**Protocol-Independent Compression for**

**Resource-Constrained Wireless Networks**

by Travis L. Massey

## Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

**Committee:**

Kristofer S.J. Pister
Research Advisor

Date

* * * * * *

Michel M. Maharbiz
Second Reader

Date

**Abstract**

In wireless devices, reducing the time the radio is on results in lower power consumption. In resource-constrained wireless networks, then, sending the same data in fewer bytes can greatly extend the lifetime of the network. In this paper, we explore the use of protocol-independent packet compression, a technique orthogonal to current explicit compaction techniques. Such a compression algorithm functions as a transparent layer inside a communication stack. Because it makes no assumption on the specific protocols used, it is generic enough to be used on multiple technologies.

Compression is performed by identifying patterns in recently sent packets and replacing those patterns with bit flags, effectively reducing the size of the packet to be sent. We discuss the trade-offs between computation, memory costs and power savings in such an algorithm. We then present the results of compressing actual packet traces collected from several commercial networks using this algorithm. Results indicate compression ratios between 40% and 80%, which yield savings of 30-70% in the average power consumption of a typical time-synchronized network.

# Contents

# 1 Introduction

The application space for resource constrained wireless networks covers fields as diverse as building automation, industrial monitoring, body sensor networks, home electronics, computer interfacing, and energy applications. Because of the unprecedented conditions under which they can operate, these networks have received significant attention in the last decade. Standardization bodies such as the IETF and the IEEE are continually developing and standardizing new protocol stacks to address these diverse wireless sensor network application requirements.

Removing the need for wires to network smart objects also means that most of these devices become battery powered, which has led to standards such as IEEE802.15.4 [1] having been developed for low-power radios. Compliant radios, such as the ubiquitous CC2420 [2], consume about 65mW when on. If left on all the time, a wireless device equipped with such a radio would have a lifetime of only 5 days on a set of two AA batteries (typically containing around 2000mAh). To further improve lifetime, smart medium access control (MAC) protocols can tune the duty cycle of the radio to below 1%, yielding lifetimes of multiple years.

IEEE802.15.4 is the de-facto standard for low-power radios, with compliant radios currently equipping nearly all resource-constrained wireless networks. This standard defines packets up to a maximum of 128 bytes in length. Fig. 1 shows the energy consumed over time when a packet is sent (collected from an eZ430-RF2500 wireless sensor node), assuming the node's radio is initially off. At time $t = 0$, the radio turns on its voltage regulator, waits for its crystal oscillator to stabilize, and for the radio oscillator to settle (tune) to the proper frequency. This initial startup process lasts for about $800\mu s$, a phase during which the radio consumes $5.2\mu C$. After this, transmission can start. Fig. 1 was measured when the radio transmitted a full 128-byte-long packet. To transmit a full packet, the radio consumes $5.2\mu C + 27.0\mu C + 63.2\mu C = 95.4\mu C$. If a node were able to compress 70% of the data it has to transmit (the white portion in Fig. 1), it would only consume $5.2\mu C + 27.0\mu C = 32.2\mu C$, a 66% decrease in the energy consumed (which can translate into longer lifetime, or smaller batteries). This is the idea behind packet compression.

We call the *compression ratio* the fraction of bytes that has been compressed out. This is formalized in (1), where $L$ represents the length, in bytes, of a packet. Note that $\mathcal{C} < 1$. $\mathcal{C} = 0$

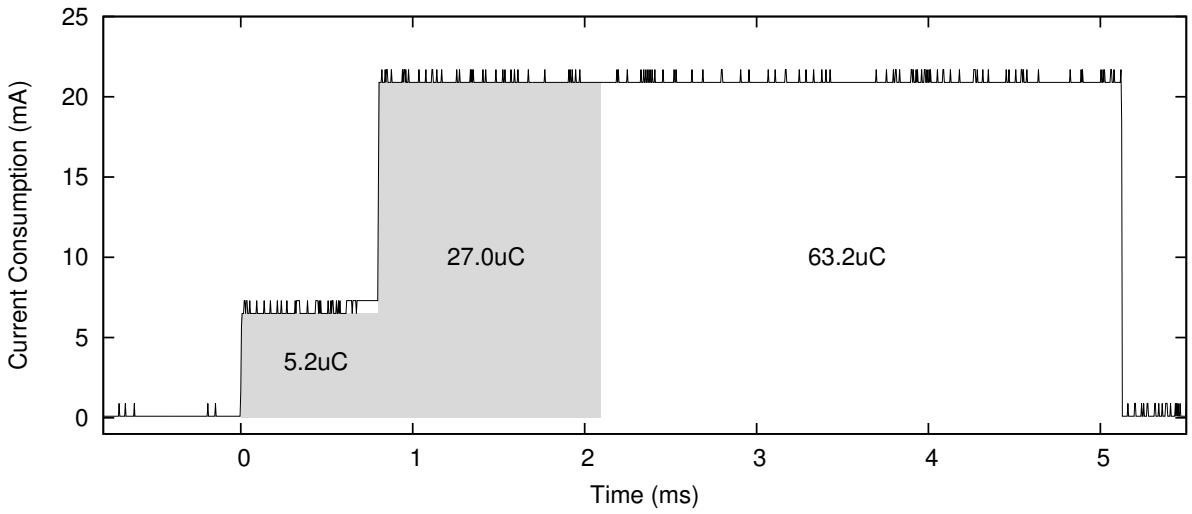


Figure 1: A 70% compression ratio translates into 66% less energy consumed.

means that no compression was possible; $\mathcal{C} \to 1$ approaches the case where all bytes have been compressed out.

$$\mathcal{C} = \frac{L_{original} - L_{compressed}}{L_{original}} \tag{1}$$

This paper explores *protocol-independent packet compression*, which relies on two principles:

- the technique used is *unaware and independent of the structure of the packet being compressed*. Specifically, it does not make any assumption on the contents of the headers. This is a key difference from, for example, 6LoWPAN header compaction, which is described in Section 2.2;
- while two nodes need to share some state to be able to compress/decompress, there is *never any explicit state exchange*. Instead, each node builds an identical state locally. No explicit signaling traffic is needed, further reducing the energy expenditure.

