Fault-tolerant Compression Algorithms for Sensor Networks with Unreliable Links

Alexandre Guitton[1], Niki Trigoni[2], and Sven Helmer[3]

[1] Université Blaise Pascal, Clermont-Ferrand, France
[2] Computing Laboratory, University of Oxford, Oxford, United Kingdom
[3] Birkbeck, University of London, London, United Kingdom

Abstract. We compare the performance of standard data compression techniques in the presence of communication failures. Their performance is inferior to sending data without compression when the packet loss rate of a link is above 10%. We have developed fault-tolerant compression algorithms for sensor networks that are robust against packet loss and transmission delays. We show the advantage of our technique by providing results from our extensive experimental evaluation using real sensor datasets.

1 Introduction

Large-scale multi-hop networks consisting of inexpensive battery-powered nodes that continuously monitor an environment are on the horizon. The main obstacles are the limited energy and bandwidth resources combined with unreliable wireless links. Since transmission is usually the most costly operation of a sensor node, compressing the data before transmissions seems like a good idea, as this saves bandwidth and energy. However, we cannot run just any compression algorithm: we need algorithms that are simple enough to run in an energy-conserving manner on processing-limited hardware, yet are robust enough against packet loss.

Sadler et al. [17] recently proposed computationally-efficient lossless compression algorithms and studied the local energy tradeoffs for a single data-producing node performing the compression. Their technique exhibits significant energy savings at the intermediate nodes between the source and the sink, which benefit from forwarding less data. The focus was on designing sensor-adapted versions of LZW, which are tailored to nodes with limited computation and storage capabilities. Sadler et al. generate the encoding/decoding table on the fly. Thus as soon as a single packet is lost during transmission, the rest of the data in a compression block (which spans several packets) cannot be decompressed anymore. The solution offered in [17] is for the receiver to send block acknowledgments and for the sender to try to retransmit any dropped packets. This causes significant delays in the decoding process especially in the case of large compression blocks[4] and intermittent/asymmetric links where not only data but also block acknowledgments can be lost.

There are environments in which delays are detrimental. For example, consider a target tracking application in which multiple sensors must combine their readings to localize a target, and to wake up sensors in the target's direction. In this case, timely dissemination and merging of their readings is critical in order not to miss the moving target. In fact it is preferred to successfully transmit a subset of the sighting readings in a timely fashion than to successfully transmit all of them after a long delay. The problem of delayed transmissions is exacerbated in networks that use a TDMA-based MAC protocol, in which sensors operate with low radio duty cycle to preserve energy and to avoid packet collisions.

Another example is security/surveillance or emergency applications, in which multiple sensors need to forward their readings in order to detect suspicious events or emergency situations. If one needs to continually monitor the borders of a fire-afflicted region, it is better to ensure timely data delivery from a subset of sensors at the edge of the fire, to quickly assess the extent of the fire, than delayed data delivery from all sensors. Data compression is important in emergency applications where the bursts of traffic quickly saturate the wireless medium, cause collisions and further delay packet transmission, especially in the bottleneck area around a gateway. However, it is critical to provide a means of data compression that does not incur decoding delays at the receiving end in the event of packet loss.

Our first goal is to understand the behavior of standard compression techniques in unreliable networks, and compare their ability to deliver packets on links of varying quality. Our second goal is to develop fault-tolerant variants of compression algorithms that are robust against packet loss (i.e., all successfully delivered packets can be decompressed), while at the same time being able to adjust to changing data distributions. One of the strongest points of our approach is the ability to avoid transmission delays when faced with unreliable links. We achieve this by delaying the update of the encoding/decoding tables. In the case of reliable links, our approach exhibits communication savings comparable to those of the original algorithms. But unlike the original algorithms, our technique degrades gracefully when the conditions turn unfavorable.

Like [17], we do not consider data compression techniques that are carefully tailored to specific data distributions (where we could exploit known spatio-temporal correlations to reduce communication) or to specific query types (where for some aggregate functions, we could apply in-network aggregation to reduce the cost of result propagation). A brief overview of these techniques is presented in Section 6. Our techniques are more general, and are better suited to dynamic environments with changing data distributions, little knowledge of data correlations and a wide range of queries.

[4] One the one hand, larger block sizes lead to a better compression rate, on the other hand, they also lead to longer delays.

Error correcting codes (ECCs) could be used to repair bits (within a packet) flipped during transmission. Our approach is orthogonal to ECCs, as we do *not try to repair erroneous packets*, but ensure that we are able to interpret (decompress) packets that arrive successfully. That is, we strive to preserve independence between packets, so that dropped packets do not compromise our ability to decompress received packets.

The main contributions of our paper are the following:

1. We compare standard compression techniques, like Adaptive Huffman, LZ77 and LZW, in terms of their efficiency in compressing information and sending it across lossy channels. The goal is to highlight the need for fault-tolerant compression schemes in lossy wireless environments.
2. We study existing compression techniques that use packet retransmissions to cope with communication failures. We refer to these algorithms as RT (ReTransmission) algorithms and we identify their shortcomings in the presence of intermittent or asymmetric links.
3. We design novel fault-tolerant compression algorithms (FT algorithms) that overcome the shortcomings of the RT algorithms. We assess the delay and energy efficiency of the two approaches using two performance metrics defined in Section 2. For our comparison, we apply the FT and RT algorithms to real datasets of car and animal traffic data.

The remainder of this paper is organized as follows: Section 2 describes the communication model, and defines the performance metrics that we use to assess the performance of compression algorithms in sensor networks. Section 3 discusses standard compression techniques and their RT (ReTransmission) counterparts and points out the weaknesses of these existing approaches. Section 4 introduces a novel fault-tolerant mechanism that can easily be applied to various compression techniques, leading to a novel class of FT algorithms. Section 5 presents an experimental evaluation of the RT and FT algorithms, Section 6 discusses related work and Section 7 concludes the paper and identifies directions for future work.

