Online Compression of Multiple IoT Sources Reduces the Age of Information

Niloofar Yazdani, *Member, IEEE*, and Daniel E. Lucani, *Senior Member, IEEE*

***Abstract*—Timely delivery of sensor data is crucial for a wide array of Internet-of-Things (IoT) applications. Due to the large space and time correlation of sensor data, there is a high potential for compression. However, conventional wisdom dictates that compression is at odds with information freshness and timely delivery of data. The reason is that sufficient data needs to be accumulated in order to achieve reasonable compression rates, which introduces additional delays on data transmission. This article studies a novel approach to perform online compression of data across multiple data sources which achieves significantly better performance in both Age of Information (AoI) and compression for sensor applications. More specifically, we show that our approach can remove the tradeoff between these two metrics, particularly, when considering an instantly decodable variant of our approach. We also propose and study techniques to further improve both these metrics by using preset and dynamically created dictionaries at the source nodes. Using real-world data sets, we show that our solution reduces the AoI (by up to a factor of 2.3) and compression ratio (by up to an order of magnitude) with respect to DEFLATE and LZW. Finally, we show that using multiple sources benefits results in an improvement of AoI and compression for each involved source compared to compressing individually.**

***Index Terms*—Age of Information (AoI), generalized data deduplication (DD), lossless data transmission reduction, multiple source nodes, timeliness, wireless sensors.**

I. Introduction

EMERGING applications powered by the Internet of Things (IoT) impose strict requirements on timely delivery of the data. In many scenarios, delivering *fresh* information is key to guaranteeing safety of people or machinery in the physical world. For instance, reliable and low latency transmission of vehicle position and speed in unmanned aerial vehicular networks [2] can not only affect the network's performance, but also avoid accidents and property damage. Age of Information (AoI) [3], [4] was proposed to quantify this freshness of the information when reaching the sink node. More precisely, the AoI quantifies the time interval between the present time and when the most up-to-date symbol at the sink node was generated by its source. Although throughput and packet delay have been used to characterize data transmission, AoI provides an information-centric, holistic view of the process.

Different system models and restrictions have spawned different metrics to characterize the AoI. For example, Zhong and Yates [5] focused on the average delay for lossless source coding systems with random symbol arrivals. In fact, it was used to characterize fixed-to-variable block coding schemes proposed by the authors. Inoue *et al.* [6] derived a general formula for the stationary distribution of the AoI, which is useful to characterize complex systems. Recent studies have also extended the AoI analysis to incorporate energy restrictions, e.g., wireless powered sensor networks that generate updates when the device's battery is fully charged [7]. From a system's perspective, Yates [8] characterized the effect of having parallel servers and Sun *et al.* [9] focused on queuing discipline. For correlated sources, the concept of Age of Correlated Information (AoCI) was proposed to solve scheduling transmissions from various sources [10]. Javani *et al.* [11] studied the AoI for several information sources being monitored by multiple independent servers considering last-come–first-serve queue model.

AoI minimization has attracted massive interest from the research community studying a variety of problems. The work in [12] introduced strict packet deadlines and small buffer sizes, thus, deleting packets after a certain amount of time as a way to improve the resulting AoI of the remaining packets. Scheduling policies at the link level [13] and considering resource-limited devices [14] have been analyzed, while policies to minimize the weighted sum AoI of multinode wireless networks have also attracted attention, e.g., [15]. Zhou and Saad [16] proposed the use of nonuniform status packet sizes as an enabler to minimize the AoI, while other studies have focused on sampling techniques [17] to achieve similar reductions. The work by He *et al.* [18] studied the minimization of AoI with multiple correlated video camera streams by allocating processing in available fog resources.

One component affecting AoI is the time invested in data transmission. Although data compression is a natural solution to reduce this time without changing other system parameters, e.g., bandwidth and transmission power, it is usually at odds with AoI. The reason is simple: efficient compressors

Manuscript received August 1, 2020; revised December 2, 2020 and January 28, 2021; accepted February 28, 2021. Date of publication March 8, 2021; date of current version September 23, 2021. This work was supported in part by the SCALE-IoT Project granted by the Danish Council for Independent Research under Grant 7026-00042B, and in part by the Aarhus Universitets Forskningsfond Starting Grant Project under Grant AUFF-2017-FLS-7-1. This article was presented in part at the IEEE International Conference on Communications, Dublin, Ireland, 2020. We extend that paper by proposing and studying new compression algorithm for data transmission using side information, studying another transformation function, studying multiple source node's behavior, and considering other real-world data sets. *(Corresponding author: Niloofar Yazdani.)*

The authors are with DIGIT and the Department of Engineering, Aarhus University, 8000 Aarhus, Denmark (e-mail: n.yazdani@eng.au.dk; daniel.lucani@eng.au.dk).

Digital Object Identifier 10.1109/JIOT.2021.3064511

typically require sufficient data to provide a good compression rate that can justify the additional memory and processing costs added to perform this operation. This means buffering samples or measurements for some time before transmission, spending time compressing and decompressing the data before it is useful to the sink node. All these added steps can negatively affect the AoI. Recent schemes have considered simple online compression strategies for the data source using common dictionaries in the sink and source nodes to streamline the compression process [5]. However, the efficiency of these schemes hinges on prior, accurate knowledge of the data source, which is not always possible in IoT systems due to the sheer number and diversity of data sources. More advanced techniques have been proposed, but they rely on collecting and analyzing enough data to guarantee sufficient compression potential. For example, Zhong *et al.* [19] used backlogs to switch between a set of codebooks to adapt to the current data streams and reduce the AoI. However, these are limited to a predefined set of codebooks, which inherently limits the overall potential of the system, and designed for a single source–destination pair, i.e., it cannot exploit correlations across multiple sources.

In [20], the AoI is improved by exploiting the temporal correlation between consecutive samples of the source node, but cannot utilize the correlation across multiple sources. Hribar *et al.* [21] took advantage of the correlated information across different sources to decrease the rate at which sources send status updates to the sink node and hence improved the timeliness of updates. But, this is achieved at the cost of accuracy. AoI is minimized in [22] considering correlation across multiple sources but particularly for sources with common observations, such as redundant sensors for reliable monitoring the same physical phenomenon. The average AoI for a real-time IoT monitoring system with correlated devices is minimized in [23]. However, this work focuses on multiple devices associated with one common physical process, such as smart camera networks with possible overlapping fields of view where the full image can be reconstructed after receiving images from all devices.

Thus, developing and identifying compression techniques that 1) adapts to the data source statistics; 2) exploits correlation across multiple sources; 3) does not affect the information's accuracy; and 4) achieves good compression for small data blocks, are crucial to reduce the AoI in the IoT.

In this article, we reduce the AoI similar to aforementioned works, e.g., [12] and [13], but by using data compression. Namely, we remove the tradeoff between AoI and compression gain. This article focuses on characterizing the AoI of a family of new protocols that allow multisource lossless compression. At the core of these protocols lies the emerging concept of generalized deduplication (GD) [24], [25]. Although originally considered as a generalization of data deduplication (DD) to be applied in large scale distributed storage systems, e.g., in the Cloud, recent works have shown its potential to compress small data packets [26], time-series data [27], [28], and providing unique random access capabilities [29]. In [27], a lossless, multisource data compression approach was proposed to reduce the amount of data transmission that uses an indexing technique inspired by GD. In fact, the technique allows for fast decompression of IoT data samples. We use this protocol as the baseline for our analysis and include a number of variations of it that help reduce the AoI, including: 1) the use of flash and/or RAM at the sensor node to keep preset and dynamically generated dictionaries and 2) the reorganization of data in the transmission to provide an instantly decodable (IDe) variation that decodes every sample as it arrives to the sink.

Our analysis identifies the AoI for these variants for multisource scenarios, by using the single-source case as a key building block for the analysis. We propose upper and lower bounds for the average AoI. We evaluate the AoI and compression performance of our GD variants and compare to transmissions with packet-level compression using LZW and DEFLATE as well as uncompressed transmissions. We also show that our schemes provide up to an order of magnitude gain in compression while also reducing the AoI by a factor of up to 2.3 with respect to state-of-the-art alternatives. Finally, we show that our GD protocols benefit significantly from the use of multiple sources as well as from small preset and dynamically generated dictionaries in the source nodes. The former allows for a system to deliver much better performance over time and exploit both temporal- and spacial-correlations, while the latter kickstarts the AoI and compression benefits for the sources and allows individual sources to adapt to their data to further improve compression.

II. Network Models and Age of Information

We begin by providing notations and definitions that are used throughout the remainder of this article.

A. Single Source Node

We consider having a single source node, a single sink node, and one channel. The source node and the sink node are assumed as the encoder and the decoder, respectively. We consider a new source symbol is generated at the source node, per unit time. The encoder maps every B source symbols into a data chunk. Data chunks are always of fixed size of n bits. The encoder maps each data chunk onto an encoded block. The encoded block has different possible lengths, $L_{\mathcal{P},i}$, where $\mathcal{P}$ shows the policy and i has finite options. $L_{\mathcal{P},i}$ is not constant and might depend on a variable. The detailed description will be discussed in Section III-C.

The queue is $D/G/1$ queuing model as the arrival distribution is deterministic but the service times have a general distribution due to the encoded blocks of different lengths. We assume that the longest encoded block is transmitted before the next encoded block arrives. Section IV-A discusses this with more details. In this article, we assume no packet loss occurs. In real scenarios, some packets may fail to reach their destination due to errors in data transmission or data congestion. This might lead to data buffering where the source node has to keep the previous packets until it makes sure it is available at the sink node.

For our model, we assume when the source node is idle, it transmits a single bit `0`. Otherwise, it transmits a `1` followed by the encoded message to capture the channel. Thus,

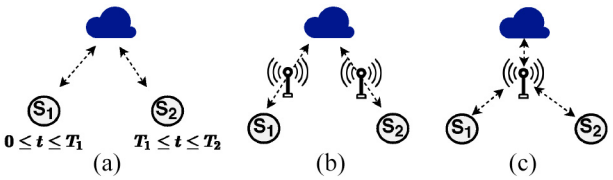

Fig. 1. Multiple source nodes (a) sequential node activity, (b) simultaneous node activity, independent communication channels, and (c) simultaneous node activity, shared communication channel.