The technique presented in this paper is founded upon *pattern recognition*, in which multiple-byte patterns are replaced by single-bit flags. All nodes, based on the buffer of previously transmitted/received packets, use a simple algorithm to identify sequences of consecutive bytes recurring

in the packets, which are called patterns. Patterns are stored in a pattern list. A transmitting node, when asked to transmit a packet, removes the previously identified patterns from that packet, and replaces them with a set of one-bit flags indicating which patterns were removed. This set of flags, call the *tag*, is transmitted at the beginning of the packet. A receiving node decompresses the packet by inserting the patterns indicated in the tag.

The severe resource constraints of wireless sensor nodes present significant challenges in the implementation of a pattern-based compression algorithm. Typical wireless sensor nodes have very little memory. The ubiquitous TelosB mote, for example, features only 10kB of RAM and 48kB of ROM – very little space for the state maintained the algorithm. Moreover, because we do not want to explicitly transmit the shared state (i.e. the pattern list), this list has to be built identically on both ends of a wireless link. This differs from the standard approach used in dictionary-based algorithms used in desktop file compression tools. Finally, depending on the type of traffic on the network, tuning the parameters differently may result in widely different performance.

The techniques to overcome these challenges are described in Section 3, which also presents a practical and simple protocol-independent packet compression method. Prior to that, Section 2 details related work in the field, with a particular focus on IETF 6LoWPAN header compaction. Section 4 presents our implementation of the compression scheme, as well as experimental results on two different packet traces collected from commercial wireless sensor networks. Section 5 discusses the applicability of protocol-independent compression techniques to real-world networks. Section 6 concludes this paper.

# 2 Existing Compression Techniques

## 2.1 Protocol-Independent Compression Techniques

Compression techniques which are unaware of the content of the data typically use dictionaries. When processing a stream of bytes, the compression algorithm recognizes often-repeated patterns, stores those patterns into a dictionary, and indicates in the stream of data where those patterns are. Patterns are typically identified by their key, i.e. the rank of the pattern in the dictionary. The idea

is that the actual pattern is only written once in the dictionary, and replaced multiple times by small keys. This idea is used in popular desktop file compression tools such as zip, gzip and winzip. The output file of these algorithms is made up of two parts: the dictionary (typically written at the beginning of the file) and the compressed data.

In this work, because an individual packet is so small, it makes little sense to prepend the packet with its dictionary. Instead, for each link, each node has a local copy of the dictionary (which we call pattern list), and only the compressed data is sent over the air. Moreover, instead of indicating the keys of the patterns inline with the data, a tag consisting of a set of binary flags indicating whether or not the corresponding pattern has been compressed out of the packet is prepended to the data.

## 2.2 Explicit Compaction Techniques

We differentiate protocol-independent compression from explicit compaction. In the former, a stream of bytes is compressed by replacing patterns by keys. In the latter, the format of the data is known *a priori*, and redundant data is compacted out by analyzing the semantics of the data. This technique is particularly applicable for communication protocols, as these comply with a well-defined structure.

Explicit data compaction on constrained links is not a new idea. An early proposal aimed at increasing the useful throughput on low data rate serial modem connections [3]. Van Jacobson (**VJ**) header compaction removes the information that remains constant over the duration of the connection, including addresses, ports, and offsets from TCP/IPv4 headers. The header is further pared down as information that can be determined from other fields or other layers is removed; moreover, the checksum is no longer necessary as the remainder of the TCP header has been compressed out. For the fields that remain, only the difference between two successive headers is sent. If the degree of change is restricted to fewer bytes than a complete field, then the bytes that remain unchanged (or change by one) can be compressed out. As a result, the 40-byte TCP/IPv4 header is compacted to five bytes.

Robust Header Compression (**ROHC**) [4] demonstrates improved robustness over links with

high error rates, specifically those in wireless networks, and generalizes the compaction algorithm to several common protocols. ROHC is more dynamic than VJ header compression in that it gradually progresses the state of the compactor based upon its "confidence" that it can predict each successive header. It compacts each field independently of the others using LSB encoding, transmitting the $k$ least significant bits and calculating the expanded value as a function of $k$ and an offset $v_{ref}$. Errors that occur during transmission are caught by the CRC or MIC at the MAC layer. A major limitation of ROHC is the quantity of memory required to support many simultaneous contexts, and the memory bandwidth required to read and write this information. In the worst case, the memory bandwidth required can be close to nine times the link rate.

[5] focuses explicitly on wireless networks in which parallel paths exist, designing the compaction scheme to exploit the multiple channels. The fundamental compaction mechanism is delta coding (as in VJ header compaction), but the header of the packet in each channel also contains information for the bases of the neighboring channels in additional information containers (AICs). If a neighboring channel loses a packet and the compaction and de-compaction engines lose their synchronization, they can recover by requesting base information from a neighboring channel.

While three AICs per channel are expected to improve reliability significantly, this is ultimately a failure as a header compaction scheme because it appends unnecessary bytes to the packet. If the base information required is substantial, then the size of the AIC quickly becomes significant, which degrades the efficiency of this approach as a compaction scheme. Advantages of this approach are high bandwidth efficiency, low memory consumption, low complexity, and robustness.

The Internet Engineering Task Force (IETF), through its working group IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) has proposed a header compaction scheme for transmitting IPv6 packets in IEEE802.15.4 networks. While early versions of the technique are referenced in [6], it has been turned into an IETF standard [7][1]. Through deep understanding of the interaction between IEEE802.15.4, IPv6 and UDP, 6LoWPAN is able to remove fields which are redundant among those headers, thereby reducing the size of the packets being transmitted over the air.