2 Preliminaries

Communication model: It is widely accepted in the mobile and sensor network communities that radio propagation (i) is non-isotropic, i.e. connectivity is not the same in all directions from the sender at a given distance, and (ii) features non-monotonic distance decay, i.e. smaller distance does not mean better link quality, and (iii) exhibits asymmetrical links [1, 5, 9, 12, 23]. It has also been shown that there is considerable difference from link to link in the burstiness of the delivery, and most node pairs that can communicate have intermediate loss rates [1].

Our work is motivated by the difficulties that arise from applying standard compression algorithms and propagating compressed data across asymmetric links with intermediate loss rates. Depending on the link quality, interference and other environmental conditions, a number of packets will be dropped, and energy or channel considerations may not allow us to retransmit until 100% delivery is achieved.

In our study, we consider two packet loss models: failure of data packets:

- A model in which packet losses are independent and identically distributed; in this case, we vary the probability of successful packet delivery to study the performance of compression algorithms in varying error conditions.
- A first-order Markov model for the success/failure process of packet transmissions. This bursty error model, known as the Elliot-Gilbert model [8, 10], is shown to accurately approximate the behavior of a fading radio channel in [25].

Performance metrics: We need to measure the amount of information that is successfully received and usable at the receiver node, as well as the delay until it becomes usable. Consider a dataset of sensor data that has been accumulated at a sender node S and must be delivered to a receiver node R along a lossy link. We compare the efficiency of various data compression techniques based on the following metrics:

Bytes of Decoded Data / Bytes Sent (BDD / BS): The metric BDD measures the number of bytes derived from successfully decoding received packets. When data is propagated without compression, this metric measures the number of bytes in successful packet transmissions. When data is propagated in a compressed manner, some packets may be successfully received, but not successfully decoded. These packets will not add to the bytes of decoded data (BDD). Hence the metric BDD/BS represents the amount of *useful* information received by the destination R, relative to the effort that the source node S puts to sending it. In an ideal channel without packet drops, this metric reflects the compression ratio achieved by an encoding scheme, i.e. the number of bytes of uncompressed data divided by the number of bytes of the compressed representation. In the context of unreliable channels, this metric combines two aspects of an algorithm's behavior: (i) its ability to reduce the size of the original sensor data as reflected by the compression rate (as in the ideal channel) and (ii) how well it encodes data into a format that can be decoded safely even in the presence of packet drops.

Delay (DEL). This metric measures the delay between the time that a packet is first sent from the sender node S and the time that it is ready to be successfully decoded by the receiver R. The delay has two components: *delivery time* and *decode waiting time.*

For packets delivered successfully the first time they are sent, the delivery time is equal to the packet transmission time: the latter consists of the delay waiting for access to the transmit channel (access time), the time needed for the packet to transit from S to R (propagation time) and the processing time required for R to pick up the packet from the channel (receive time). For simplicity, we assume that the packet transmission time is constant for all packet transmissions and lasts for 1 time unit. If a packet is dropped on the first attempt, and there are k intermediate packet transmissions (of the same or other packets) before it is retransmitted successfully, then the packet's delivery time is $k + 2$ time units.

The *decode waiting time* is the time between the arrival of a packet and its decompression. When the compression dictionary is up-to-date on receipt of a packet (or no compression is used), the decode waiting time is 0. We will show in Section 3 that existing compression algorithms require packets to be received in the order in which they were encoded, so that they can keep the compression dictionary up-to-date and decode packets successfully. In this case, packets received out of order may experience higher delays before they can be decoded, until all packets that preceded them in the encoding process are also successfully received by R.

3 Existing Algorithms

Before delving into details on how our adapted compression algorithms for sensor networks operate, we are going to briefly review some popular compression algorithms. We will demonstrate why these algorithms fail to achieve good performance when deployed in typical sensor network scenarios and show how an approach by Sadler and Martonosi [17] tries to overcome some of these problems.

3.1 Standard Compression Algorithms

Due to space constraints, we will only describe those algorithms that we are adapting to our approach later on.

LZW LZW is a refinement of the Ziv-Lempel approach by Welch [21]. LZW replaces strings of symbols with (shorter) code words. The mapping of strings of symbols is done with the help of a table (or dictionary). The initial table maps individual symbols to default codes, e.g. characters to their ASCII code. Every time a new string is encountered – a substring of length $i + 1$ starting with the current symbol that is not yet found in the table – the code for the substring is output and a new code for the substring of length $i + 1$ is added to the table. Substring codes are restricted to a certain number of bits and a new code is assigned the smallest unused number. When decompressing, the coding table can be built on the fly, as long as the same initialization table is used on both the sender and the receiver side.

Adaptive Huffman Coding Standard Huffman coding [11] uses a compression/decompression dictionary that is based on known symbol frequencies. Adaptive Huffman coding [20] determines the symbol frequencies based on the input it receives and builds the dictionary on the fly. Both sender and receiver start with an empty tree and for each symbol that is transmitted, modify the tree accordingly.

Each node in Adaptive Huffman coding has a weight associated with it: the number of occurrences of a certain symbol for leaf nodes, and the sum of the weights of the children for an inner node. In addition to this, nodes are numbered in non-decreasing order by weight. This numbering scheme is called *sibling property* and reflects the order in which nodes are combined using regular Huffman coding. When updating a tree after the transmissions of a symbol, it does not suffice to adjust the weights of the corresponding leaf node and its ancestors. In order to guarantee the sibling property, the algorithm may have to swap some nodes.

3.2 Weaknesses of Standard Compression Algorithms

Let us now have a look at how the algorithms described above perform in a typical sensor network environment. We simulated the transmission of road traffic data over a lossy link varying the packet drop rate of the link from 0% to 90%.

