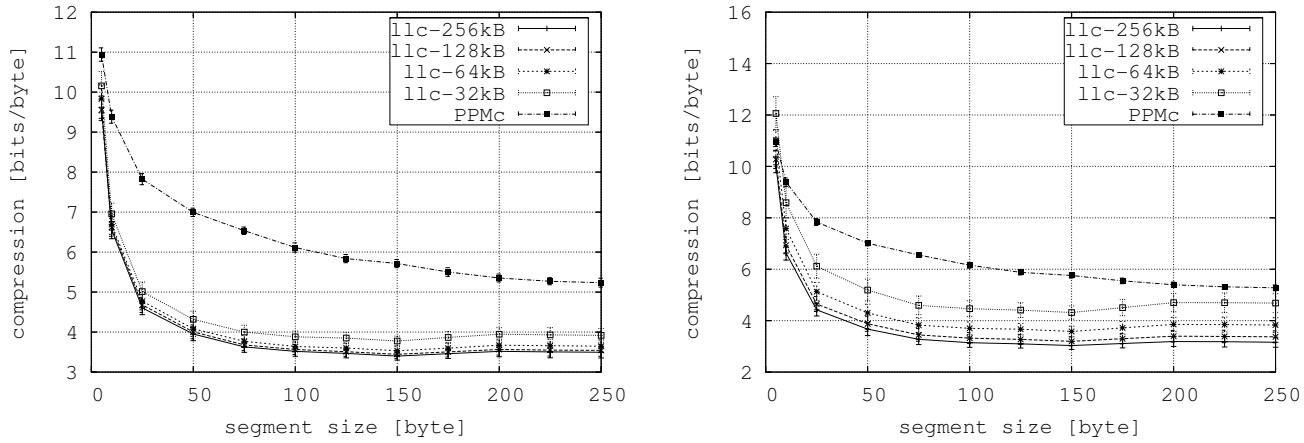


Figure 15. Operational (*enc*,*dec*) and transceiver energy (*snd*,*rcv*) needed to convey a message of different length with our compression scheme (*cmpr*) and without it (*uncompr*), respectively. The numbers on top of the bars give the battery savings in per cent for the sender and the receiver battery. Compression not only pays off in user costs and effective medium usage, it also gives battery savings for the user's devices.

C. Phone measurements

Figure 15 illustrates the required energy on the Nokia 6630 phone for encoding, sending, decoding, and receiving a message of $203, 291,$ and 302 Bytes denoted as *enc*, *snd*, *dec*, and *rcv*, respectively. The total energy for sending a compressed message (*cmpr*) is compared with the total energy for sending an uncompressed message (*uncmpr*). We denote the energy for sending or receiving a message as *transceiver* energy and the energy for encoding or decoding a message as *operational* energy. The two numbers above the bins indicate the battery savings for the sending phone (lower number) and the receiving phone (upper number) in per cent. As the operational energy is much smaller than the transceiver energy, the lcc compression scheme on average pays off by 34% in battery savings for the receiver and by 33% for the sender, respectively. Figure 16 shows the operational time



a) order 2 b) order 3

Figure 14. lcc compression performance for short messages for order 2 in fig. a) and order 3 in fig. b). PPMC without a preloaded model and the 32 kByte order 3 model fail for short messages. The other models indicate the ability of lcc to provide a scalable trade-off between compression efficiency and computational requirements.

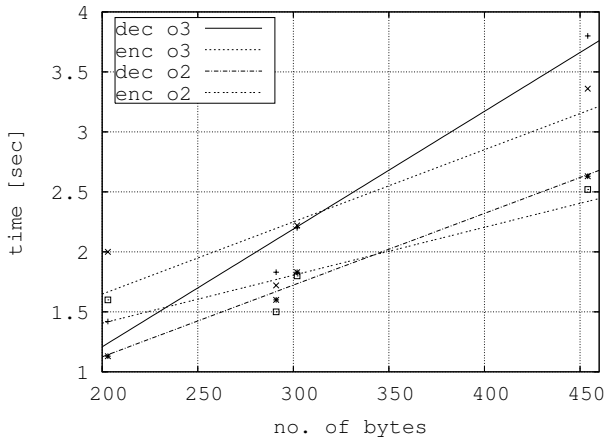


Figure 16. Time needed for encoding and decoding a message for orders 2 and 3 on the Nokia 6630.

for encoding and decoding a message of different lengths with order two and three on the Nokia 6630. When the messages are shorter than 320 Bytes the encoding time tends to be longer than the decoding time. When the messages are longer than 340 Bytes the decoding time tends to be longer than the time for encoding. The time for encoding and decoding a message of 300 Bytes is approximately 2.25 seconds for each using order three and 1.8 seconds using order two, respectively. We refer the interested reader to the study in [24], where an extended performance evaluation of lcc is given including the Nokia 6600 phone.