To evaluate the performance of 6LoWPAN, let's assume the simplest case of a point-to-point

[1]Note that RFC4944 [7] will probably be made obsolete by [8], a work-in-progress at the IETF.

link in a wireless multi-hop network network (the best case for 6LoWPAN). Fig. 2 depicts the IEEE802.15.4, IPv6 and UDP headers commonly found in a packet. Alliances such as IP for Smart Objects [2] provide a clear indication that this type of standard packet structure will become ubiquitous in future networks of resource-constrained wireless devices. Fig. 3 shows the same packet, compressed using 6LoWPAN.

6LoWPAN removes a number of fields in the IPv6 and UDP headers because they take well-known values, or because they can be inferred from fields in the IEEE802.15.4 header. In the IPv6 header, the `version` field is always 6 for IPv6, the `traffic class` and `flow label` are never used[3], and the `length` field is always equal to the `length` field of IEEE802.15.4 minus the length of the IPv6 header. All these fields can hence be removed. Because `Next Header` typically point to either UDP or TCP, this 8-bit field can be replace by a 2-bit field as part of the the `HC1` field of the 6LoWPAN header. Finally, RFC2464 [9] defines how 128-bit IPv6 addresses can be recovered from 64-bit MAC addresses, such as the IEEE802.15.4's `Source` and `Destination` fields. This removes the IPv6 `Source Address` and `Destination Address` fields. In the end, only the `Hop Limit` field needs to be present in the 6LoWPAN header. Similarly for UDP, the `Length` can be calculated from the IEEE802.1.54's `Length` field; in the most common case, only a limited number of ports with be used, so 4-bits can be used to describe them, rather than the original 8 bits.

The drawback of a technique such as 6LoWPAN is that it relies entirely on a deep understanding of the protocol stack, in this case UDP over IPv6 over IEEE802.15.4. Explicit compaction can hence not act as a transparent layer inside a protocol stack. Moreover, the compaction only applies to headers and not to the application payload. We will see in Section 5 that explicit compaction and protocol-independent compression are not mutually exclusive and that they can, in fact, be used simultaneously on the same packet.

[2] http://www.ipso-alliance.org

[3] Note that this assumption is changed in [8].

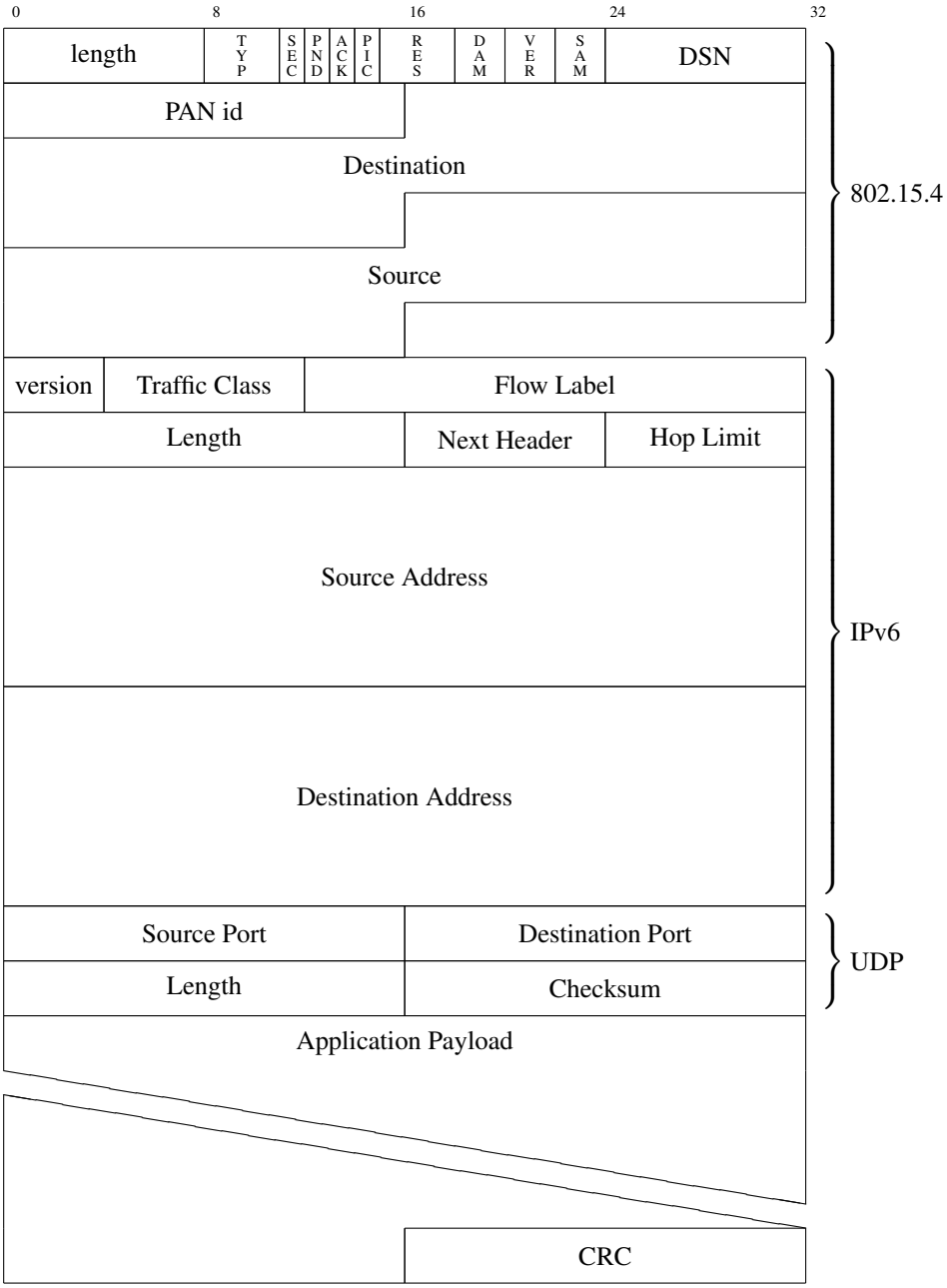


Figure 2: A non-compressed packet with IEEE802.15.4, IPv6 and UDP headers: headers take up 72 of the 128 available bytes, or 56%.