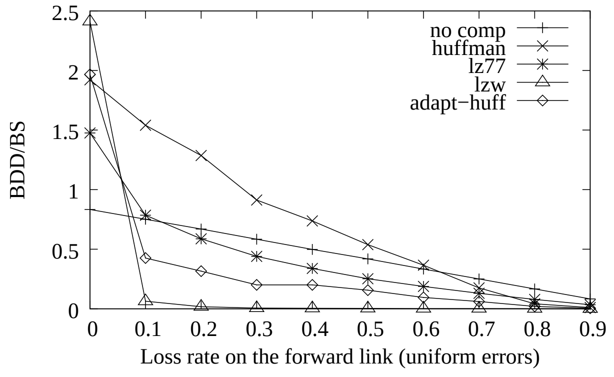


Fig. 1. Using standard compression algorithms

Our findings are quite surprising. In the presence of packet loss, most of the compression algorithms (except for Huffman coding) degrade to the point where using them is worse than not compressing at all.[5] What are the reasons for this? Losing packets has a devastating effect on LZ77 [24], LZW, and Adaptive Huffman coding, because these algorithms rely on previously decompressed data to successfully decompress the remainder of the data. If we lose any of this data, the algorithms will not be able to continue decoding packets successfully. Ordinary Huffman coding can restart decompression after losing packets, since it does not rely on previous data. This makes it robust against packet losses, once the dictionary has been sent successfully. Nevertheless, Huffman coding also has

[5] The value for BDD/BS is lower than 1 for no compression due to the overhead of packet headers.

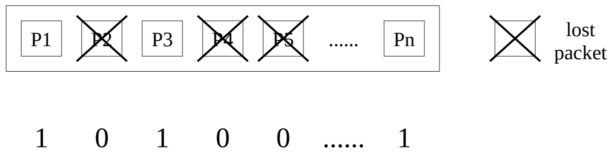


Fig. 2. Block acknowledgment

drawbacks. It needs to know the data distribution before starting in order to yield an acceptable compression ratio. Every time this distribution changes, we have to construct a new encoding table and transmit this table to the receiver side. Doing so over an unreliable channel may result in considerable overhead.

3.3 S-LZW

Sadler and Martonosi adapted the LZW compression algorithm to sensor networks [17]. Packets are sent in blocks. In order to cope with lost packets, they keep the size of blocks containing interdependent packets small. Each block is compressed with its own dictionary, which means that a packet loss will only affect the following packets of the same block. A small block size also means that the dictionary for encoding the packets of the block will not grow very large (Sadler and Martonosi work with dictionaries having a size of 512B) keeping the memory requirements down.

When looking at the effects of unreliable links on their algorithm, Sadler et al. only take into consideration the energy savings of their approach. When dealing with reliability issues they rely on a RT (ReTransmission) strategy [17]:

> The receiver then sends a 32B acknowledgment that tells the sender which packets it received correctly and the sender retransmits what is missing. If the transmission is not complete and the receiver does not start receiving data immediately, it assumes that the acknowledgment was lost and retransmits it. This process iterates until the receiver has successfully received and acknowledged the whole block.

However, there are no results in the paper on how much delay this causes to the data transmission. Let us illustrate this with a drastic example: assume that we transmit a block consisting of 50 packets and the first packet gets lost. All packets after the first cannot be decompressed until the first packet arrives. This retransmission will take place only after an acknowledgment has been sent, which means that there is considerably delay until the whole block is decompressed. Worse, if the connection breaks down before the first packet is resent, the whole information contained in the first block is lost. This is why Sadler and Martonosi opt for small block sizes. This, however, compromises LZW's compression rate, as LZW needs some time before reaching its full compression potential. We will

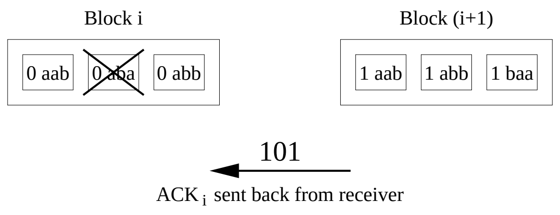


Fig. 3. Example for blockwise transmission

demonstrate the adverse effects of the RT strategy on the speed of delivering packets in Section 5.

4 Proposed Fault-tolerant Algorithms

In the previous section, we showed that the performance of standard compression techniques degrades significantly in the presence of packet losses, and advocated the need for fault-tolerant algorithms. The current state of the art in fault-tolerant compression consists of RT algorithms that retransmit packets in case they are dropped with the goal of achieving 100% delivery. We have shown that the RT algorithms have two weaknesses: i) they cannot handle sudden interruptions of connectivity along a link (e.g. in mobile ad-hoc networks) and ii) they incur high delays in packet decoding when the forward or backward channel is lossy. The cause of these problems is that both standard and RT compression algorithms update their dynamic dictionaries as soon as new data is being compressed or uncompressed. When no fault-tolerant mechanism is provided and some packets are lost, the sender and receiver end up using different versions of the dictionary. The RT algorithms amend this problem by retransmitting all dropped packets, but this significantly delays the process of decoding received packets.

In this section, we propose a novel mechanism that addresses these problems and can be applied to standard compression techniques, like Adaptive Huffman, LZ77 and LZW. The proposed class of algorithms, referred to as FT algorithms (e.g. FT-LZW), are robust to sudden interruptions of connectivity, and they incur considerably less delay than the RT algorithms when the forward or backward channel is lossy. Our fault-tolerant mechanism consists of three parts:

– **Block acknowledgments:** The sender sends packets in blocks (a block consists of n packet transmissions; see upper part of Figure 2 for an example) and expects to get an acknowledgment (ACK) from the receiver at the end of a block. Packets of a block are numbered from 1 to n, and when the receiver successfully receives a packet, it reads the sequence number of the packet. At the end of a block, the receiver sends an ACK that contains an n-bit vector. The i-th bit of the vector is 1 if packet i was successfully received, and 0 if it was dropped (see lower part of Figure 2).