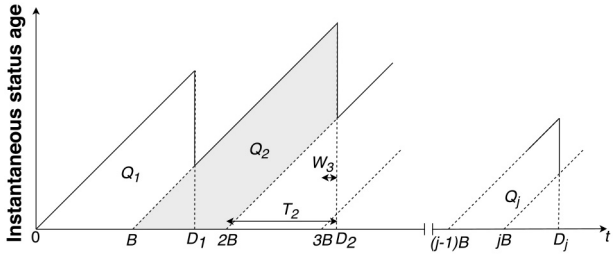

Fig. 2. Example of instantaneous status age of streaming fixed-to-variable length coding.

we consider s as the bit length of the signaling bits which is equal to 1. Note that this assumption can represent more practical settings, where no transmission would be necessary when idle, but where there is still a cost to capture the communication channel. Transmitting the single bit in idle time is not mandatory. This is chosen to keep the model as simple as possible. This is somewhat similar to wired communication protocols that keep a voltage level in the wire when there are no transmissions. This can be replaced with other approaches, such as having a higher layer protocol for distinguishing the beginning of the packet. A constant bit rate of R (bits per unit time), without propagation delay, between the encoder and the decoder is assumed.

B. Multiple Source Nodes

For our model, we consider α as the number of source nodes. The ith source node is depicted with S_i. In this article, we study multiple sources for three different scenarios by extending the model for a single source node.

Sequential Node Activity: In this case, we assume there is only one channel. Each source node starts transmitting its data after the previous source node finishes transmitting all its data as in Fig. 1(a). It is assumed that there is no competition between the nodes for accessing the channel. The time allocated to each of them is known, initially.

Simultaneous Node Activity, Independent Communication Channels: In this case, we assume that each of the source nodes has its own channel for data transmission as depicted in Fig. 1(b). This implies that the number of channels is equal to the number of source nodes and there is no competition for accessing the channels. Consequently, they can transmit whenever a new piece of data is ready for transmission, i.e., they do not have to wait for any other source node to transmit its data. As an example, satellites normally have a private channel for communication with the same sink node.

Simultaneous Node Activity, Shared Communication Channel: In this case, all source nodes share the same channel for data transmission as in Fig. 1(c), i.e., there is only one channel. We assume that source nodes access the channel in order at a specific time, e.g., using a time-division duplexing scheme. We consider the time interval until having the next data chunk ready for transmission is constant and equal for all the source nodes while there is enough time for each of them to send their data before having a new chunk.

Example 1: The mentioned three different scenarios are illustrations of potential events. We use an example of connected vehicles to clarify this.

1) *Sequential Node Activity:* Vehicles may drive at different times (but can be vehicles of the same brand and model as well as the same road) and data from the former travel of the first vehicle can be useful for compressing more effectively the data coming from a second vehicle the next day.
2) *Simultaneous Node Activity, Independent Communication Channels:* Two vehicles could drive at the same time but in different towns without competing for wireless resources.
3) *Simultaneous Node Activity, Shared Communication Channel:* Multiple vehicles in the same region could share the same wireless resources.

A large-scale system would likely have a combination of these various scenarios.

C. Age of Information

In this article, we study the timeliness of our scheme proposed in Section III-B following a similar approach to the streaming source coding analysis in [5].

The generation time and the delivery time of the jth encoded block, Y_j, is represented by jB and D_j, respectively, as shown in Fig. 2. This is due to the fact that each chunk contains B source symbols while each source symbol is generated per unit time as stated in Section II-A. If the decoder's most freshly decoded symbol at time t is generated at time i, the instantaneous status age is $\Delta(t) = t - i$. Subsequently, the time-average status age is

$$\Delta = \lim_{\mathcal{T}\to\infty} \frac{1}{\mathcal{T}} \int_0^{\mathcal{T}} \Delta(t) d(t). \tag{1}$$

In order to calculate the integral, we follow the same technique presented in [3] to calculate the sum of trapezoids areas, Q_j's (shown in Fig. 2). Considering N as the number of encoded blocks

$$\Delta = \lim_{N\to\infty} \frac{1}{BN} \sum_{j=1}^{N} Q_j = \lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} \left(D_j - jB\right) + \frac{B}{2} \tag{2}$$

where $D_j - jB$ measures the time interval between the $j-$th encoded block being ready at the encoder and it being received by the decoder. Let us define W_j and T_{S_j} as the waiting time and service time for the $j-$th encoded block, then, $D_j - jB = W_j + T_{S_j}$. The time that the encoder's FIFO buffer is busy with transmitting other encoded blocks determines the *waiting time*, W_j. The *service time* is the time required for an encoded block of length L_j to be transmitted, i.e., $T_{S_j} = (L_j/R)$.

We are interested in studying the changes in average of status age as the source node transmits more and more encoded blocks. Following the scheme in [3], the time average age over an interval $(0, \mathcal{T})$ is equal to:

$$\Delta_{\mathcal{T}} = \frac{1}{\mathcal{T}} \int_0^{\mathcal{T}} \Delta(t) d(t) \tag{3}$$

where the area of the integral is the concatenation of the trapezoids Q_j's and the last triangular area of width T_N.

Using similar methods as those in [3] and [5] and considering using lossless fixed-to-variable block coding and streaming source coding, the average of status age at the time of receiving the $N-th$ encoded block by the decoder would be equal to

$$\begin{aligned}\Delta_{\text{ave}}(N) &= \frac{1}{BN} \int_0^{BN} \Delta(t) d(t) = \frac{1}{BN} \left[\sum_{j=1}^{N} Q_j + \frac{T_N^2}{2} \right] \\ &= \frac{1}{N} \sum_{j=1}^{N} \left(D_j - jB \right) + \frac{B}{2} + \frac{(D_N - NB)^2}{2BN}. \end{aligned} \tag{4}$$

Considering a stable queue, the third term in (4), i.e., $([(D_N - NB)^2]/2BN)$ will be negligible as N grows, because of the finite value of $D_N - NB$. A queue is stable if the mean number of parts (encoded blocks in this article) in the queue stay bounded [30].

III. Data Transmission Reduction Schemes

A. Background

GD [26] is a lossless compression scheme introduced as an alternative to DD. While DD only finds equal data chunks, GD compresses data by finding both equal and similar data chunks. GD uses a transformation function to convert a sequence of equal-sized data chunks, $(C_1, C_2, \ldots, C_i, \ldots)$ into a set of bases and deviations $(b_1, b_2, \ldots, b_i, \ldots)$ and $(d_1, d_2, \ldots, d_i, \ldots)$, where C_i constitute the ith data chunk and (b_i, d_i) are the basis-deviation pair corresponding to that chunk. The pair (b_i, d_i) is unique to the content of C_i. The system is designed so that $|b_i| >> |d_i|$. Each basis, b_i, is given a fingerprint, $f(b_i)$, e.g., using a hash function, which is much smaller than b_i or C_i. The purpose is to map several different chunks to the same basis while deviations represents their differences. To do so, rather than saving each chunk as a pair of (b_i, d_i), it can be saved as a pair of $(f(b_i), d_i)$ while the bases are deduplicated.

In our context, this transformation process takes place at the source node, e.g., a sensor node, with the purpose of sending the basis only if not available at the sink node (e.g., Edge device, Cloud) to decrease the amount of transmitted data. More specifically, each source node generates data chunks to be sent to the sink node. By applying GD, each C_i is mapped onto a pair (b_i, d_i) using a transformation function, e.g., using an error-correcting code (ECC) [31] or permute-and-truncate transformation function [28]. The source node will first send the basis' fingerprint and the deviation for each piece of data. If the sink node has the basis that matches the transmitted fingerprint, the source does not need to transmit any further data. Otherwise, the sink requests for the basis to be transmitted.

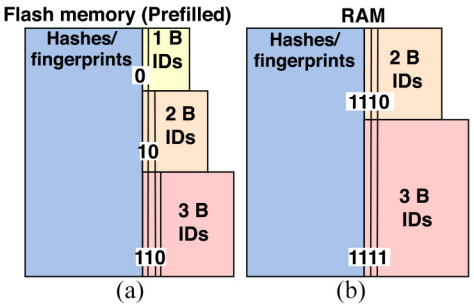

Fig. 3. (a) Limited flash memory to keep a preset dictionary. (b) Limited RAM.

Each source node benefits from all the fingerprints at the sink node, whether it is generated by that source node or not [27]. This is due to the fact that all the source nodes share the same function for generating the fingerprints. Accordingly, the same basis will be mapped to the same fingerprint in all the source nodes. We define n, k, d, and h as the chunk, basis, deviation and fingerprint bit lengths, respectively. *DD* is a special case of *GD*, where there is no deviation.

Although the original concept provided in [27] requires minimal resources from the source node, which is a good feature for sensor devices, it can be extended with a number of optimizations that use memory resources in a controlled fashion, i.e., using some additional memory or storage. In the following, we describe all these options in more detail.

B. Compression Algorithms

1) Compression Without Side Information: The data transmission scheme described in Section III-A is a case that requires no additional memory or state to be kept at the source node to track the available bases at the sink node. By default, the system transmits the deviation and the fingerprint without a prior knowledge of the fingerprint's availability at the sink node. The sink node is responsible for requesting the basis, if it does not have it. The source node only needs to maintain in memory the basis and deviation until receiving confirmation from the sink node that the transmission was successful.

2) Compression With Side Information: We consider the case of compression with side information, when the source node takes advantage of an internal flash memory and/or RAM to reduce the transmission cost. To achieve it, this additional memory is exploited to store a preset dictionary or develop a dictionary on-the-fly (or both) that is shared with the sink. We draw inspiration from [32] to define the dictionary structure.