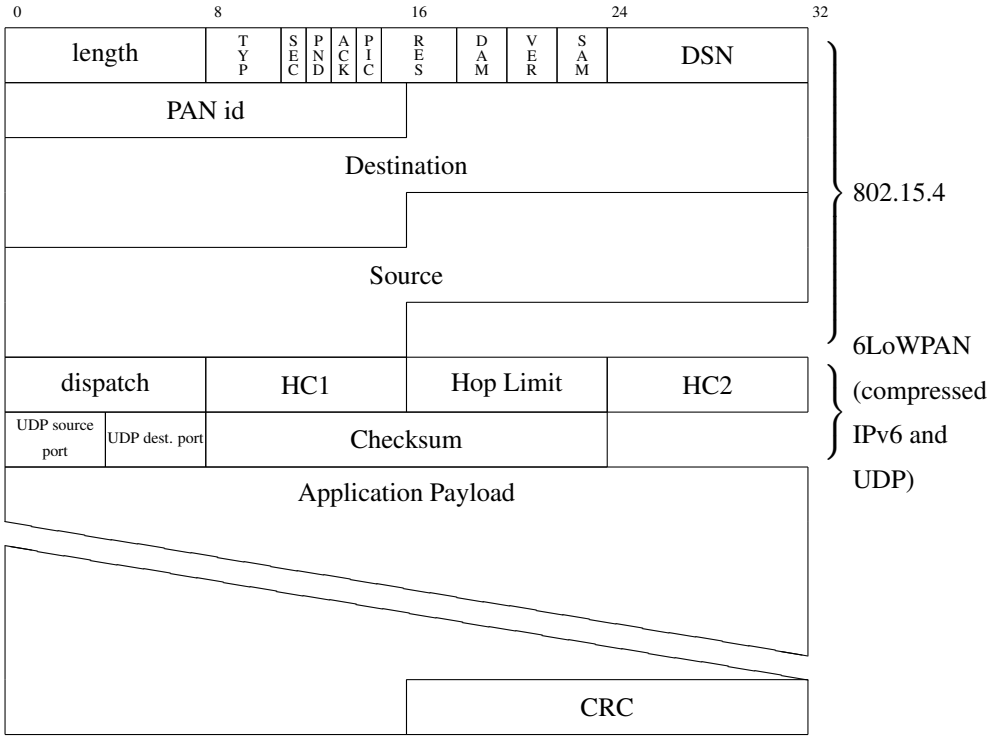


Figure 3: The packet form Fig. 2, now compacted by 6LoWPAN. IPv6 and UDP headers are compressed from 48 bytes to 7 bytes.

# 3 Protocol-Independent Compression for Wireless Networks

## 3.1 Overview

Upon receiving a packet from the upper layer, the compression algorithm matches that packet against the pattern list and identifies which patterns are present in the packet. It then transmits a shorter version of the packet, where patterns have been replaced by flags in a tag attached at the beginning of the packet. Upon reception, the receiving node performs the inverse operation.

In order for this technique to function, a node needs to maintain the following state for each of its neighbors, in each of both directions:

- a pattern list. This contains the patterns that have been learned by the pattern discovery engine (detailed in Section 3.3) used to compress a packet. The pattern list is empty when a node is first switched on.
- a recent packet buffer. For the receiving node, this contains the most recent packets that have been received; for the transmitting node, this contains the latest packets that have been sent. This buffer is used to generate the pattern list. This buffer is also initially empty.

This requires 3 parameters to define the algorithm:

- $\mathcal{B}$, the size of the packet buffer, in entries.
- $\mathcal{P}$, the size of the pattern list, in entries.
- $\mathcal{S}$, the minimum size of a pattern, in bytes.

Note that $\mathcal{B}$, $\mathcal{P}$, and $\mathcal{S}$ are common for all the nodes in the network and do not change with time. For easier reference, their meaning is recapitulated in Table 1. While this section describes the algorithm, Section 4 discusses how to set those parameters based upon experimental results on real-world traces.

| | | |
|---|---|---|
| $\mathcal{C}$ | compression ratio ($0 \leq \mathcal{C} \leq 1$) | output |
| $\mathcal{B}$ | recent packet buffer size | input |
| $\mathcal{P}$ | pattern list size | input |
| $\mathcal{S}$ | minimum pattern size | input |

Table 1: Parameters of the proposed compression scheme.

A pattern is defined as a *contiguous sequence of constant bytes at a specific location* inside the packet. Because patterns are identified only based upon recurring byte sequences, this technique requires no knowledge of the type of headers and can be extended to technologies other than IEEE802.15.4.

The transmitter and receiver must independently maintain the same patterns in their pattern list, without explicitly transmitting state information over the air. The algorithm for building the pattern list is described in Section 3.3.

A packet to be transmitted is sent down from the upper layer and compressed transparently before being passed to the lower layer (see Fig. 4). Compression proceeds according to the following 4 steps:

1. Before the packet is compressed, a cyclic redundancy check (CRC) is computed on its uncompressed form. This will be appended to the compressed packet and used to verify correct decompression.

2. The packet is compared against the patterns that in the pattern list; previously identified patterns found in the packet are removed from the packet, and the corresponding flag in the tag is raised. The tag is composed of $\mathcal{P}$ bits (typically 8, 16 or 24).