– **Periodic dictionary updates:** Dictionary updates occur at the sender only immediately after receiving a block ACK. The sender reads the bit vector in the ACK to understand which packet transmissions were successful. It then modifies the dictionary by considering only the symbols encoded in the successfully delivered packets. It updates the dictionary, and uses the new updated dictionary to encode information in the next block of packets. Of course, the exact mechanism in which the dictionary is updated depends on the specifics of the algorithm. However, the idea of updating a dictionary not after each symbol, but after a block of successfully received symbols, is common to all FT algorithms and can be applied easily to a variety of compression techniques like LZW, LZ77 or Adaptive Huffman.
– **Countering link unreliability:** Since the medium is unreliable, it is possible that the ACK sent by the receiver is lost. However, it is crucial for updating the dictionary that both the sender and the receiver agree upon which packets were successfully transmitted. Instead we propose the following simple protocol: If the sender receives a block ACK, it updates the data dictionary, otherwise the dictionary remains unchanged. Each data packet transmitted by the sender (encoder) in the next block contains a flag that denotes whether the encoder has received an ACK message for the last block and has changed its dictionary accordingly. For example, when the packets of the current block start with 1, it means that the encoder received an ACK for the previous block and changed its local dictionary based on the symbols included in the successful packets. This notifies the receiver (decoder) to also update its local version of the dictionary based on the received packets of the previous block, before using the dictionary to decode the packets of the current block.

Let us illustrate the application of our mechanism to Adaptive Huffman (the resulting algorithm being FT-Adapt-Huff). Assume that the sender sends blocks of three packets each, and the receiver sends ACKs back at the end of each block, as depicted in Figure 3. If we assume that the ACK sent for Block (i-1) is dropped, the sender does not update its dictionary before starting to transmit packets of Block i. The sender denotes this by starting each packet of Block i with the flag 0. The receiver also refrains from updating the dictionary before decoding packets of Block i. At the end of Block i, the sender receives ACK 101 and updates its dictionary by inserting the following stream of symbols into the coding tree: a,a,b,a,b,b. As soon as the receiver picks up a packet of Block (i+1) with a leading 1 bit, it also modifies its dictionary according to the symbols in the acknowledged packets of Block i and then immediately starts decompressing the packets of Block (i+1) based on the updated dictionary.

Note that this fault-tolerant mechanism is applicable to several compression algorithms with dynamic dictionaries. An important feature of this mechanism is that it does not delay the transmission of data, but only the adaptation of the dictionary at the encoder and decoder. This is in contrast with the RT algorithms who delay the decoding of a received packet, until all preceding packets are successfully retransmitted. Our approach requires buffering a block's data locally at the encoder and the decoder, in order to adjust the data dictionary at the end of each block, in case an ACK is received. Our approach is fault-tolerant to losses of both data packets and ACK packets, and it is suitable for links with intermittent connectivity, that occasionally become asymmetric thus prohibiting the transmission of ACKs. It combines adaptive encoding with the ability to survive packet drops on links with intermediate or low link quality.

5 Experimental Evaluation

In Section 3, we discussed standard compression algorithms and showed that they are not suitable for sensor networks with lossy links. In particular, Figure 1 illustrated that most adaptive compression algorithms behave worse than having no compression at all (in terms of our metric: Bytes of Decoded Data / Bytes Sent), when more than 10% of the packets sent across a link get dropped.

In this section, we evaluate the performance of compression algorithms that have a built-in mechanism for handling packet drops. We compare the existing class of RT algorithms [17], which are based on packet retransmissions, to our novel class of FT algorithms, which are based on periodic dictionary updates.

5.1 Simulation Setup

In our comparison, we use two performance metrics defined in Section 2: i) BDD/BS and ii) DEL. Recall that the first metric measures the number of decoded bytes at the receiver relative to the number of bytes sent by the sender, and therefore reflects the communication and thus energy efficiency of an algorithm. The second metric reflects the delay between the time that a packet is first sent by the sender to the time that it is ready to be decoded at the receiver. The delay is measured in abstract time units.

In our experiments, we vary the error rate (packets dropped / packets sent) of the forward and backward direction of a lossy link. In both FT and RT algorithms, the forward link is used to send compressed data, whereas the backward link is used to propagate ACKs. When not explicitly defined, the default value of error rate at the forward and backward links is 20%. We compared existing and new algorithms using both uniform and bursty error rates. Because of limited space, we decided to include only the graphs that concern bursty errors which models the real world more closely. The Elliot-Gilbert loss model is initialized with two parameters p and r, where p is the probability that a packet transmission is successful given that the previous one was successful, and r is the probability that a packet transmission is successful given that the previous one was unsuccessful. Note that $1/r$ represents the average length of a burst of errors, and $1/(1-p)$ the average length of a burst of successful transmissions. In steady state, the percentage of erroneous transmissions ϵ is:

$$\epsilon = 1 - \frac{r}{1-p+r}.$$

To achieve a loss rate of ϵ, we choose $p = 1 - 0.3\epsilon$ and $r = (1 - \epsilon)(1 - p)/\epsilon$. For $\epsilon = 0.2$, this results into having $p = 0.94$ and $r = 0.24$.

We also consider the impact of the block size on the performance of the two classes of algorithms. Recall that grouping packets into blocks has a different role in the two classes of algorithms: at the end of each block, the FT algorithms update their dictionaries, whereas the RT algorithms reset their dictionaries and start compressing the new block with an empty dictionary. Both FT and RT algorithms can use block ACKs to retransmit dropped packets if the application requires 100% accuracy. We set the packet size to 60 bytes, of which 10 bytes are the header. This explains why the value of BDD/BS is less than 1 in the no-compression algorithm in Figures 1, 4 and 6.