Preset Dictionary Stored in Flash Memory: The source node can maintain a static preset dictionary in flash memory with the most common fingerprints as in Fig. 3(a). This dictionary is common between the source node and the sink node. The sink node also has the bases assigned to these fingerprints. Each fingerprint in the preset dictionary is given a compressed ID. For example, each of the first 128 (2^7) top most repetitive fingerprints are given an ID of length 1 B starting with a 0 prefix. The prefixes are uniquely assigned showing the length of the ID and whether the compressed ID belongs to the preset dictionary or not. The transmission cost is reduced as there is

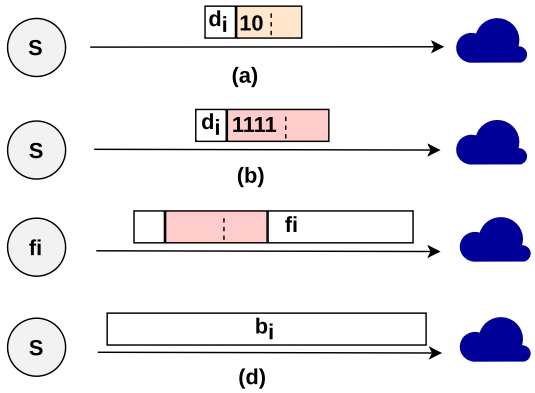

Fig. 4. Example of (a) fingerprint available in the flash memory, (b) fingerprint available in RAM, (c) new fingerprint which is added to the RAM, and (d) basis transmission if the new fingerprint is not available in the sink node.

no need to send the associated bases or a full fingerprint, but only the compressed IDs.

Although the dictionary can be developed for all sources of a given type, it can also be created and updated specifically for each source node based on past data. The latter improves the probability of finding bases in the dictionary. This results in a reduction of the transmission cost and the freshness of information. The number of fingerprints in the dictionary is determined by the flash memory size and the fingerprint length.

Dynamic Dictionary Stored in RAM: The sink node can use its extra RAM to store as many as possible of the observed fingerprints. Each of these fingerprints can be assigned a compressed ID shared with the sink node during the first transmission of the fingerprint. The sink node keeps a map of the fingerprints and the compressed IDs generated on-the-fly in RAM as in Fig. 3(b). This dynamic dictionary is initially empty. After the first chunk, C_1, is transformed into (b_1, d_1), the source node looks up the basis' fingerprint in the dynamic dictionary. If it is available in the dictionary, the source can send the associated compressed ID and the deviation and await for confirmation of reception from the sink. Otherwise, the source adds the fingerprint to the dynamic dictionary in RAM and assigns a compressed basis ID to it.

The IDs are assigned in order. The first ID is `1110000000000000` where the prefix `1110` indicates that the ID belongs to the dynamic dictionary and is 2 B. The next ID is `1110000000000001`. Although fingerprints are common across all the source nodes, compressed IDs differ. In fact, the same compressed ID generated dynamically in two different nodes are most likely mapped to different bases. This adds some complexity in the sink node, which needs to keep the assignment map for each individual source.

After assigning a compressed ID to the new fingerprint (i.e., not present in the dictionary), the source node sends the fingerprint, compressed ID, and the deviation to the sink node to: 1) signal the assignment of the new compressed ID to the new fingerprint to the sink node; 2) send the information about the chunk to the sink node; and 3) check availability of the new fingerprint in the sink node. If the fingerprint is available at the sink node, the sink node sends back an acknowledgment and saves the new assigned compressed ID to it for the specific source node. Otherwise, the sink saves the fingerprint and the compressed ID and sends a request for the corresponding basis. After receiving the basis, the sink acknowledges this to the source.

Similar to the preset dictionary, dynamic dictionary can keep a limited number of fingerprints. In contrast to the preset dictionary, a source node replaces older, less frequently used fingerprints in the dynamic dictionary with newer ones. In this case, the source node reassigns the compressed IDs. For instance, if a rarely used fingerprint is assigned the compressed ID `1110000000011110`, this compressed ID will be reassigned to a new fingerprint hereafter. To signal this, the source sends the new fingerprint and the assigned compressed ID to the sink node alongside the deviation.

For determining the older, less frequently used fingerprints, we use a logical queue. If a fingerprint of a basis matches is in the logical queue, the fingerprint and the associated ID is moved to the end of the queue. The first element of the logical queue is the least frequently used one. When the queue is full or the last compressed ID is used, that first element will be removed.

Combination of Dynamic and Preset Dictionaries: If the source has both the dynamic and preset dictionaries, it follows a similar procedure as the case of a dynamic dictionary. The main difference is that the source checks the preset dictionary first prior to checking the dynamic dictionary. If available, it will transmit the compressed ID in the preset dictionary and the deviation. Otherwise, the procedure is the same as the dynamic dictionary. Bases in the preset dictionary are not added to the dynamic dictionary. Fig. 4 shows an example of different transmissions types with a source node with both preset and dynamic dictionaries.

Compression with side information can help with reducing memory buffering. If the fingerprint is available at the flash memory or RAM, the source node would be aware that the basis is available at the sink node. Thus, it does not need to keep the basis, which is the largest part of the data. Our scheme also opens the door to sending multiple compressed IDs and/or fingerprints in the buffer with a single transmission, which can further improve the efficiency and freshness of the information. These topics are out of scope, but will be studied in future work.

C. Encoded Block Size

As mentioned in Section II-A, $L_{\mathcal{P},i}$ represents the length of the encoded block. The encoded block has different possible lengths considering our compression and transmission protocol.

For compression without side information, if there is a match to a previously sent fingerprint, the output is of length $L_{wo,1} = s + d + h$. Otherwise, the basis should be transmitted as well and the output is of length, $L_{wo,2} = s + d + h + k$.

For compression with side information, we define I as the bit length of the compressed ID where $I \in \{8, 16, 24\}$. If there is a match to one of the available fingerprints in the preset or dynamic dictionary, the output is of length, $L_{w,1}(I) = s+d+I$. If the fingerprint has no matches in the source dictionary, but

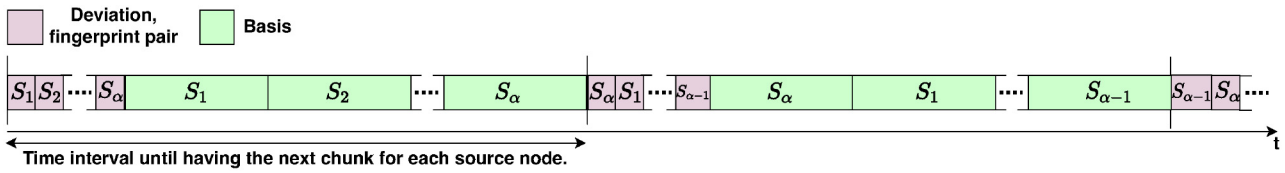

Fig. 5. Multiple source nodes transmitting data using a shared channel.

TABLE I
GD'S ENCODING PROCESSING SPEED

chunk length [B]	PC [Mbps]	Raspberry Pi 3 [Mbps]
8	720	19.52
16	880	30.24
128	3360	170.8
512	8560	350.4

the sink node has previously seen the fingerprint, then the output is of length, $L_{w,2}(I) = s + d + h + I$ as we also need to send the fingerprint. Finally, if the sink has not seen the fingerprint before, the output is of length, $L_{w,3}(I) = s + d + h + I + k$ as the basis, the fingerprint and the compressed ID must be sent.

D. Practical Considerations for Multiple Source Nodes

The data transmission schemes described in Sections III-A and III-B apply directly to having a single source node as well as multiple source nodes for *sequential node activity* and *simultaneous node activity, independent communication channels* as either there is one channel per source node or the nodes are active sequentially. However, for *simultaneous node activity, shared communication channel* some transmission orders are required to be considered as well.

As shown in Fig. 5, each of the source nodes starting from the first one, S_1, to the last one, S_α, send their deviation-fingerprint pair for their new chunk. If we used techniques with side information, we would use compressed IDs when possible. The order of the nodes shifts circularly by one every round. For instance, in the second round, S_1 is the second node who accesses the channel. In the third round, S_1 will be the third node. This helps to have a fair waiting time to access the channel for each of the nodes.

The source nodes receive an acknowledgment or a signal to send its corresponding basis. The latter also specifies the node's turn to send the basis. If all the source nodes need to send their bases, the order of the bases transmission would be exactly similar to the order of deviation-fingerprint pair transmission. If a source node's basis is already available at the sink node, its turn would be given to the next source node. Therefore, basis request message, in channel sharing, maintains a few extra bits to specify source node's turn.

As the transmission time for each channel is predetermined, there is no need for extra transmission of one set bit at the beginning of the encoded message to capture the channel, i.e., $s = 0$.

Note that in *sequential node activity*, the first source node cannot take advantage of the data transmitted by the second source node. However, the second source node can use all the fingerprints common between itself and the first source node. Similarly, source node S_j can take advantage of all the data transmitted by all the previous source nodes, i.e., $S_{j-1}, S_{j-2}, \ldots, S_1$.

IV. SINGLE SOURCE NODE AGE OF INFORMATION ANALYSIS: COMPRESSION WITH SIDE INFORMATION

Following a similar approach as in Section II-C, we study the timeliness of our scheme for compression with side information for a single source node. In Section V, we will study the timeliness of our scheme for compression without side information for a single source node. Section VI will generalize this analysis to the case of multiple source nodes.

Applying GD is the most time consuming part of our compression schemes. At the same time, GD uses very simple operations, such as XOR or bit shifting. Thus, it provides a very high processing speed even when considering resource limited devices, such as Raspberry Pis. Table I reports GD's encoding processing speed for a range of chunk sizes on PC [25] and on Raspberry Pi 3 [27]. The results for PC are calculated on a Core i7 3770 at 4.08 GHz. The results on the PC and Raspberry Pi are calculated considering a single thread and without Single Instruction Multiple Data. As it is depicted in the Table I, the processing speed increases for larger chunks. Even considering a small chunk of 8 B, the encoding speed is as high as 19.5 Mb/s on Raspberry Pi 3 which is several orders higher than the expected data generation rate at the edge nodes in IoT applications (e.g., 1 Mb/s for controller area network (CAN) data [33]). Accordingly, the time required for applying the proposed compression algorithms will not be considered in the analysis.