3. The tag is prefixed to the compressed packet, and the CRC is appended; the resulting data is sent to the lower layer.

4. The state is updated with the uncompressed packet.

Decompression proceeds according to the following 4 steps:

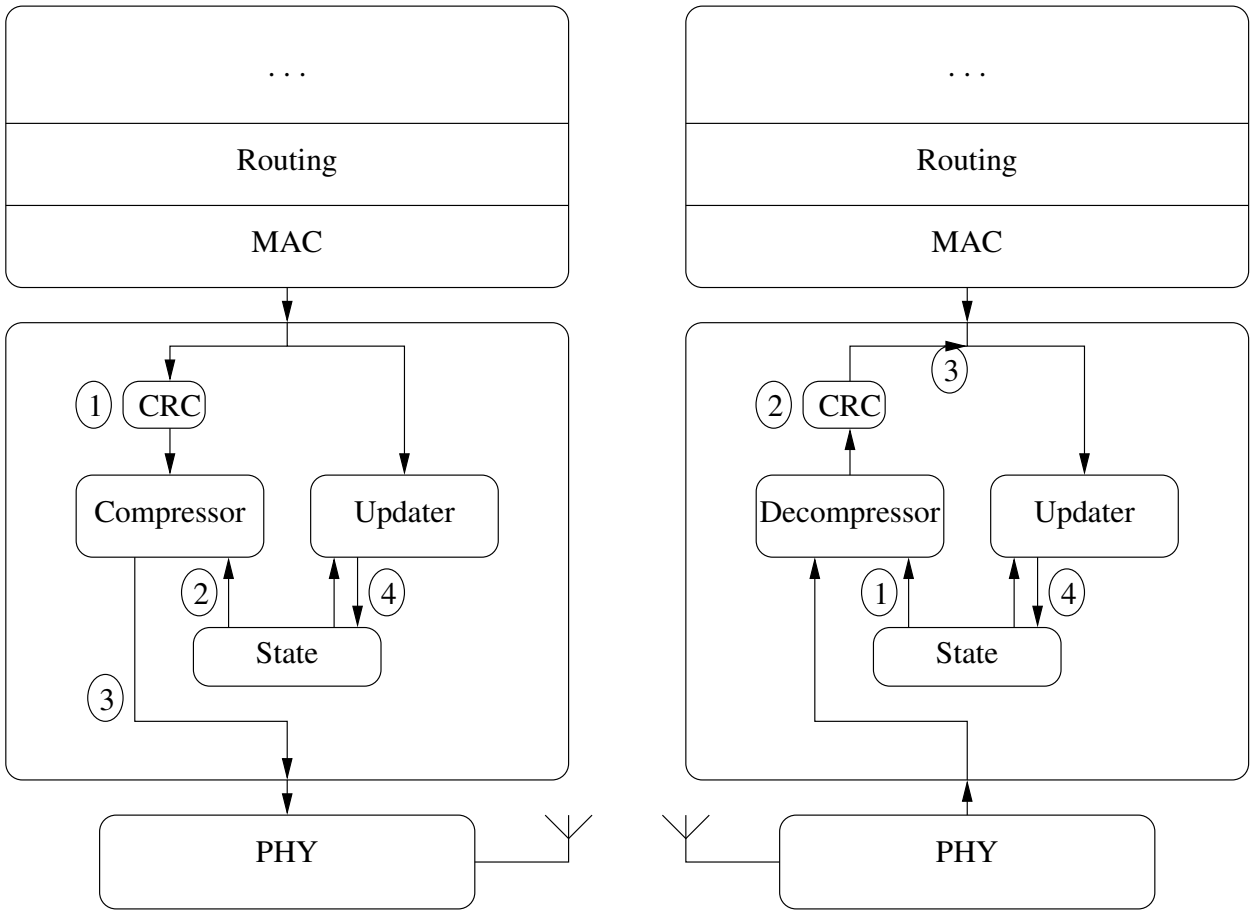


Figure 4: Compression and decompression flow for protocol-independent packet compression.

1. The receiving node reads the tag and inserts the indicated patterns into the compressed packet to form the candidate decompressed packet.

2. The receiver recomputes the CRC on the candidate packet. If this packet passes the check, it is used as the decompressed packet. If CRC fails, this means that the packet was not compressed in the first place[4]. and the initially received packet is used as the decompressed packet.

3. The decompressed packet is passed on to the upper layer.

4. The state is updated with the decompressed packet.

The state update (step 4) is identical for both the transmitter and receiver and is performed using the same uncompressed packet. Given identical a priori states, the following 2 steps will yield the same a posteriori states:

1. The uncompressed packet is analyzed by the *pattern discovery engine*. Possible new patterns are discovered by comparing the current packet with the packets in the recent packet buffer (details in Section 3.3). If a new pattern is found, it will be added to the pattern list.

2. The uncompressed packet is inserted into the recent packet buffer, replacing the oldest packet.

## 3.2 Data Representation

This section describes how the recent packet buffer and pattern list are represented. The packet buffer is composed of $\mathcal{B}$ arrays of bytes, each as long as the maximum packet length (128 bytes in the case of 802.15.4). The most recently seen packets are stored in this buffer, which stores and replaces incoming or outgoing packets according to a first in, first out (FIFO) policy. The packets stored in this recent packet buffer are compared against the current incoming or outgoing packet by

[4]Note that this CRC field is used solely to detect successful decompression; lower layers typically use their own CRC to detect transmission errors

the pattern discovery engine (Section 3.3). While this buffer consumes the most memory, Section 4 shows that $\mathcal{B} = 2$ is sufficient to achieve good performance.

A pattern is a contiguous sequence of bytes, at a given position in a packet. A pattern is hence represented by its starting position, its length, and its contents. Patterns stored in the pattern list follow a least recently used (LRU) replacement policy.

$\mathcal{B}$, $\mathcal{P}$ and $\mathcal{S}$ are tuning knobs to optimize the compression for a given application; how to set them is presented in Section 4.

## 3.3 Pattern Discovery

The goal of the pattern discovery engine is to find patterns while maintaining low complexity. Pattern discovery consists of a byte-wise comparison between the current packet and each of the packets in the recent packet buffer. When the number of consecutive byte matches is greater than or equal to the minimum pattern size $\mathcal{S}$, the corresponding sequence of bytes, is added to the pattern list (duplicates are avoided). If the pattern list is full, adding a new pattern causes the least recently used pattern to be removed.

The minimum pattern size $\mathcal{S}$ ensures that the pattern list is not populated by numerous short patterns at the expense of much longer patterns that would yield greater compression. Note that pattern discovery is only performed on non-compressed packets, i.e. before the compression engine at the transmitter and after the decompression engine at the receiver.

As discussed in Section 5, the comparison operations are efficiently translated by compilers, but can also be implemented on an application-specific integrated circuit (ASIC).