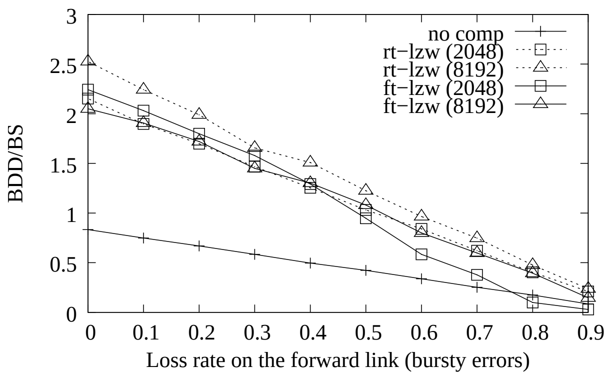
Finally, the value of BDD/BS may be affected by the selection of a compression algorithm or the data to be compressed. To capture differences, we apply the FT and RT mechanisms to two different compression algorithms, LZW and Adaptive Huffman. The default dictionary size of LZW in both approaches is 8192B. We also used two different real datasets, Scoot and Zebranet: the Scoot data is generated by 118 inductive loops monitoring the flow of cars on several roads in Cambridge, UK[6]. We focus on the data generated by 68 sensors which is propagated along the most congested branch of the collection tree, from the neighbor of the gateway to the gateway. We also use a dataset from the Zebranet 2 experiment, in which several zebras wear a collar with a GPS device. Each data record includes zebra sighting information, including GPS coordinates and time-stamps[7].

5.2 Simulation Results

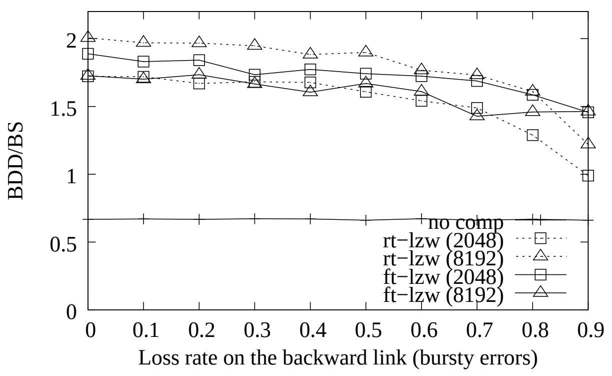
Impact of loss rate on the forward link: We start by applying the FT and RT mechanisms to the LZW algorithm, and compare the two approaches as we vary the rate of packet losses across the forward link. We use a bursty error model, i.e. we have bursts of successful packet transmissions followed by bursts of dropped packets. We compare FT and RT algorithms using blocks of two different sizes: 2048 bytes and 8192 bytes, corresponding to approximately 20 and 66 packets respectively.

Figure 4(a) shows that, in terms of BDD/BS, both FT and RT algorithms degrade gracefully (drop linearly) as we increase the loss rate of the forward link. They are both 2-3 times more efficient than not compressing data at all, even for very lossy links. This is in contrast with the standard compression algorithms that behaved worse than no-compression when the loss rate exceeded 10% (Figure 1). In terms of communication efficiency (BDD/BS) we observe very small differences between the FT and RT approaches, which depend on the block size used. These are explained in detail in the paragraph that discusses the impact of the block size.

[6] http://www.cl.cam.ac.uk/research/dtg/~rkh23/time/timeloops.html
[7] http://crawdad.cs.dartmouth.edu/meta.php?name=princeton/zebranet



(a)

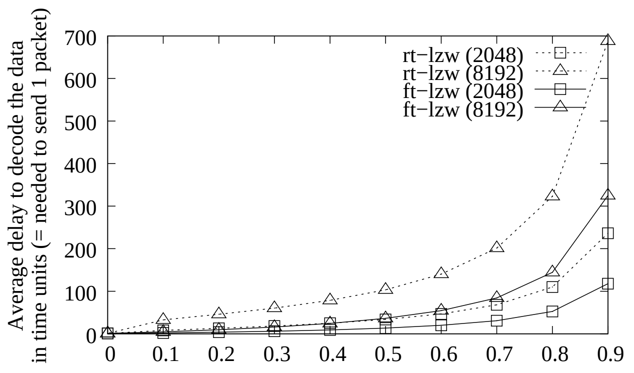


(b)

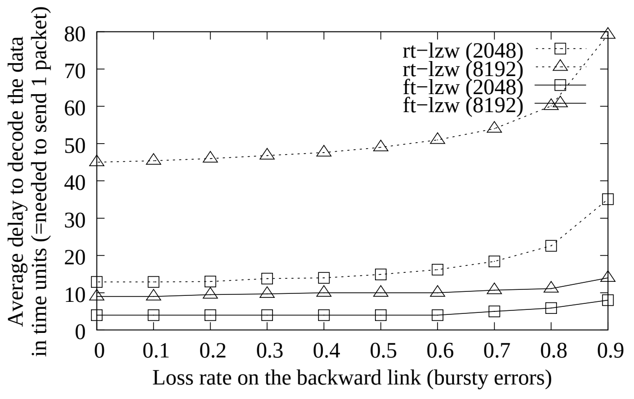
Fig. 4. Comparing RT to FT algorithms in terms of BDD/BS for forward and backward errors

However, in terms of delay, our FT algorithms significantly outperform their RT counterparts. Figure 5(a) shows that as we increase the loss rate of the forward link, more compressed packets get dropped, and the average number of retransmissions per packet increases. Hence, both FT and RT algorithms experience higher delays due to retransmissions. However, the delay exhibited by the RT algorithms is 2-3 times higher than that by the FT algorithms for a given loss rate. The reason is that in the FT algorithms, packets can be decoded as soon as they are successfully received, whereas in the RT algorithms, packets can be decoded only if all preceding packets in the block are successfully received and decoded.