There is another factor affecting the waiting time W_j for GD aside from the encoder's FIFO butter being busy with transmitting other encoded blocks, $W_{b,j}$. Note that $W_{b,j} = 0$ if the source node finishes transmitting each encoded block before the next encoded block arrives. The second factor affecting the waiting time comes from the time interval the source node waits for the sink node to send the request for the basis because the fingerprint is not available at the sink node, $W_{r,j}$. Note that $W_{r,j} = 0$ if the fingerprint is available at the sink node. Otherwise, $W_{r,j}$ is determined based on the length of the request message, $L_{\text{req.}}$ and R, i.e., $W_{r,j} = W_r = (L_{\text{req.}}/R)$, where W_r is a constant in our system model.

$\Delta_{\text{ave}}(N)$ follows the same formula as in (4) with:

$$D_j - jB = \begin{cases} \frac{L_{w,1}(I)}{R}, & \text{if match at the source node} \\ \frac{L_{w,2}(I)}{R}, & \text{else if match at the sink node} \\ \frac{L_{w,3}(I)}{R} + W_r, & \text{else (corresponds to no match).} \end{cases} \tag{5}$$

A. Timely Throughput Condition

In this work, we assume R is high enough that the longest encoded block is transmitted before the next encoded block arrives, which takes B time units. This includes the required time to receiving the final acknowledgment, $T_{\text{Ack}} = (L_{\text{Ack}}/R)$, where L_{Ack} is the bit length of the acknowledgment. The longest encoded block is of length $L_{w,3}(24)$ bits. Thus

$$\frac{L_{w,3}(24)}{R} + W_r + T_{\text{Ack}} < B \Rightarrow R > \frac{L_{w,3}(24) + L_{\text{req.}} + L_{\text{Ack}}}{B}. \quad (6)$$

This condition is more stringent compared to having a stable buffer (stable queue). The reason of putting this restriction is due to the fact that our knowledge is limited to the worst case transmission. The value of R can be adapted considering having limited RAM and flash memory or based on the data set. We will study the adaptive R selection in future work.

B. Instant Decodability

It is assumed in [5] that all the source symbols within an encoded block are decoded together once the sink node has received all the encoded block. In [1], the concept of an IDe strategy was proposed. This idea improves the AoI by allowing measurements to become available at the receiver even if the entire basis is still in transit. This is achieved by a judicious organization of the data and by sending the deviation of a block before the basis. No additional complexity is added to the system. Although [1] defined IDe when using Hamming codes as the transformation function, an IDe can be developed for other transformation functions. Fig. 6 shows an example of IDe for a case of having four source symbols X_1, X_2, X_3, and X_4 within a chunk. As shown in Fig. 6(a), GD maps the chunk to a basis and a deviation where the basis has the information about X_1, X_3, and X_4 and half of the bits of X_2, while deviation contains the information for the rest of X_2's bits. The source node sends the deviation bits first and then the bases as in Fig. 6(b). Consequently, the receiver can decode the data symbol by symbol as it receives the encoded block [Fig. 6(c)]. This is due to the fact that the sink node is already aware of the bits that need to be modified or added given the transmitted deviation information.

After receiving B source symbols, a new chunk is ready which is encoded into an encoded block ready for transmission [Fig. 6(c)]. Using IDe, the decoder does not decode all the source symbols of an encoded block, Y_j, at time D_j, as in [5]. Instead, it decodes the symbol one by one, as shown with solid purple line, with a small delay introduced by the deviation and compressed basis ID transmission at the beginning of the encoded block transmission.

The ith source symbol is decoded at time D'_i. Rather than calculating the gray areas, we should calculate the blue areas, Q'_i's, to calculate the $\Delta_{\text{ave}}(N)$. The improvement in average AoI by using IDe for GD is shown by the purple area covered by stars in Fig. 6(c). If the basis is available at the sink node, all the symbols of the encoded block will be decoded at the same time as shown in Fig. 6(c) for the third transmission (endpoint of Q'_{12} and Q'_{10} are equal).

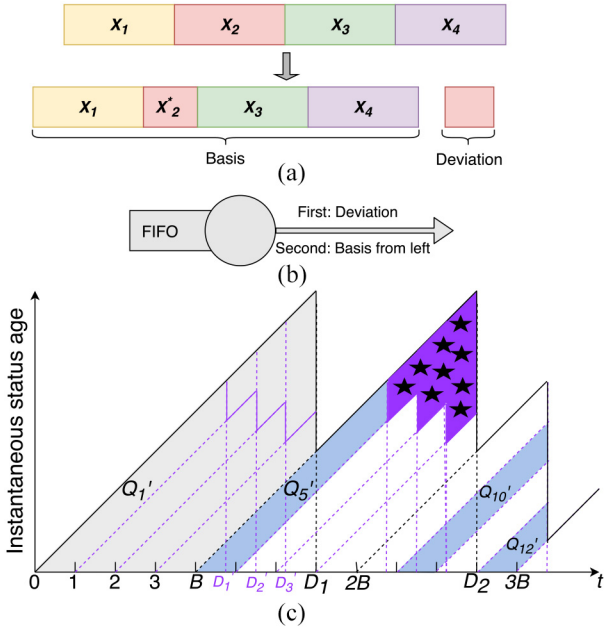

Fig. 6. Example of the *IDe scheme* to reduce the AoI for $B = 4$. (a) chunk transformed into a basis and a deviation, (b) order of transmission, and (c) instantaneous status age.

Assuming $N' = BN$, $B' = 1$ and by following a similar method as in [3] to calculate the sum of trapezoids areas Q'_i

$$\Delta_{\text{ave}}(N) = \frac{1}{N'}\left[\sum_{i=1}^{N'} Q'_i + \frac{\left(D'_N - N'\right)^2}{2}\right] = \frac{1}{N'}\sum_{i=1}^{N'}\left(D'_i - i\right) + \frac{1}{2} + \frac{\left(D'_{N'} - N'\right)^2}{2N'}. \quad (7)$$

We define $\mathcal{X}_i = i$ mode B, and $M = n/(B.R)$ as the time interval between arriving of the source symbols of an encoded block at the decoder. Then

$$D'_i - i = \begin{cases} \frac{L_{w,1}(I)}{R}, & \text{if match at the source node and } \mathcal{X}_i = 0 \\ (B - \mathcal{X}_i) + \frac{L_{w,1}(I)}{R}, & \text{else if match at the source node and } \mathcal{X}_i \neq 0 \\ \frac{L_{w,2}(I)}{R}, & \text{else if match at the sink node and } \mathcal{X}_i = 0 \\ (B - \mathcal{X}_i) + \frac{L_{w,2}(I)}{R}, & \text{else if match at the sink node and } \mathcal{X}_i \neq 0 \\ \frac{L_{w,3}(I)}{R} + W_r, & \text{else if no match and } \mathcal{X}_i = 0 \\ (B - \mathcal{X}_i) + \frac{L_{w,3}(I)-k}{R} + \mathcal{X}_i.M + W_r & \text{else (no match and } \mathcal{X}_i \neq 0). \end{cases} \quad (8)$$

C. Performance Bounds

Considering the definition of $\Delta_{\text{ave}}(N)$ and the structure of our problem, clear upper and lower bounds for $\Delta_{\text{ave}}(N)$ can be calculated. The lower bound can be calculated considering the case where every fingerprint matches an already existing fingerprint at the flash memory or RAM with lowest possible associated compressed ID of length 8 bits, i.e., the encoded block is of length $L_{w,1}(8)$. In this case, no basis transmission is required. This happens when all the chunks are mapped to

a small set of bases. According to (4) and (5) and based on the fact that the third item in (4) is temporary for a stable buffer (because the time interval between the delivery time of the last generated encoded block, D_N and its generation time, NB, has a limit value while N is increasing by having more encoded blocks.)

$$\Delta_{\text{ave}}(N) \geq \Delta_{\min} = \frac{L_{w,1}(8)}{R} + \frac{B}{2}. \tag{9}$$

The upper bound is calculated considering the case where no fingerprint matches an existing fingerprint at the sink node and the associated compressed IDs are the longest possible value, i.e., 24 bits. In this case every basis must be transmitted and length of the encoded block is equal to $L_{w,3}(24)$. Accordingly

$$\Delta_{\text{ave}}(N) \leq \Delta_{\text{peak}} = \frac{L_{w,3}(24)}{R} + W_r + \frac{B}{2} + \frac{\left(\frac{L_{w,3}(24)}{R} + W_r\right)^2}{2B} \tag{10}$$

where the last term in (4) is evaluated at $N = 1$.

For a large enough N

$$\lim_{N\to\infty} \Delta_{\text{ave}}(N) \leq \Delta_{\max} = \frac{L_{w,3}(24)}{R} + W_r + \frac{B}{2}. \tag{11}$$

Using a similar strategy, the upper bound for IDe can be found. $D'_i - i$ is periodic over the period of B. Considering (7) for a steady buffer when the last term is removed

$$\Delta_{\max,\text{IDe}} = \frac{1}{B}\sum_{i=1}^{B}\left(D'_i - i\right) + \frac{1}{2} \tag{12}$$

where based on (8)

$$D'_i - i = \begin{cases} \frac{L_{w,3}(24)}{R} + W_r, & \text{for } i = B \\ (B-i) + \frac{L_{w,3}(24)-k}{R} + i.M + W_r, & \text{for } i < B. \end{cases} \tag{13}$$

Example 2: If $B = 4$, $d = 9$, $n = 256$, $L_{\text{req}} = 16$, $R = 90$ [bits]/[Unit Time] with a 4 B fingerprint, the upper bound is equal to 5.65 and 4.64 for the standard GD and GD with IDe strategy, respectively. In this example, IDe has improved the upper bound by 17.9%.