# 4 Implementation and Experimental Results

## 4.1 Data collection

We test the compression algorithm using IEEE802.15.4 packet traces collected from two commercial wireless sensor networks; 62,101 packets were collected from company A and 63,988 packets were collected from company B[5]. Packets were recorded using Integration's IEEE802.15.4 USB Dongle on a single channel in the 2.4 GHz band. Company A data was recorded from a data collection network that uses 16-bit source and destination IEEE802.15.4 addresses. Company B data was recorded from a network performing numerous large file transfers; 64-bit source and destination IEEE802.15.4 addresses were used. Note that the address size and description of the data content is provided for the reader only and does not impact the functioning of the protocol-independent compression scheme.

## 4.2 Implementation Details

The traces collected from company A and company B are run off-line through a C implementation of the compression algorithm in order to predict the energy gain had these networks been running protocol-independent compression. Running compression algorithms on packet traces allows us to precisely quantify the gain of this scheme in real-world commercial applications, something which an in-lab limited deployment would not allow. The algorithm was implemented for a single link per the description of Section 3, sans error checking such as CRC or handling for loss of synchronization as discussed in Section 5.

## 4.3 Compression Ratio $\mathcal{C}$

Compression ratio $\mathcal{C}$, as defined in (1), is plotted versus the parameters $\mathcal{B}$, $\mathcal{S}$, and $\mathcal{P}$. As is seen in Figs. 5-7, the compression ratios were in the range of $50 \pm 5\%$ for data from company A and

[5]Unfortunately, marketing considerations do not allow us to disclose the names of these companies.

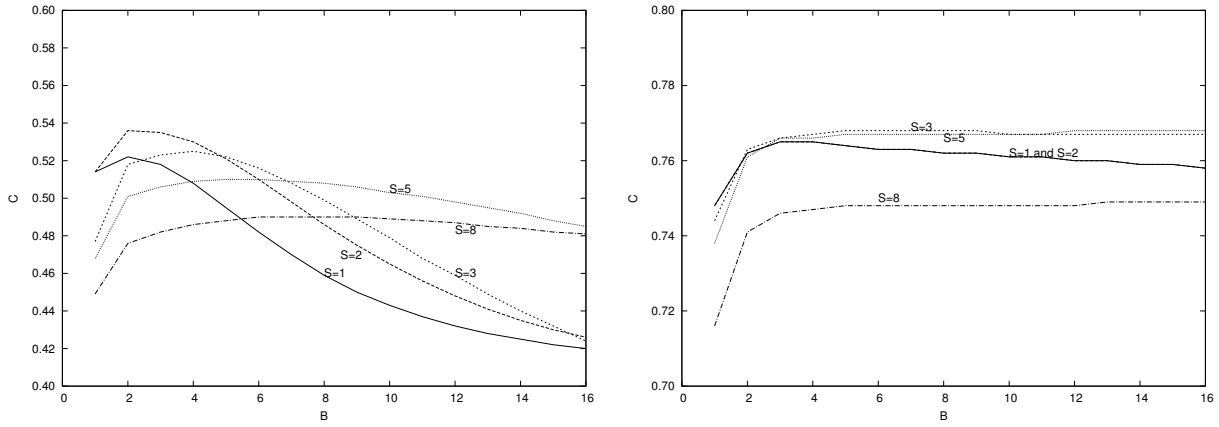


Figure 5: Impact of the recent packet buffer size $\mathcal{B}$ on the compression ratio $\mathcal{C}$ for company A (left) and company B (right), for $\mathcal{P} = 8$. Note that the y-axis scale is different between graphs.

$75 \pm 3\%$ for data from company B. These results are consistent with the observation that the packets from company B have a relatively large static header and are generally quite repetitive given the nature of a file distribution, whereas the company A data is far less regular and, as a result, benefits less from compression.

## 4.4 Impact of the recent packet buffer size $\mathcal{B}$

Increasing the recent packet buffer size $\mathcal{B}$ impacts the compression ratio $\mathcal{C}$. When $\mathcal{B}$ is large, the pattern discovery engine can compare the current packet to more recent packets, and can hence find more patterns.

The regions of positive slope in Fig. 5 intuitively correspond to more points of comparison from which to discover patterns useful for compression. The decrease in $\mathcal{C}$ for large $\mathcal{B}$ comes as a result of useful patterns being removed from the pattern list in favor of new patterns identified, according to the LRU replacement policy. These patterns (identified in older packets) are often less beneficial or relevant for future compression than those from more recent packets.

$\mathcal{C}$ peaks for lesser values of $\mathcal{B}$ and for shorter minimum packet lengths $\mathcal{S}$ because patterns are more quickly cycled through the pattern list due to the increase in number of patterns being identified. As can be seen from Fig. 5 and Fig. 6, $\mathcal{B}$ and $\mathcal{S}$ roughly track each other; a small $\mathcal{B}$ warrants a small $\mathcal{S}$, and vice versa. As $\mathcal{B}$ grows, $\mathcal{S}$ grows as well. As $\mathcal{S}$ increases to large values,