Impact of loss rate on the backward link: We can observe similar behavior of the FT and RT algorithms as we vary the loss rate of the backward link. Figure 4(b) shows that the communication efficiency of the two approaches is comparable, and 2-3 times better than not compressing data at all. It also



(a)



(b)

Fig. 5. Comparing RT to FT algorithms in terms of delay for forward and backward errors

shows that the efficiency of both classes of algorithms drops very slowly as we increase the packet drops in the backward link. The reason is different for the two classes of algorithms. In the case of the RT algorithms, as more ACKs are lost, we observe multiple retransmissions of ACK packets at the end of each block, until one of them is received by the data sender. This incurs a communication overhead that adversely affects BDD/BS. In the case of the FT algorithms, ACK packets that get dropped are not retransmitted. However, because the sender of the compressed data does not receive them, it does not update the compression dictionary, and thus compresses the packets in the next block less efficiently than if the dictionary was properly updated.

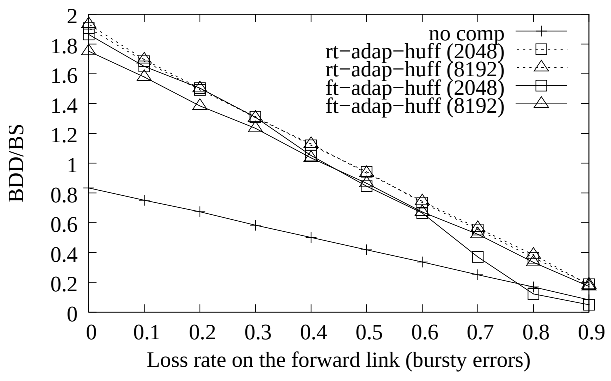
Figure 5(b) shows another interesting feature of our FT algorithms, that makes them particularly appealing in the presence of asymmetric links. The delay of successfully decoding packets at the receiver remains almost constant as we increase the loss rate of the backward link from 0 to 90%. However, the delay of the RT algorithms raises significantly as we increase the error rate in the backward link. The reason is that the sender waits for an ACK packet before proceeding to the next block, which never arrives due to bursty errors in the backward link. Hence, our FT algorithms are particularly delay-efficient in the presence of asymmetric links with good quality of the forward link, but bursty and unreliable connectivity in the other direction.

Impact of block size: Let us now compare the FT and RT algorithms in Figure 4(a) paying particular attention to the block size. When we use small blocks of 2048 bytes, and for low loss rates, FT-LZW outperforms RT-LZW in terms of BDD/BS. The reason is that RT-LZW resets the dictionary at the end of each block, thus degrading its efficiency when trying to compress the first packets of the new block with an empty dictionary. In contrast, FT-LZW updates its dictionary periodically at the end of each block. However, we observe an interesting opposite trend when the loss rate of the forward link rises above 60%. In this case, using small block sizes in FT-LZW raises the danger of dropping all packets within a block, and thus not being able to synchronize successfully the dictionaries at the sender and the receiver. Hence, it is preferred to use larger block sizes in the FT algorithms in the extreme case of highly lossy links.

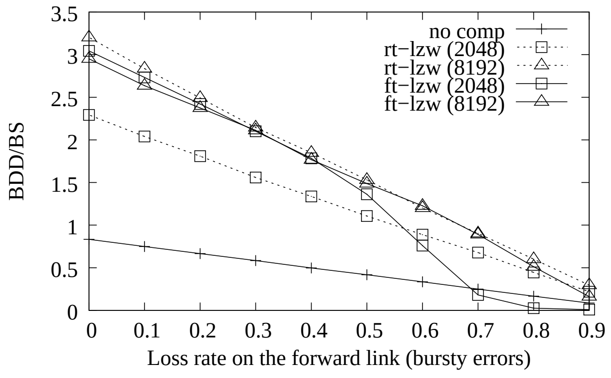
When we use a larger block size (8192 bytes), we observe a different effect. RT suffers less often by the problem of resetting the dictionary, whereas FT compresses data less efficiently because the dictionary is updated less frequently (at the end of each long block). Thus, RT-LZW with block size 8192 bytes, is slightly more communication efficient than FT-LZW with the same block size. The question that arises is whether in highly mobile environments, it is realistic for RT-LZW to use large blocks. When the sender and receiver are data mules that might get disconnected at any time, using large block sizes in the RT algorithms is not advisable. The reason is that we risk propagating large amounts of information which cannot be successfully decoded because the block transmission is interrupted before being able to retransmit successfully some of the initial packets of the block that got dropped. These initial packets are crucial to successfully decoding all ensuing packets.

The effect of block size on delay of the FT and RT algorithms is illustrated in Figure 5. For a given loss rate of the forward link (or of the backward link), the higher the block size, the higher the delay in both FT and RT algorithms. The reason is that the larger the block, the longer the sender waits for an ACK that is necessary to determine which packets need retransmission. As we mentioned above, FT algorithms are always faster than RT algorithms, given a fixed block size, because the packets sent by the FT algorithms are ready to uncompress as soon as they arrive at the receiver, even if preceding packets have not arrived successfully.

Impact of compression algorithm and dataset: The last set of experiments was conducted to measure BDD/BS with a different compression algorithm (Adaptive Huffman instead of LZW), or with a different real dataset (Ze-



(a)



(b)

Fig. 6. Comparing RT to FT algorithms (BDD/BS for Adaptive Huffman and Zebranet)

branet instead of Scoot). As expected the delay measurements are not affected by the selection of algorithm or dataset, hence we omitted these graphs to save space. We only present the behavior of the FT and RT approaches in terms of communication efficiency (BDD/BS). Figure 6(a) shows that the FT-Adapt-Huff and RT-Adapt-Huff are comparable in terms of BDD/BS, they degrade gracefully as we vary the loss rate of the forward link, and they significantly outperform the policy of propagating data without compressing them. These conclusions are very similar to the ones drawn when we compared FT-LZW and RT-LZW.