V. Conclusion

In this paper we have first demonstrated the potential usability of an heuristic data model for PPM using a smart hash function. We then have have described a static compression scheme with low memory requirements for short text sequences. The particular advantage of the scheme is that compression already starts for files longer than 50 Bytes. This is achieved by a preloaded model that uses a hash-table to store a parity and a count Byte for each string to be modeled. The scheme is scalable in complexity as it allows for the use of an arbitrary model order without source code refinements. In addition, it is scalable in required memory size and thus can be applied to many embedded systems. The order two models with required memory sizes ranging from 32 to 256 kBytes achieve compression ranging from 4 to 3.5 Bits/Byte. The 256 kByte order three model gives approximatively 3.2 Bits/Byte.

A key-component of the scheme is the hash-function that should equally fill the table and only retain the statistical important information. The complete low-complexity method including the arithmetic coder can be implemented with just a few pages of source code. Applications of the scheme may be in sensor networks for exchange of signaling data - i.e., for routing, for the lossless compression of specific, small, and thus very valuable sequences of sensor data, space communications, or in the cellular world. We finally have developed a cell-phone application with the here detailed compression scheme that is freely available at `http://kom.aau.dk/project/mobilephone` and for which we have recorded more than 20000 downloads within 3 months (excluding downloads from other pages). With such an application cell phone users may cut their costs for short message services in half. Furthermore, the application results into battery savings including the sending and the receiving phone.

Acknowledgment

We would like to thank Morten V. Pedersen and Thorsten Schneider from Aalborg University for the help on the cell phone implementations and measurements. We

are also grateful for Nokia's support providing us the phones.