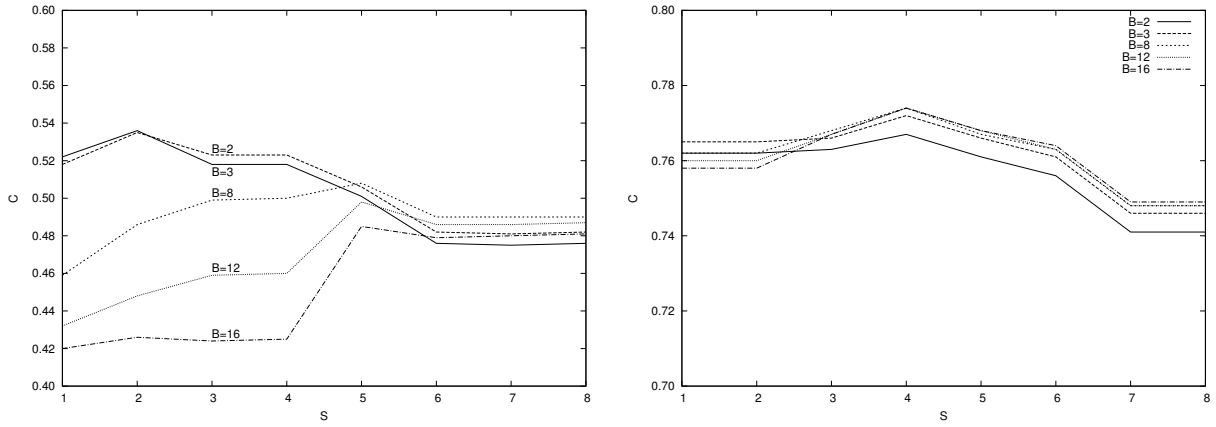


Figure 6: Impact of the minimum pattern size $\mathcal{S}$ on the compression ratio $\mathcal{C}$ for company A (left) and company B (right), for $\mathcal{P} = 8$. Note that the y-axis scale is different between graphs.

eventually the compression ratio with respect to $\mathcal{B}$ flattens out because few patterns of that length exist, but those that do exist are likely to be often used, representing a large static portion of the packet.

## 4.5 Impact of the minimum pattern size $\mathcal{S}$

Additional insight can be gained into the relationship of $\mathcal{B}$ to $\mathcal{S}$ by carefully examining Fig. 6. The data from company A indicates that, for small values of $\mathcal{B}$, the change in $\mathcal{C}$ with $\mathcal{S}$ is relatively smooth, while as $\mathcal{B}$ becomes large the transition between the compression ratio for small $\mathcal{S}$ and large $\mathcal{S}$ becomes more abrupt. This abrupt transition occurs because above a minimum pattern size useful patterns are no longer being ejected from the pattern list; all of the most meaningful patterns are retained. Data from company B shows little variation with $\mathcal{B}$ over a range of $\mathcal{S}$ because the patterns in that data set tend to not change quickly, and the small change in $\mathcal{C}$ over the entire range of $\mathcal{S}$ suggests that the patterns in the company B data are typically long.

Another interesting feature to note from Fig. 6 is the convergence of the curves in the company A data for various $\mathcal{B}$ converging to a common $\mathcal{C}$ as $\mathcal{S}$ exceeds 5 bytes. This behavior indicates that the only patterns being compressed for large $\mathcal{S}$ are those that rarely change, so the rapidly changing patterns must generally be less than 5 bytes long, most likely corresponding to a specific four-byte field within the packet.

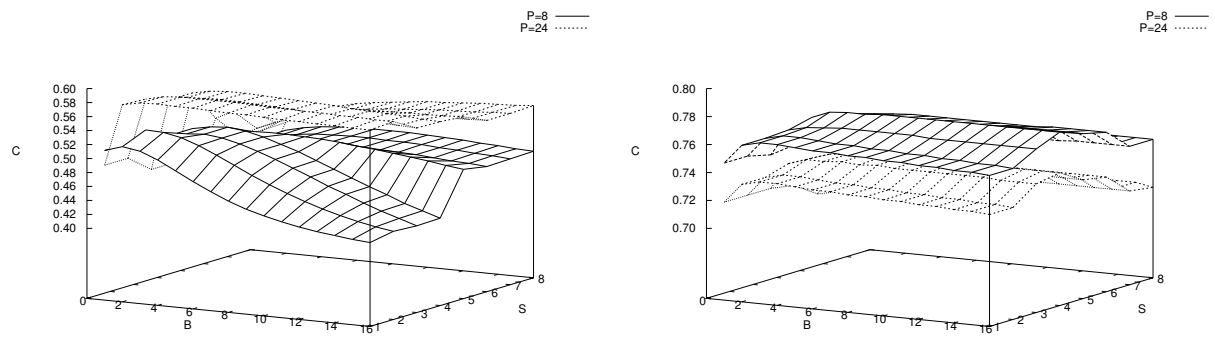


Figure 7: Impact of the pattern list size $\mathcal{P}$ on the compression ratio $\mathcal{C}$ for company A (left) and company B (right). Note that the y-axis scale is different between graphs.

## 4.6 Impact of the pattern list size $\mathcal{P}$

The last trend to discuss is the effect of varying the pattern list size $\mathcal{P}$. $\mathcal{P}$ was varied coarsely from 8 to 24 in increments of 8, corresponding at each step to one additional tag byte required at the beginning of the compressed packet. At first glance, Fig. 7 seems to yield contradictory results for the two data sets; company A data benefits from a larger pattern list, whereas the compression ratio for the company B data is greater for smaller $\mathcal{P}$. Upon considering what we know about the data, however, this makes perfect sense; company A has a much greater degree of randomness than the company B data, so it can benefit from the additional pattern allowance. Data from company B , on the other hand, changes infrequently and therefore requires fewer than eight patterns to effectively be compressed. Additional patterns beyond those required result in unnecessary tag bytes being added to the beginning of the packet. Indeed, the 3% decrease in compression ratio for the company B data corresponds to two additional bytes being added to a sixty to seventy byte packet. Finally, we notice also that the declining $\mathcal{C}$ for increasing $\mathcal{B}$ for company A data is no longer as pronounced for larger $\mathcal{P}$. As the number of patterns that may be stored is increased, the useful patterns that were formerly being ejected due to the large $\mathcal{B}$ are no longer being removed from the pattern list and compression does not decrease with increasing $\mathcal{B}$.

## 4.7 Parameter Tuning Conclusions

With this final piece of information, the intuitive result that the compression ratio can be improved by providing the algorithm with more memory is shown to be true, up to the point that the cost of adding additional tag bytes outweighs the benefit of additional patterns. However, the parameters must be increased with care such that the aforementioned dangers of unbalanced parameters are avoided. That being said, it is also clear that, at some point, the compression ratio saturates and there is little compression benefit to further tuning the parameters. From a memory and computation standpoint, it is advantageous to remain at the edge of this saturated region, or possibly even at a point with lower compression ratio. As is evident from Figs. 5-7, the compression ratio does not change drastically over the full range of $\mathcal{B}$, $\mathcal{S}$, and $\mathcal{P}$; leaving each of the parameters at or near their minimum values is reasonable, especially in situations where available memory is limited. The peak compression configurations consume approximately 400 and 1200 bytes for Companies A and B respectively. Efficient protocol-independent compression can hence be implemented in resource-constrained wireless devices. Additionally, a figure of merit may be established by weighing the relative importance of power savings versus memory consumption.