Figure 6(b) shows the performance of FT-LZW and RT-LZW when applied to the Zebranet instead of the Scoot dataset. Here, we observe a clearer pattern of how the behavior of RT-LZW changes as we decrease the block size from 8192 to 2048 bytes. The adverse effect of restarting the dictionary more frequently is more obvious in the Zebranet dataset. Hence, in networks with intermittent connectivity, where RT algorithms must use small block sizes, they are inferior to our proposed FT algorithms, not only in terms of delay, but also in terms of communication overhead and thus energy efficiency.

5.3 Experiment with real motes

In order to prove the feasibility of our FT mechanism in a real sensor system, we implemented FT-Adapt-Huff on Tmote Sky nodes. Our goal is to validate that the FT mechanism performs similarly in a real setting as in a simulation environment, and to measure the memory requirements of the application on resource-constrained nodes (Tmote Sky motes have 10kB of RAM and 48kB of program space). The NesC code, which is available online[8], requires approximately 19 kB of program space and 7kB of RAM to store the Adaptive Huffman tree and buffers of uncompressed and compressed data. The algorithm updates the dictionary periodically every block of 10 packets of 28 bytes each (8 bytes header and 20 bytes payload).

We use two sensor nodes, a sender and a receiver, and vary the distance between them in order to change the packet loss rate. Each experiment runs for 10 minutes, which generates approximately 1780 bytes of data at the sender, at a rate of 3 bytes per second. The data is a subset of the Zebranet2 dataset. The number of packets sent varies from 45 to 89 packets (4 to 8 blocks) depending on the link quality. When the link quality is low, many packets are dropped, the dictionary is updated less frequently and the potential for compression decreases leading to more packet transmissions.

Figure 7 shows the efficiency of FT-Adapt-Huff over No-Comp, in terms of BDD/BS. When there is no packet loss, the value of BDD/BS for FT-Adapt-Huff is 1.8, compared to 0.7 for no compression. In the latter case, the value is less than one due to the overhead of the header. In the case of faulty links, the performance of FT-Adapt-Huff degrades gracefully with the packet loss rate, as we observed in the simulation results. It remains better than no compression for all packet loss rates.

6 Related work

Compression, the process of finding and removing redundancy in information, is a powerful tool for data propagation in energy- and bandwidth-constrained sensor networks. Classical data compression theory studies trade-offs between rate (bits / sample) and distortion (e.g. mean square error) between the original and reconstructed signal. In sensor networks, a decrease of the compression rate lowers the amount of energy spent in communication, while an increase in distortion results in lower accuracy of sensor data returned to the application users. A detailed discussion of data compression trade-offs is presented in [6]. In our paper, we focused on lossless algorithms (no distortion is allowed).

[8] http://web.comlab.ox.ac.uk/oucl/work/alexandre.guitton/ft-adaphuff/ft-adaphuff.html

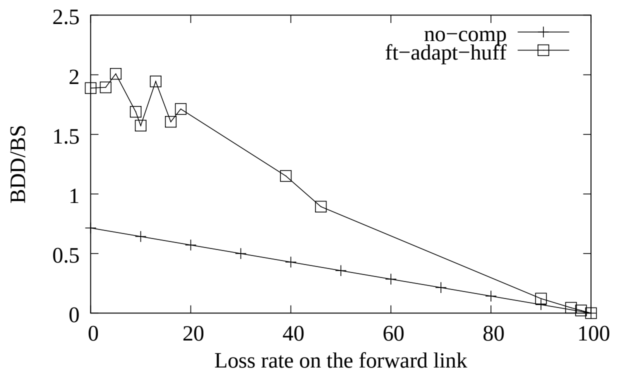


Fig. 7. Comparing No-Comp and FT-Adapt-Huff on a real platform

A large body of research has recently focused on distributed compression techniques for sensor networks [3, 7, 14]. According to the Slepian-Wolf coding theorem [19], if the joint distribution quantifying the correlation structure between two data sources X and Y is known, then the encoder of X can achieve the same compression ratio as if it knew exactly the data generated by source Y. This means that in theory, sensor nodes can compress their data independently without internode communication, by exploiting knowledge of correlations with other nodes. In our work we do not assume any spatio-temporal correlations among nodes, which might anyway change dynamically over time, but instead we focus on compressing data that is available locally at each node independently of other sensors' data.

Scaglione et al. [18] study the interdependence of routing and data compression in wireless sensor networks (WSNs). They consider the problem of collecting samples of a field at each sensor node, and using them to obtain an estimate of the entire field within a distortion threshold. Puri et al. [15] consider the problem of rate-constrained estimation of a physical process based on reliably receiving data from only a subset of sensors. Rachlin et al. [16] study the interdependence of sensing and estimation complexity in WSNs. Their goal is to estimate the underlying state of the environment based on noisy sensor observations. They show that by taking advantage of additional sensor measurements, it is possible to reduce the computational complexity of estimation (decoding). Xiao et al. [22] study how the signal processing capability of a WSN scales up with its size, and explore distributed signal processing algorithms with low bandwidth requirements. They derive efficient local quantization schemes and distributed estimators of the quantized sensor data under energy and bandwidth constraints. Whereas these works focus on estimating a physical phenomenon based on unreliable noisy data, our paper aims to compare the performance of standard source coding techniques in terms of reliability, and to devise fault-tolerant versions of these techniques for WSNs. The impact of spatial correlation on routing with compression has recently been studied by Pattem et al. [13]. Unlike our work, they consider carefully selecting routes to maximize the potential of compressing spatially correlated data.