REFERENCES

[1] A. Moffat, "Implementing the ppm data compression scheme," in *IEEE Transactions on Communications, Transactions Letters*, vol. 38, no. 11, Nov 1990, pp. 12–23.
[2] J. Cleary, W. Teahan, and I. Witten, "Unbounded length contexts for ppm," in *Proceedings of the IEEE Data Compression Conference (DCC'95)*, March 1995, pp. 52–61.
[3] M. Drinic, D. Kirovski, and M. Potkonjak, "Ppm model cleaning," in *Proceedings of the IEEE Data Compression Conference (DCC'95)*, March 1995, pp. 52–61.
[4] P. Skibinski and S. Grabowski, "Variable-length contexts for ppm," in *Proceedings of the IEEE Data compression conference (DCC'04)*, March 2004, pp. 409–418.
[5] D. Lelewer and D. Hirschberg, "Streamlining context models for data compression," in *Proceedings of the IEEE Data compression conference (DCC'91)*, Snowbird UT, 1991, pp. 313–322.
[6] E. Hatton, "Samc–efficient semi-adaptive data compression," in *Proceedings of the IBM 1995 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada, Nov. 1995, p. 29.
[7] D. Shkarin, "Ppm: one step to practicality," in *Proceedings of the IEEE Data compression conference (DCC'02)*, April 2002, pp. 202–211.
[8] N. Kimura and S.Latifi, "A survey on data compression in wireless sensor networks," in *Proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC'05)*, April 2005, pp. 8–13.
[9] H. Gupta, V. Navda, S. Das, and V.Chowdhary, "Efficient gathering of correlated data in sensor networks," in *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing (MobiHoc '05)*, May 2005, pp. 402–413.
[10] N. Gehrig and P. Dragotti, "Distributed sampling and compression of scenes with finite rate of innovation in camera sensor networks," in *Proceedings of the IEEE Data Compression Conference (DCC'06)*, March 2006, pp. 83–92.
[11] J. Lansky and M. Zemlicka, "Compression of small text files using syllables," in *Proceedings of the IEEE Data Compression Conference (DCC'06)*, March 2006, p. 458.
[12] G. Korodi, J. Rissanen, and I. Tabus, "Lossless data compression using optimal tree machines," in *Proceedings of the IEEE Data Compression Conference (DCC'05)*, March 2005, pp. 348–357.
[13] S. Rein, C. Gühmann, and F. Fitzek, "Low complexity compression of short messages," in *Proceedings of the IEEE Data Compression Conference (DCC'06)*, March 2006, pp. 123–132.
[14] S. Rein and C. Gühmann, "A free library for context modeling with hash-functions– part i," Wavelet Application Group, Tech. Rep., May 2005. [Online]. Available: http://www.mdt.tu-berlin.de
[15] T. Bell, J. Cleary, and I. Witten, *Text Compression*. Prentice Hall, 1990.
[16] W. Teahan and J. Cleary, "Modeling of english text," in *Proceedings of the IEEE Data Compression Conference (DCC'97)*, March 1997, pp. 12–21.
[17] J. Storer, *Data Structures and Algorithms*. Birkhäuser/Springer, 2002.
[18] S. Rein and C. Gühmann, "Arithmetic coding – a short tutorial and free source code," Wavelet Application Group, Tech. Rep., April 2005. [Online]. Available: http://www.mdt.tu-berlin.de
[19] ——, "A free library for context modeling with hash-functions– part ii: Extensions," Wavelet Application Group, Tech. Rep., August 2005. [Online]. Available: http://www.mdt.tu-berlin.de
[20] R. Arnold and T.Bell, "A corpus for the evaluation of lossless compression algorithms," in *Proceedings of the IEEE Data compression conference (DCC'97)*, 1997. [Online]. Available: http://corpus.canterbury.ac.nz
[21] M. P. M. Hart, G. Newby, "Project gutenberg literary archive foundation," 809 North 1500 West, Salt Lake City, UT 84116. [Online]. Available: http://www.gutenberg.org
[22] T. Bell, I. Witten, and J. Cleary, "Modelling for text compression," in *ACM Computing Surveys (CSUR)*, vol. 21, no. 4, Dec. 1989, pp. 557–591.
[23] B. Jenkin, "Algorithm alley," in *Dr. Dobb's Journal*, Sept. 1997. [Online]. Available: http://burtleburtle.net/bob
[24] F. Fitzek, S. Rein, M. P. G. Perucci, T. Schneider, and C. Gühmann, "Low complex and power efficient text compressor for cellular and sensor networks," in *15th IST Mobile and Wireless Communication Summit*, June 2006.

Stephan Rein is a research and teaching assistant with the Department of Electronic Measurement and Technical Diagnostic Technology, Technical University Berlin. He studied electrical engineering at Technical University of Aachen (RWTH), Germany, Technical University of Berlin, and Arizona State University (ASU), United States. He received the Dipl.-Ing. degree in Communications Engineering at TU Berlin, in 2003. His research interests include data compression, signal processing, and communications.

Stephan is a member of the IEEE Signal Processing Society, the ACM, and a member of the Wavelet Application Group.

Clemens Gühmann is with the Technical University Berlin, where he is professor in the Electrical Engineering and Computer Science Faculty, and head of the department Electronic Measurement and Technical Diagnostic Technology (MDT). His current research interests are in the areas of modern data processing methods (e.g. Wavelets) for automotive systems, data compression, modeling and real time simulation, pattern recognition, technical diagnosis and hybrid vehicle control strategies.

Frank Fitzek is an Associate Professor in the Department of Communication Technology, University of Aalborg, Denmark heading the Future Vision group. He received his diploma (Dipl.-Ing.) degree in electrical engineering from the University of Technology - Rheinisch-Westfälische Technische Hochschule (RWTH) - Aachen, Germany, in 1997 and his Ph.D. (Dr.-Ing.) in Electrical Engineering from the Technical University Berlin, Germany in 2002 for quality of service support in wireless CDMA networks. As a visiting student at the Arizona State University he conducted research in the field of video services over wireless networks. He co-founded the start-up company acticom GmbH in Berlin in 1999. In 2002 he was Adjunct Professor at the University of Ferrara, Italy giving lectures on wireless communications and conducting research on multi-hop networks. In 2005 he won the YRP award for the work on MIMO MDC and in 2005 he received the Young Elite Researcher Award of Denmark. His current research interests are in the areas of 4G wireless communication networks, cross layer protocol design and cooperative networking.