V. Single Source Node Age of Information Analysis: Compression Without Side Information

Following a similar approach as in Section IV, we study the timeliness of our scheme for a single source node without side information. This was studied extensively in [1]. In the following, we mention the differences with respect to the model in Section IV.

$\Delta_{\text{ave}}(N)$ follows the same formula as in (4) with:

$$D_j - jB = \begin{cases} \frac{L_{wo,1}}{R}, & \text{if match} \\ \frac{L_{wo,2}}{R} + W_r, & \text{else (no match).} \end{cases} \tag{14}$$

This is because the encoded block for communication without side information is $L_{wo,1}$ and $L_{wo,2}$ bits long in case of having or not having a match, respectively. And, if the fingerprint does not match to an already available fingerprint, the source node needs to wait for the basis request as long as W_r.

Considering the same argument as in Section IV-A, we consider the timely throughput R to be

$$\frac{L_{wo,2}}{R} + W_r + T_{\text{Ack}} < B \Rightarrow R > \frac{L_{wo,2} + L_{\text{req}} + L_{\text{Ack}}}{B} \tag{15}$$

so that the longest encoded block is transmitted before the next encoded block arrives.

By using the IDe strategy, $\Delta_{\text{ave}}(N)$ can be calculated using (7) with

$$D'_i - i = \begin{cases} \frac{L_{wo,1}}{R}, & \text{if match and } \mathcal{X}_i = 0 \\ (B - \mathcal{X}_i) + \frac{L_{wo,1}}{R}, & \text{else if match and } \mathcal{X}_i \neq 0 \\ \frac{L_{wo,2}}{R} + W_r, & \text{else if no match and } \mathcal{X}_i = 0 \\ (B - \mathcal{X}_i) + \frac{L_{wo,2}-k}{R} + \mathcal{X}_i.M + W_r & \\ \quad \text{else (no match and } \mathcal{X}_i \neq 0). & \end{cases} \tag{16}$$

The lower bound for $\Delta_{\text{ave}}(N)$ can be calculated by considering that every fingerprint matches to an already existing fingerprint at the sink node, that is,

$$\Delta_{\text{ave}}(N) \geq \Delta_{\min} = \frac{L_{wo,1}}{R} + \frac{B}{2} \tag{17}$$

while the upper bound is found by considering having no matches for $N = 1$

$$\Delta_{\text{ave}}(N) \leq \Delta_{\text{peak}} = \frac{L_{wo,2}}{R} + W_r + \frac{B}{2} + \frac{\left(\frac{L_{wo,2}}{R} + W_r\right)^2}{2B}. \tag{18}$$

If N goes to infinity

$$\lim_{N\to\infty} \Delta_{\text{ave}}(N) \leq \Delta_{\max} = \frac{L_{wo,2}}{R} + W_r + \frac{B}{2}. \tag{19}$$

The upper bound using IDe is

$$\Delta_{\max,\text{IDe}} = \frac{1}{B}\sum_{i=1}^{B}\left(D'_i - i\right) + \frac{1}{2} \tag{20}$$

where

$$D'_i - i = \begin{cases} \frac{L_{wo,2}}{R} + W_r, & \text{for } i = B \\ (B-i) + \frac{L_{wo,2}-k}{R} + i.M + W_r, & \text{for } i < B. \end{cases} \tag{21}$$

There is a clear relationship between the upper bounds and lower bounds for compression with or without side information. Compression without side information has the highest lower bound and the lowest upper bound compared to compression with side information because

$$\Delta_{\min\text{without side information}} - \Delta_{\min\text{with side information}} = \frac{h-8}{R} \tag{22}$$

$$\Delta_{\max\text{with side information}} - \Delta_{\max\text{without side information}} = \frac{24}{R}. \tag{23}$$

These results show that however compression with side information has a potential to effectively reduce the transmission cost, there is a possibility of increasing the transmission cost if implemented, inappropriately, e.g., for a data set with limited number of matches to the same fingerprint.

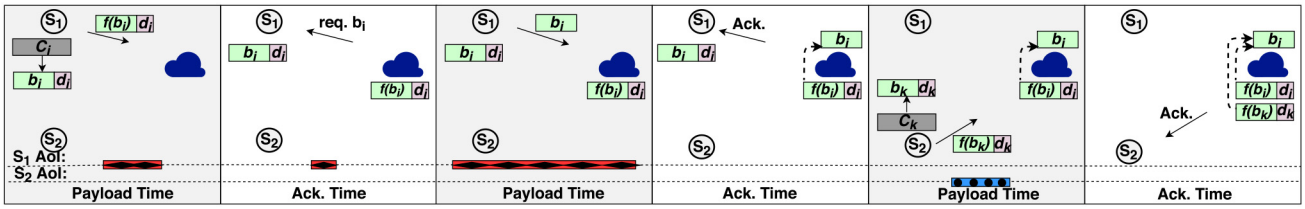

Fig. 7. Example of AoI improvement by considering multiple source nodes. ($b_i = b_k$).

VI. Multiple Source Node Age of Information Analysis

Although the system supports the case of a single source node, the sink node can build knowledge across multiple sources, i.e., shared knowledge. By having more source nodes, the sink node can build up its knowledge faster and naturally improve the compression potential for other active nodes as well as reducing the delay to convey data to the sink. In other words, having multiple active source nodes can improve both AoI and compression.

We consider having different time durations as payload and acknowledgment times. The payload time is the required time for transmitting deviation-fingerprint pair (deviation-compressed ID pair or deviation-compressed ID-fingerprint trial in case of compression with side information) or transmitting the basis from the source node to the sink node. The payload time for the former is shorter than the payload time for transmitting the basis. This is because the basis is the large part of the data. The acknowledgment time is the required time to transmit an acknowledgment or a basis request from the sink node to a source node. The acknowledgment time is shorter than the payload time.

Fig. 7 shows a simple example of how AoI can be improved by having more than one source node. We assume we have no side information and fingerprints are generated with a function common to all nodes in the system. We also assume that source node S_1 has a new chunk. The associated basis to this chunk is not available at the sink node. Thus, it takes two payload time slots and two acknowledge time slots for S_1 to transmit the data to the sink node. Fig. 7 also shows the AoI for this transmission which is equal to the aggregation of the red bars covered with diamonds. The second acknowledgment time does not affect the AoI of the current transmission as the sink has successfully received all required information and can operate on it.

Afterwards, source node S_2 will transmit a different chunk which is mapped to the same basis ($b_k = b_i$). Since the basis is available at the sink node, it takes only 1 payload time for S_2 to transmit its data to the sink node which is shown with the blue bar covered with circles. The time to receive the acknowledgment is not counted in AoI for this transmission due to the same reason as for S_1's transmission. Thus, the contribution of S_1 has greatly improved the freshness of information for S_2.

The AoI analysis for *sequential node activity* and *simultaneous node activity, independent communication channels* are exactly equal to the AoI analysis for a single source node for compression with or without side information. This is due to the fact that none of the source nodes has to wait for another source node finishing its transmission. However, better results for AoI is expected due to a higher probability of matching to an existing fingerprint. In *sequential node activity*, the second source node will take advantage of all the bases already available at the sink node, transmitted by the first source node. This increases the matching probability and reduces the AoI. More precisely, $D_j - jB$ will not include the basis transmission or W_r for many cases where we have a match. Unfortunately, the first source node will not take advantage of any data transmitted by the second source node since the second node's data is not available at the sink node at that time. In *simultaneous node activity, independent communication channels*, each source node can use all the previous bases already transmitted by other source nodes. This increases the probability of matching for all the source nodes. The timely throughput condition for these two cases is the same as the case of a single source node.

In *simultaneous node activity, shared communication channel*, source nodes are required to wait to access the channel. This waiting time should be considered in the AoI analysis. For each source node, $\Delta_{\text{ave}}(N)$ follows the behavior modeled in (4). However, the aforementioned extra waiting time should be considered in $D_j - jB$. We define $W_{m,j}$ as the time interval a source node waits to access the channel for the jth encoded block. We can then redefine W_j as $W_j = W_{b,j} + W_{r,j} + W_{m,j}$ to include this new parameter in the analysis presented in Section IV.

As stated previously, in *simultaneous node activity, shared communication channel*, $s = 0$ and W_r is greater compared to having a single source node. For simplicity, we study a case of having two source nodes and study the AoI for the first source node. The analysis for the second source node is quite similar and the general case of α nodes follows the same approach. Fig. 8 depicts 8 different transmission order models. In Fig. 8, any transmission by S_2 which does not affect the AoI for S_1 is faded.

The first transmission order stands for a case where the fingerprint from the first and second source nodes are available at the sink node (no basis transmission is required). These transmission order models are categorized into three different groups. In group 1, S_1 does not have to wait for S_2 to access the channel. In the second group, S_1 only waits for S_2 to transmit its fingerprint-deviation pair and receive the acknowledgment. In the third group, S_1 has to wait for S_2 to transmit the basis plus the fingerprint-deviation pair and

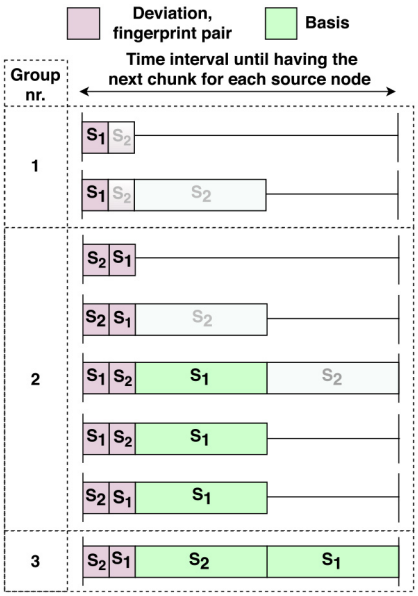

Fig. 8. Eight different transmission order models for having two source nodes.

to receive the basis request and acknowledgment. Thus, if S_2 compresses data without side information

$$W_{m,j} = \begin{cases} 0, & \text{for } j \in \text{Group 1} \\ \frac{L_{wo,1}}{R} + T_{\text{Ack}}, & \text{for } j \in \text{Group 2} \\ \frac{L_{wo,2}}{R} + W_r + T_{\text{Ack}}, & \text{for } j \in \text{Group 3.} \end{cases} \quad (24)$$

On the other hand, if S_2 compresses with side information

$$W_{m,j} = \begin{cases} 0, & \text{for } j \in \text{Group 1} \\ \frac{L_{w,2}(24)}{R} + T_{\text{Ack}}, & \text{for } j \in \text{Group 2} \\ \frac{L_{w,3}(24)}{R} + W_r + T_{\text{Ack}}, & \text{for } j \in \text{Group 3.} \end{cases} \quad (25)$$

In (25), we have considered the longest possible lengths for each case, i.e., $L_{w,2}(24)$ and $L_{w,3}(24)$. It is because, we assumed the time given to each node supports the worst case scenario. If the source node has lower bits to transmit, e.g., for $I = 8$ or when the basis transmission is not required, the extra time could be used for transmitting additional information or to save energy by setting one or more devices in idle or low power modes.

