Packet Compression for Time-Synchronized Wireless Networks

Travis L. Massey, Ankur Mehta, Thomas Watteyne, Kristofer S.J. Pister
BSAC, University of California, Berkeley, USA
{tlmassey,mehtank,watteyne,pister}@eecs.berkeley.edu

Abstract—Reducing the number of transmitted bytes in a wireless sensor network reduces the time the radio is on, resulting in a significant increase in battery lifetime. Toward this end we have developed a compression technique that is independent of the protocols used in the network, acts as a transparent layer, and consumes minimal computing resources. Patterns in recent packets are identified and replaced in the transmitted packet by bit flags. This algorithm was tested on packet traces collected from commercial wireless sensor networks for 40-80% compression, yielding comparable energy savings in a time-synchronized network.

I. INTRODUCTION

Removing the need for wires to network smart objects also means that most of these devices become battery powered. These devices use low-power radios such as the popular CC2420 [1], which consumes 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. To 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 [2] is the de-facto standard for most low-power radios in wireless sensor networks. Fig. 1 shows the current consumed versus time for a packet transmission. We call the compression ratio, C the fraction of bytes that have been eliminated by compression divided by the original number of bytes.

This work explores protocol-independent compression, which relies on two principles:

• the technique used is independent of the structure of the packet being compressed. No assumption is made regarding the contents of the headers.
• while two nodes maintain common state to compress/decompress properly, there is never any explicit state exchange. Each node builds identical state locally.

The technique presented in this paper is founded upon pattern recognition, in which multi-byte patterns are replaced by single-bit flags. All nodes, based on a buffer of previously transmitted/received packets, use a simple algorithm to identify sequences of consecutive bytes recurring in the packets. These sequences are called patterns and 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; for example, the TelosB mote features 10kB of RAM and 48kB of ROM – very little space for the state maintained by the algorithm. Moreover, because we do not want to explicitly transmit the shared state (i.e. the pattern list), this list must be built identically on both ends of a wireless link. This differs from the standard approach used in dictionary-based algorithms used for desktop file compression tools.

II. EXISTING COMPRESSION TECHNIQUES

A. Data-Independent Compression Techniques

Compression techniques that 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. The actual pattern is only written once in the dictionary, and replaced multiple times by small keys. 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 the 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.

B. Explicit Compaction Techniques

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 [3], it has been developed into an IETF standard [4]. 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.

6LoWPAN removes 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. The drawback of a technique such as 6LoWPAN is that it relies upon full knowledge of the protocol stack and hence cannot act as a transparent layer. Moreover, the compaction only applies to routing and transport headers, leaving the MAC header, application payload, and 6LoWPAN-specific header fields untouched. Fortunately, explicit compaction and agnostic compression are not mutually exclusive and can be used simultaneously.

III. PROTOCOL-INDEPENDENT COMPRESSION

A. Overview

Upon receiving a packet from the upper layer, the compression algorithm compares 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 compression technique to be applied a node must maintain the following state for each of its neighbors, both for transmission and reception:

- a pattern list. This contains the identified patterns that are used to compress a packet. A pattern is a contiguous sequence of bytes at a given position in a packet, and is represented by its starting position, length, and contents. Patterns stored in the pattern list follow a least recently used replacement policy.
- a recent packet buffer. For the receiving (transmitting) node, this contains the most recent packets that have been received (transmitted). The recent packet buffer is composed of $B$ arrays of bytes, each as long as the maximum packet length (128 bytes in the case of IEEE802.15.4). The packets are stored in this buffer with a first in, first out (FIFO) replacement policy.

Algorithm tuning parameters $B$ (number of entries in the recent packet buffer) and $S$ (minimum size of a pattern, in bytes) are common for all nodes in the network and do not change with time. Section IV discusses the effect of these parameters on compression.

A packet to be transmitted is sent down from the upper layer, and is compressed transparently before being passed to the lower layer (see Fig. 2). Compression proceeds according to the following 4 steps:

1) A CRC is computed on the uncompressed packet. This will be appended to the compressed packet and used to verify correct decompression.
2) The patterns in the packet are compared against the pattern list. Known patterns are removed from the packet, and the corresponding flag in the tag byte is raised.
3) The tag byte is prepended to the compressed packet, the CRC is appended, and the resulting packet is sent to the lower layer.
4) The pattern list is updated with the uncompressed packet.

Decompression proceeds according to the following 4 steps:

1) The receiving node reads the tag and inserts the corresponding 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, the packet may not have been not compressed in the first place and the initially received packet is used as-is.
3) The decompressed packet is passed on to the upper layer.
4) The pattern list is updated with the decompressed packet.

B. Pattern Discovery

The transmitter and receiver must maintain the same patterns in their pattern list without explicitly transmitting signaling information and 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 $S$, the corresponding sequence of bytes is added to the pattern list (duplicates are avoided). The minimum pattern size $S$ ensures that the pattern list is not populated by numerous short patterns at the expense of longer patterns which yield greater compression. Note that pattern discovery is only performed on uncompressed packets. After pattern discovery, the uncompressed packet is inserted into the recent packet buffer.

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

A. Implementation Details

We tested the compression algorithm using 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\(^3\). Company A data was recorded from a data collection network which 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 IEEE802.15.4 addresses were used.

The traces collected at company A and company B were run off-line through an implementation of the compression algorithm in order to predict the energy gain had these networks been running protocol independent compression. Thus we are able to precisely quantify the gain of this scheme in real-world commercial applications, something an in-lab deployment would not allow. The algorithm was implemented for a single link per the description of Section III, sans CRC.

B. Compression Results and Parameter Tuning

Compression ratio is plotted versus the parameters $B$ and $S$ in Figs. 3 and 4. The compression ratios are in the range of $50 \pm 5\%$ for data from company A and $75 \pm 3\%$ 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.

The intuitive result that the compression ratio can be improved by providing the algorithm with more memory is true to the extent that unnecessary patterns identified from packets in a large packet buffer don’t flush more relevant patterns from the pattern table. With properly tuned parameters the compression ratio saturates and there is little compression benefit to further increasing memory; decreased compression may even result. From a memory and computation standpoint, it is advantageous to remain at the edge of this saturated region, or even at a point with lower compression ratio.

As is evident from Figs. 3 and 4, the compression ratio does not change drastically over the full range of $B$ and $S$; leaving both of the parameters at or near their minimum values is likely sufficient. The peak compression configurations consume approximately 400 and 1200 bytes of RAM for Companies A and B respectively. Efficient protocol-independent compression can hence be implemented on wireless sensor nodes.

V. CONCLUSION

We have demonstrated a protocol independent compression algorithm for wireless networks that requires minimal resources and which maintains shared state between communicating nodes without exchanging explicit signaling information. This algorithm may be used at any layer in a protocol stack, and may be used in conjunction with existing compaction techniques. 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, potentially more than doubling the lifetime of the network.

REFERENCES

[1] 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].