Our work is more similar to the study of Sadler et al. on data compression algorithms for energy-constrained devices [17]. Like [17] we focus on compressing data locally without considering correlations with other nodes, in order to reduce the overall energy consumption of wireless data propagation. Unlike [17] that focuses on designing algorithms for nodes with limited computation and storage constraints, our goal is to tackle the problem of unreliable transmissions of compressed data. PINCO [2] buffers sensor data at each node for as long as possible, whilst adhering to user-defined end-to-end delay requirements. Energy savings are achieved by reducing available redundancy of buffered data before communicating them over the wireless medium. Barr et al. [4] measure the energy requirements of several lossless data compression schemes using a specific (Skiff) platform. They point out that energy savings can be achieved in some cases by not applying the same algorithm for both compression and decompression. Unlike our work, [2] and [4] do not address the problem of recovering from unsuccessful transmissions of compressed data.

7 Conclusions

This paper is focused on fault-tolerant mechanisms for compressing data in wireless networks with unreliable links. We demonstrated that standard compression techniques behave worse in terms of communication efficiency than using no compression at all, on links with packet loss rates greater than 10%. We thus directed our attention at robust compression techniques, namely the existing RT (ReTransmission) mechanism and our novel FT (Fault-Tolerant) mechanism.

We showed that the FT mechanism is comparable to the RT mechanism in terms of communication efficiency in relatively static networks. Both algorithms degrade linearly in the loss rate of the forward link. However, the proposed FT algorithms are preferred in terms of communication efficiency in scenarios with frequent network disconnections, e.g. when two mobile data mules exchange data whilst being in range, and get disconnected as they move away from each other. In this case, RT algorithms with large block sizes are at risk of not being able to uncompress a large volume of successfully transmitted packets of a block, just because the first packet of the block is not retransmitted successfully before the disconnection. On the other hand, RT algorithms with small block sizes simply cannot achieve a high compression ratio, because they reset the compression dictionary at the end of each block.

In addition, our proposed FT algorithms are 2-3 times faster than their RT counterparts in delivering usable packets. The reason is that the FT mechanism allows each successfully received packet to be decompressed immediately without waiting for preceding packets of the same block to arrive successfully. The delay of FT algorithms remains relatively constant even for asymmetric links, i.e. cases where the backward link has a prohibitively high loss rate close to 80%.

In the future, we plan to investigate fault-tolerant compression in large-scale sensor deployments. As we propagate data along multi-hop paths, we have two choices: to compress/decompress data either hop-by-hop or only at the two ends of the path. The former may allow us to exploit correlations of data coming from neighboring nodes, but at the same time it would require more computation and memory resources. Our plan is to compare the two approaches in terms of energy-efficiency, delay and robustness.

References

1. D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *SIGCOMM*, pages 121–132, 2004.
2. T. Arici, B. Gedik, Y. Altunbasak, and L. Liu. Pinco: A pipelined in-network compression scheme for data collection in wireless sensor networks. In *ICCCN*, 2003.
3. S. Baek, G. de Veciana, and X. Su. Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation. Technical report, University of Texas, 2004.
4. K. Barr and K. Asanovic. Energy aware lossless data compression. In *MOBISYS*, 2003.
5. A. Cerpa, N. Busek, and D. Estrin. Scale: a tool for simple connectivity assessment in lossy environments. Tech report CENS-21, UCLA, 2003.
6. M. Chen and M.L. Fowler. Data compression trade-offs in sensor networks. In *SPIE*, 2004.
7. J. Chou, D. Petrovic, and K. Ramchandran. A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks. In *INFOCOM*, 2003.
8. E.O. Elliot. Estimates of error rates for codes on burst-noise channels. *Bell Systems Technical Journal*, 42:1977–1997, September 1963.
9. D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. CSD-TR 02-0013, UCLA, February 2002.
10. E.N. Gilbert. Capacity of a burst-noise channel. *Bell Systems Technical Journal*, 39:1252–1265, September 1960.
11. D.A. Huffman. A method for the construction of minimum redundancy codes. *IRE*, 40:1098–1101, September 1952.
12. D. Kotz, C. Newport, and C. Elliott. The mistaken axioms of wireless-network research. Technical Report 467, Dartmouth College Computer Science, July 2003.
13. S. Pattem, B. Krishnmachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. In *IPSN*, 2004.
14. S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed compression in a dense sensor network. *IEEE Signal Processing Magazine*, 2002.
15. R. Puri, P. Ishwar, S.S. Pradhan, and K. Ramchandran. Rate-constrained robust estimation for unreliable sensor networks. In *AsilomarSSC*, volume 1, pages 235–239, 2002.
16. Y. Rachlin, R. Negi, and P. Khosla. On the interdependence of sensing and estimation complexity in sensor networks. In *IPSN*, pages 160–167, 2006.
17. C.M. Sadler and M. Martonosi. Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks. In *SENSYS*, November 2006.

18. A. Scaglione and S.D. Servetto. On the interdependence of routing and data compression in multi-hop sensor networks. In *MOBICOM*, pages 140–147, 2002.
19. D. Slepian and J. K. Wolf. Noiseless coding of correlated information sources. *IEEE Trans. Inform. Theory*, IT-19:471–480, July 1973.
20. J.S. Vitter. Design and analysis of dynamic Huffman codes. *J. ACM*, 34(4):825–845, 1987.
21. T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.
22. J.-J. Xiao, A. Ribeiro, G. B. Giannakis, and Z.-Q. Luo. Distributed compression-estimation using wireless sensor networks. *IEEE Signal Processing Magazine*, 23(4):27–41, 2006.
23. J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SENSYS*, pages 1–13, 2003.
24. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transac. on Inf. Theory*, 23(3):337–343, 1977.
25. M. Zorzi, R.R. Rao, and L.B. Milstein. On the accuracy of a first-order markov model for data transmission on fading channels. In *ICUPC*, pages 211–256, 1995.