In general, for having α source nodes, the number of groups, i.e., different values $W_{m,j}$ can take, is equal to $1+2^{\alpha-1}$. This is because, each source node has to wait either for no other nodes ($W_{m,j} = 0$) or for each of the other source nodes ($\alpha - 1$ source nodes) for either their fingerprint-deviation pair transmission or fingerprint-deviation-basis trial transmission. The minimum R for multiple source nodes is equal to the minimum R for a single source node multiplied by the number of source nodes.

VII. Performance Evaluation

A. Compression Schemes

We compare the AoI and the compression gain of our GD data transmission schemes to an equivalent DD scheme as well as protocols that compress each data packet using standard lossless compression algorithms. We use the case of raw data transmissions (i.e., transmission without compression) as a baseline for our comparisons.

Raw: Data is transmitted without applying any compression technique.

DEFLATE: DEFLATE [34] is a standard lossless data compression technique which employs a combination of LZ77 [35] and Huffman [36]. We apply DEFLATE on the payload of each data packet prior to transmission.

Lempel–Ziv–Welch (LZW): LZW [37] is a well-known lossless compression scheme which is developed based on LZ77 [35] and LZ78 [38]. We apply it in the same way as DEFLATE, i.e., on the payload of each transmitted packet.

Data Deduplication: DD can be seen as a specific case of GD where the whole data chunk is considered as the basis, i.e., there is no deviation.

Generalized Deduplication: GD considers our proposed schemes described in Section III-B. We considered two transformation functions. First, a transformation based on Hamming codes as in [27]. The data chunk is considered as the *codeword*. By applying the decoding function of a Hamming ECC, the recovered *message* constitutes the basis [27]. The deviation conveys the information about the difference between the error-free codeword created by encoding the *message* and the codeword. This can be captured by the *syndrome* corresponding to the codeword. Hamming supports codewords of length $2^m - 1$, where m is the number of parity bits. Since data is typically organized in bytes, we consider the extra bit at the end of the data chunk (or codeword) as part of the deviation. We apply the Hamming decoding algorithm to the rest of the data chunk to compute the basis and the deviation.

Second, we consider the permute-and-truncate transformation function introduced in [28]. This transformation function splits the data chunk in two parts to form the basis and deviation. The transformation typically assigns the bits with highest variations (or with lower correlations to other bits) to the deviation and the rest of the bits to the basis. As an example, a source node generating chunks containing a time stamp (changing with every sample) and sensor measurements (changing slowly), the system would assign some of the least significant bits of the time stamp to the deviation. The advantage is that the probability of finding a match between basis becomes much higher if the high variability bits are "removed" from the bases.

B. Data Sets

We evaluate our techniques using two real world data sets.

Temperature Data Set: The Intel Berkeley Research Laboratory contains 2.3 million readings from more than 50 sensor nodes [39]. We consider temperature data with a range of 15.0 to 32.0 Celsius of all sensors with accuracy of one digit after the decimal point and represented in 8 bits.

We consider the differential samples where the first sample is kept untouched and the next ones are replaced with the difference between the current sample and the previous one. This is applied as a preprocessing step for all schemes. We assume each chunk contains 16 samples each of length 1 B,

i.e., $B = 16$ for all compression schemes. For this data set, we use a Hamming code as the transformation function for GD.

Controller Area Network Data Sets: We collected CAN bus data from three vehicles: 1) Ford Fiesta Gasoline 2015; 2) Hyundai Ioniq Plugin Hybrid 2018; and 3) Toyota CHR 2018 using a CAN bus data logger, *CANedge2* [40] from CSS Electronics. We connected the device to the OBD-II port of the vehicles to log the messages on CAN bus in MDF4 [41] file format. We collect time series data with a bit rate of 500 000 b/s of in-vehicle sensor information, such as fuel consumption and vehicle speed. We installed the most up to date firmware at the time which was firmware version 00.07 [40]. Drivers freely chose their routes inside or outside the city. A total of 38, 33, and 27 MDF4 files were collected for Ford, Hyundai and Toyota data sets, respectively. The Ford data set's size is 1.3 GB while the Hyundai and Toyota data sets are of size 31 MB and 124 MB, respectively. This is related to a combination of duration of travel, but also of number of reported parameters for the different vehicles. MDF4 files maintain a collection of *records*, where each *record* keeps all the data for one timestamp recorded from CAN bus. Each record is 27 B long maintaining a timestamp alongside the other information.

We compute the differential values for the timestamps, where the timestamp for the first record is kept unchanged and the timestamp in the following records are replaced with the differential values between the current timestamp and the previous one. Each data chunk contains 1 record, i.e., $B = 1$ and is of length 27 B for all the compression schemes. We use a variation of the permute-and-truncate transformation, where we consider the last 2 B of the differential timestamp as the deviation and the rest of the chunk as the basis.

C. Other Considerations

Definition 1: We calculate the compression gain for N data chunks (N encoded blocks) and policy $\mathcal{P}$, $G_{\mathcal{P}}(N)$, as the size of the uncompressed data, $N \cdot n$, divided by the size of the compressed data, i.e., a higher compression gain means more compression. The size of the compressed data is the addition of the size of the transmitted deviations, $N \cdot d$, plus the size of the transmitted bases, $\mathcal{B} \cdot k$, where $\mathcal{B}$ is the number of unique bases whose fingerprints are not available at the sink node, plus the size of the transmitted fingerprints and compressed IDs, $F_{\mathcal{P}}$. $F_{\mathcal{P}}$ for compression without side information and with side information are equal to $F_{wo} = N \cdot h$ and $F_w = I_{Fl,8} \cdot 8 + (I_{Fl,16} + I_{RA,16}) \cdot 16 + (I_{Fl,24} + I_{RA,24}) \cdot 24 + H \cdot h$, respectively, where H is the number of observed fingerprints not available at the flash memory or RAM and $F_{x,I}$ represents the number of matches to the flash memory ($x = Fl$) or RAM ($x = RA$) for compressed IDs of length I bits. Accordingly

$$G_{\mathcal{P}}(N) = \frac{N \cdot n}{N \cdot d + \mathcal{B} \cdot k + F_{\mathcal{P}}}. \tag{26}$$

We use python's zlib and lzw modules for DEFLATE and LZW, respectively. The preset dictionary is tailored for GD and DD for each data set as described in Section III-B2. For each case and each file, we provide a list of fingerprints that are organized based on how many times they are repeated.

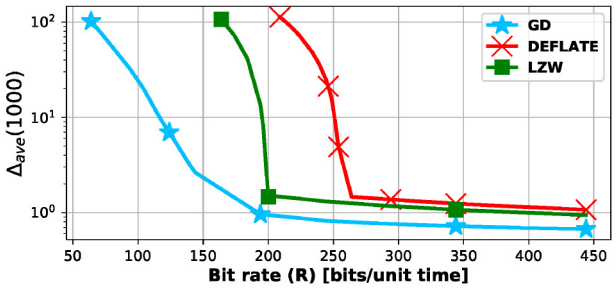

Fig. 9. AoI after receiving the 1000th chunk with respect to R. GD compresses data without side information.

Then, we collect the first one (the most repetitive one) from each of them followed by the next one, and so on.

The selected bit rate for each experiment for GD and for other compression schemes is equal to the minimum bit rate calculated in (6) for compression with side information. Note that the minimum bit rate for compression with side information is higher than the minimum bit rate for compression without side information. Accordingly, the selected bit rate for compression without side information does not violate the (15). For having multiple source nodes, this minimum bit rate is multiplied by the number of the source nodes for all the different multiple source node's schemes.

Due to having small data chunks, we use CRC32 to generate 4 B long fingerprints. In case of having bigger data chunks, we can use other hash functions, such as SHA-1. We consider $L_{\text{req}} = L_{\text{Ack}} = 10$ for our evaluation.

Example 3: If we consider the CAN data set, compression with side information and a single source node, the minimum bit rate is equal to $R > [(L_{w,3}(24) + L_{\text{req}} + L_{\text{Ack}})/B] \Rightarrow R > [(1 + 16 + 32 + 24 + 200 + 10 + 10)/1] \rightarrow R > 293$ bits per unit time.

Around 50 symbols (recordings) are generated per second in our CAN data sets which implies each unit time is 0.02 s in our analysis. Thus, the bit rate in a real system should be higher than $(293/0.02) = 14650$ b/s for this data set.