# 5 Discussion

## 5.1 Insertion into the Protocol Stack

The protocol-independent compression scheme presented in this paper is a transparent layer that sits between any two protocol layers in a protocol stack; it compresses the data provided by the layer sitting above it. One possibility is to place this compression layer between the routing and MAC layers. In this case, the MAC headers are not compressed, which may be desirable if destination address filtering is done in the radio chip and the MAC destination address needs to be sent unaltered over the air.

The compression ratio can be further increased by placing the compression layer between MAC and physical layers. This optionally enables the compression to be performed on a dedicated chip,

sitting between the microcontroller and the radio on the sensor node. Thanks to its simplicity, the algorithm can be translated into digital logic using a hardware description language such as Verilog and fabricated on an application-specific integrated circuit (ASIC). Implementing the compression algorithm on an ASIC reduces the cost of computation, both in time and energy, and compression becomes transparent for the programmer.

## 5.2 Efficiency and Flexibility

For only very minimal resources, data can be compressed beyond 50%. In a typical wireless network, this can potentially double the lifetime of the network. Given the of use of a CRC field, protocol-independent compression allows for compressed and uncompressed packets to coexist on a network. This is a valuable level of flexibility that motivates the inclusion of the CRC.

Because the algorithm is independent from other communication protocols (to the extent that it can be implemented on a separate chip), this compression scheme can be plugged into existing wireless networking stacks. Protocol-independent compression can even be used in conjunction with explicit header compaction techniques. Assuming a 6LoWPAN-enabled IEEE802.15.4 network in which the header can not be compressed, such a layer can sit on top of the 6LoWPAN adaptation layer, compressing only the application data.

## 5.3 Potential Improvements to the Algorithm

A more elaborate version of the compression algorithm involves weighing the entries in the pattern list according to criteria different from least recently used. Possible metrics are the length of the pattern, the number of times a specific pattern has been used, whether one pattern is a subset of another, or any combination thereof. While such more elaborate metrics can increase the compression ratio in specific cases, this efficiency increase should be weighed against the additional complexity.

A challenge for using this compression scheme is to tune $\mathcal{B}$, $\mathcal{P}$ and $\mathcal{S}$. While good performance can be achieved with an informed guess, another 20% performance gain can be achieved by fine-

tuning the parameters.

A challenging situation is when two nodes lose synchronization in updating their pattern lists, a difficulty that can not be avoided in the presence of lossy links. Such a situation can be detected by reserving 2 bits from the tag to serve as a counter to indicate the state of the pattern table. If a node receives a packet with this counter set to a different value than its own, it may decide to flush the pattern list. While this causes a few packets not to be compressed (less than $\mathcal{B}$), this does not cause any packet to be lost.

The payloads of packets are often encrypted, which causes the payload to appear randomized. In this case, only the header of the packet is compressed. A possible way of improving the algorithm is to keep track of the location of the patterns; if they are all at the beginning of the packet, the recent packet buffer can be shrunk to contain only the header portion of recent packets.

# 6 Conclusion

We have demonstrated a compression algorithm for resource constrained wireless networks that is independent of network protocols and does not require explicit signaling messages to maintain shared state between communicating nodes. Furthermore, due to the use of a CRC, compressed and uncompressed packets may coexist in a network. Compression ratios above 50% and 75% were achieved on two traces gathered from commercial IEEE802.15.4 networks. This translates into energy savings of the same order in a time-synchronized network given the minimal energy cost of computation [6], potentially more than doubling the lifetime of the network. Protocol-independent compression can be used at any layer in a protocol stack, offering the flexibility to compress certain headers or not. It can hence be used in conjunction with existing compaction techniques such as 6LoWPAN. This paper also presented guidelines and insights into fine-tuning the parameters of the compression algorithm to the needs of a specific network. This work has been published externally in [10].

# References


[1] *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE Std., Rev. 2006, 8 September 2006.

[2] CC2420, *2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. B)*, Texas Instruments, Inc., 20 March 2007, data Sheet SWRS041B [available online].

[3] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, IETF Std. RFC1144, February 1990, RFC1144.

[4] D. Taylor, A. Herkersdorf, A. Doring, and G. Dittmann, "Robust Header Compression (ROHC) in Next-Generation Network Processors," *IEEE/ACM Transactions on Networking*, vol. 13, no. 4, pp. 755–768, August 2005.

[5] F. H. P. Fitzek, T. K. Madsen, P. Popovski, R. Prasad, and M. Katz, "Cooperative IP Header Compression for Parallel Channels in Wireless Meshed Networks," in *IEEE International Conference on Communications (ICC)*, vol. 2, May 2005, pp. 1331–1335.

[6] J. W. Hui, "An Extended Internet Architecture for Low-Power Wireless Networks - Design and Implementation," Ph.D. dissertation, EECS Department, University of California, Berkeley, September 2008.

[7] G. Montenegro, N. Kushalnagar, J. W. Hui, and C. D. E., *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, 6LoWPAN Std., September 2007, RFC4944.

[8] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams in 6LoWPAN Networks," IETF 6LoWPAN, Tech. Rep., 5 October 2009, draft-ietf-6lowpan-hc-06 (work in progress).

[9] M. Crawford, *Transmission of IPv6 Packets over Ethernet Networks*, IETF RFC2464, December 1998, RFC2464.

[10] T. L. Massey, A. Mehta, T. Watteyne, K. S. J. Pister, "Protocol-Agnostic Compression for Resource-Constrained Wireless Networks," in *IEEE Globecom 2010*, December 2010.