VIII. Numerical Results

A. Single Source Node Evaluation

Encoder's Buffer Stability: Fig. 9 shows the average of status age at the time of receiving the 1000th chunk by the decoder with respect to R for GD, DEFLATE, and LZW, considering the Hyundai data set. The average of status age is calculated based on the simulation by measuring the required time for transmission and waiting time. This is because the analysis in Section V supports $R > 249$ according to the (15) for timely throughput condition. As it can be seen in Fig. 9, with decreasing R, the encoder's buffer is becoming unstable for all three schemes. This happens as the source node cannot transmit the accumulated data on time. Fig. 9 shows that GD is more resilient to the reduction of R. The encoder's buffer is stable for GD, DEFLATE and LZW for a minimum bit rate of 150, 200, and 265 bits per unit time.

Performance Bounds: Fig. 10 shows the performance bounds with respect to R for $B = 16$ and chunk length of 32 B

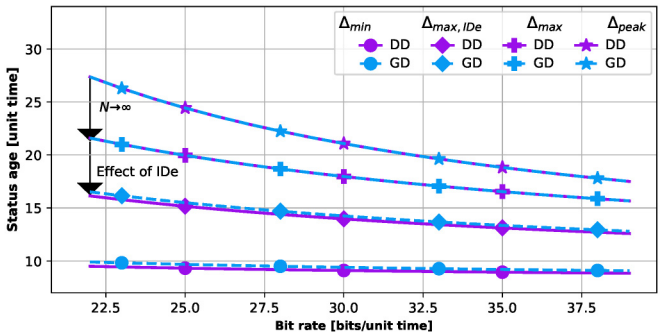

Fig. 10. Boundaries for $B = 16$, and chunk length of 32 B, using a Hamming code as the transformation function and compression without side information. One source symbol is generated per unit time.

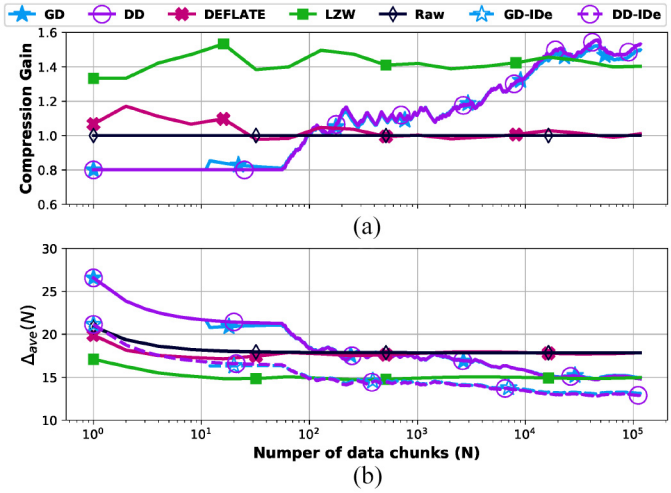

Fig. 11. (a) Compression gain, and (b) AoI, both as the number of data chunk grows for the temperature data set. GD and DD compress the data without side information.

when using a Hamming code as the transformation function for compression without side information. When N goes to infinity, the upper bound is reduced by up to 21%. By using IDe, the upper bound is improved further by up to 24%. The lower bound for DD is slightly better than the lower bound for GD as it does not need to send a deviation in the best case of transmission.

Effect of the Number of Transmitted Chunks: Fig. 11 shows how compression gain and $\Delta_{\text{ave}}(N)$ changes by transmitting more and more data chunks for different compression schemes including GD and DD without side information. Note that GD and DD have similar performance for this data set. DEFLATE and LZW provide nearly constant compression gain, which leads to having an almost constant $\Delta_{\text{ave}}(N)$. Early variations in $\Delta_{\text{ave}}(N)$ for all the schemes are caused by the temporary effect of the last term in (4). After transmitting around 60 data chunks, GD and DD start finding matches in the sink node to previously sent data chunks. This increases the compression gain and improves the $\Delta_{\text{ave}}(N)$. As expected, Fig. 11 shows that GD and DD have a better compression gain as more chunks have been received at the sink node. Our schemes using GD and DD outperform DEFLATE and LZW after having 100 and 16 000 transmitted data chunks, respectively. By applying IDe, our schemes outperform DEFLATE and LZW in terms

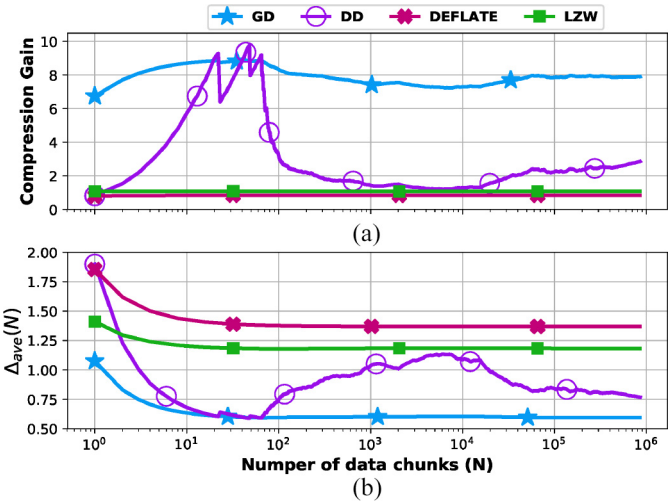

Fig. 12. (a) Compression gain, and (b) $\Delta_{\text{ave}}(N)$, both as the number of chunks grows for the Hyundai data set for 100-kB flash memory and 20-kB RAM.

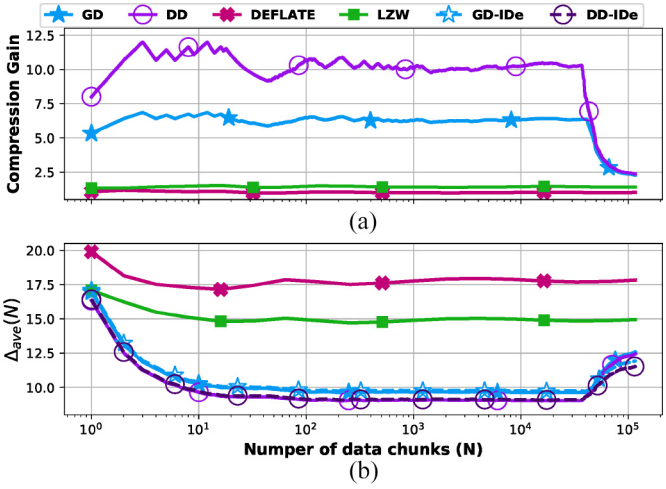

Fig. 13. (a) Compression gain, and (b) $\Delta_{\text{ave}}(N)$, both as the number of chunks grows for the temperature data set for 100-kB flash memory and 20-kB RAM.

of the $\Delta_{\text{ave}}(N)$ after transmitting only 3 and 100 data chunks, respectively.

Let us consider the case of GD and DD with side information. Figs. 12 and 13 depict how the compression gain and $\Delta_{\text{ave}}(N)$ change by having more transmitted chunks for GD and DD considering 100-kB flash memory and 20-kB RAM for the Hyundai and the temperature data sets. The results for DEFLATE and LZW are shown as a baseline. Our results show that using dynamic and static dictionaries have the potential to improve the performance, considerably. For instance, GD provides up to 56% and 50% better $\Delta_{\text{ave}}(N)$ compared to DEFLATE and LZW, respectively, for the Hyundai data set. For both data sets, GD and DD outperform DEFLATE and LZW.

For the Hyundai data set in Fig. 12, GD has a compression gain of an order of magnitude higher than DEFLATE, LZW and also DD (after enough chunks are considered). This is due to the fact that the present dictionary for this data set is particularly effective, i.e., GD typically finds bases in the

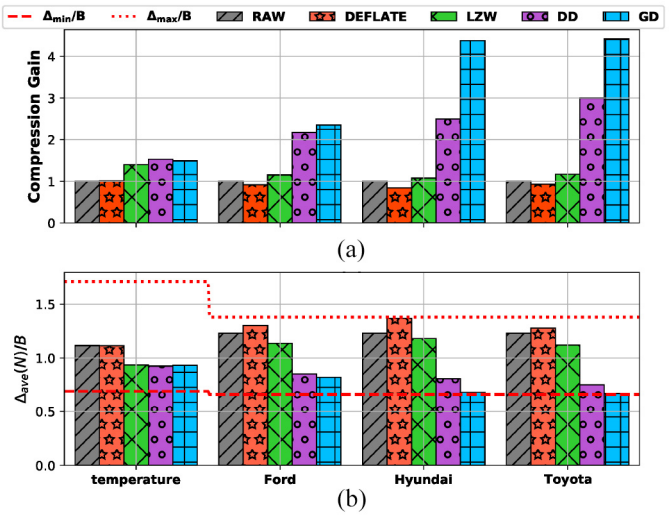

Fig. 14. Final values of (a) compression gain, and (b) final normalized $\Delta_{\text{ave}}(N)$, and the normalized upper and lower bounds for different real-world data sets. GD and DD compress the data without side information.

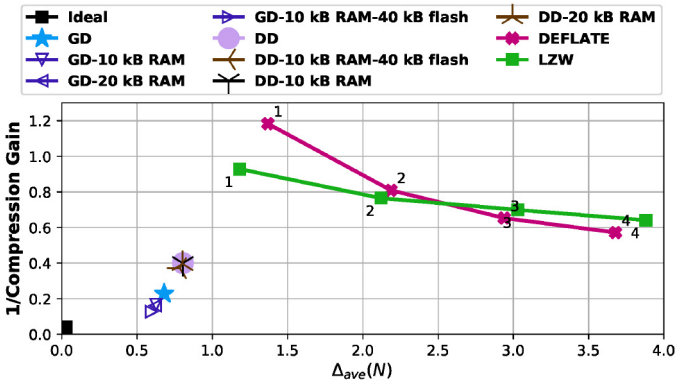

Fig. 15. Effect of the chunk size on DEFLATE and LZW for the Hyundai data set. Bigger chunks are made by bundling several small chunks. The numbers 1, 2, 3, and 4 show the number of bundled chunks per bigger chunk. The results for GD and DD with and without side information is reported.

static dictionary. This large compression gain provides a direct benefit on the $\Delta_{\text{ave}}(N)$ of GD compared to the other schemes. As $B = 1$ for the Hyundai data set, we do not show separate results for GD-IDe and DD-IDe in Fig. 12.

For the temperature data set in Fig. 13, there is a larger number of bases with small repetitions and few bases with high repetitions. Thus, the static dictionary has a larger number of fingerprints repeated only a few times, which limits the compression potential. In this particular data set, as $N > 40000$ the number of matches to the static dictionary is reduced, which causes both the compression gain and the $\Delta_{\text{ave}}(N)$ to degrade. By default, GD and DD will be decoding immediately the data in this scenario.

Resulting Compression Gain and AoI for Each Data Set: Fig. 14 compares the final values for compression gain and the $[(\Delta_{\text{ave}}(N))/B]$ for our data sets. We consider the case without side information for GD and DD. We use $[(\Delta_{\text{ave}}(N))/B]$ as the normalized value for $\Delta_{\text{ave}}(N)$ to be able to carry out a fair comparison. This is due to the fact that $\Delta_{\text{ave}}(N)$ is normally close to B, because the source node needs to wait for B time units to have the next data chunk, and that the temperature and CAN bus data sets have different B values (16 and 1, respectively). Fig. 14 shows that GD outperforms all other schemes for the Ford, Hyundai and Toyota data sets while DD provides the best results for the temperature data set. In general, LZW performs better than DEFLATE for these data sets considering our small data chunks. For the Hyundai data set, GD provides up to 1.75, 4.06, and 5.18 times better compression gain compared to DD, LZW, and DEFLATE, respectively. Fig. 14 also shows the normalized values for $\Delta_{\min}$, and $\Delta_{\max}$ for all data sets. As depicted in Fig. 14, $[(\Delta_{\text{ave}}(N))/B]$ for the Hyundai data set and the Toyota data set, considering GD, is pretty close to the normalized value of the lower bound, i.e., $[\Delta_{\min}/B]$. However, for the Ford data set and the temperature data set, there is an obvious difference.

Effect of the Chunk Size on DEFLATE and LZW: Standard compression techniques, such as DEFLATE and LZW normally require collecting and analyzing enough data to provide good compression [27]. On the other hand, the time required to collect this amount of data negatively affects the timeliness and freshness of the data. Fig. 15 compares the final values for (1/compression gain) (a lower value is better) and $\Delta_{\text{ave}}(N)$ for different schemes for the Hyundai data set when we bundle several chunks into bigger chunks for LZW and DEFLATE. The numbers beside the markers show the number of bundled chunks to create a bigger chunk. Thus, the number 1 shows the case where the chunk size for LZW and DEFLATE is equal to the chunk size for GD and DD. This is our default configuration for our results. The ideal case is shown with a black square at the bottom left corner. A closer point to this point means better performance. As we bundle more and more chunks for LZW and DEFLATE, the resulting compression gain improves slightly but it does so at the expense of a larger $\Delta_{\text{ave}}(N)$ driven primarily due to the delay to collect the data prior to compression. Namely, LZW and DEFLATE introduce a tradeoff between compression gain and the AoI.

Fig. 15 shows that GD has the closest performance to the ideal case with or without side information. Fig. 15 also shows that the final values for both the compression gain and $\Delta_{\text{ave}}(N)$ improve by introducing extra RAM and flash memory (side information) to GD and DD. In other words, GD removes the tradeoff between compression gain and the AoI. However, GD's performance improves significantly when providing more RAM and flash memory sizes at the source nodes. The performance of DD improves with more side information, but not as dramatically as GD. This is related to the fact that GD's use of preset and dynamic dictionaries in the flash and RAM, respectively, is more efficient than DD. GD's mapping of a single basis in the dictionary can benefit a large number of chunks, while DD will only benefit a chunk with exactly the same content.

Effect of RAM and Flash Memory Size: In general, increasing the size of the flash memory increases the number of already available fingerprints and bases at the source node by default. Increasing the RAM size helps to reduce the transmission costs once a basis has been seen before. In Fig. 16, we study how these parameters change the final values for compression gain and $\Delta_{\text{ave}}(N)$ of GD for the Toyota and Hyundai

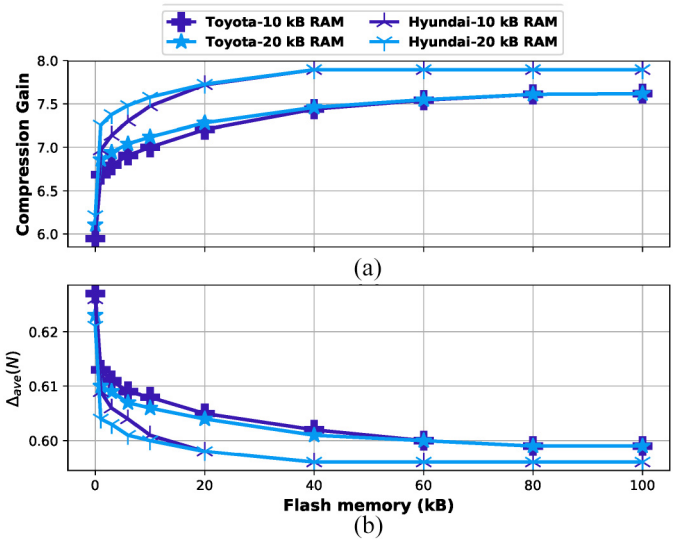

Fig. 16. Final values for (a) compression gain, and (b) $\Delta_{\text{ave}}(N)$ considering different sizes for RAM and flash memory for Toyota and Hyundai data sets under the GD scheme.

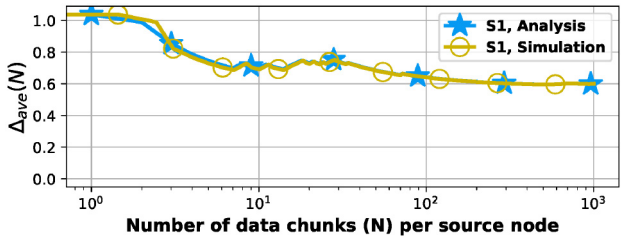

Fig. 17. Analysis and the simulation results for $\Delta_{\text{ave}}(N)$ for S_1 as the number of chunks grows for the case of *sequential node activity, shared communication channel* considering the Toyota data set. GD compresses the data without side information.

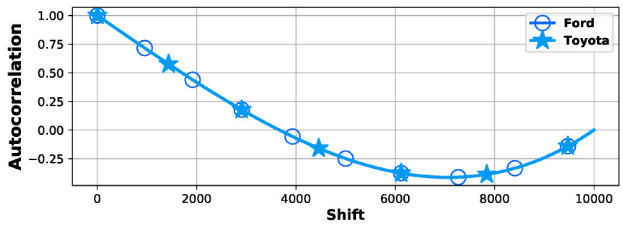

Fig. 18. Autocorrelation for the Ford and Toyota data sets considering 10 000 chunks.

data sets. As can be seen, a small flash memory improves the compression gain and $\Delta_{\text{ave}}(N)$. For instance, the compression gain for the Hyundai data set with 10-kB RAM increases from 5.97 to 7.47 by having only 10-kB flash memory. Likewise, the compression gain for the Toyota data set with 20-kB RAM increases from 6.10 to 6.86 by having only 1-kB flash memory. The improvement caused by introducing the extra RAM is more obvious for smaller flash memory sizes. For the Hyundai data set and no flash memory, compression gain improves from 5.97 to 6.21 by increasing the RAM size from 10 to 20 kB. As expected, changes in compression gain affect the $\Delta_{\text{ave}}(N)$.

Fig. 16 shows that the compression performance is nearly constant after 40-kB flash memory size. This is due to the fact that there is a limited set of bases that are repeated a large number of times. Including additional bases in the dictionaries does not have a large impact in performance, because these additional bases are not frequently used. However, the key take away from these results is that keeping a small preset dictionary in resource-limited source nodes can make a large difference in performance.

B. Multiple Source Nodes Evaluation

Verification of the Theoretical Analysis: In Section VI, we studied the timeliness of our scheme for multiple source nodes. In Fig. 17, we show that the simulation results verifies the analysis. Fig. 17 shows the analysis and the simulation results for $\Delta_{\text{ave}}(N)$ for the first 1000 chunks for S_1 as the number of chunks grows for the case of *sequential node activity, shared communication channel* considering GD without side information. Two source nodes are considered. We used part of the Toyota data set for one source node and part of it for the other source node. To calculate the simulation results, the transmission and waiting time for each encoded block is calculated to calculate the instantaneous status age. Then, the area under the instantaneous status age is calculated using python.

Data Similarity Versus Performance: GD performance is a function of data similarity. One criteria which measures data similarity is *autocorrelation*. Autocorrelation measures the correlation of a signal with a shifted copy of itself as a function of shift. Fig. 18 shows autocorrelation parameter for the Ford and the Toyota data sets considering 10000 consecutive chunks chosen at random. In our case, the signal is the series of chunks, i.e., each chunk is considered as a sample. Fig. 18 depicts that there is a strong correlation between different chunks for both data sets. However, this confirms the good performance of GD for these data sets, but it does not show the difference between the two data sets. In Fig. 14, it was depicted that GD provides better compression and AoI for the Toyota data set compared to the Ford data set. That is, why we also study the number of bit flips between consecutive and simultaneous chunks as a measure of similarity.

Fig. 19 shows the distribution of the number of bit flips for the consecutive chunks from the same source node (temporal) and the simultaneous chunks from different source nodes (spatial) for the Toyota and the Ford data sets. Two source nodes are considered. Fig. 19 shows the bit flip distribution for the raw chunks, chunks with differential timestamps and the bases. Each chunk is $27 \cdot 8 = 216$ bits. The temporal and the spatial bit flip peaks for the raw chunks for the Toyota data set are 7 and 23, respectively. That is while these values are 32 and 46, respectively, for the Ford data set. After calculating the differential timestamps, the bit flip distribution moves slightly to the left for both data sets. However, for the bases, the distribution of the bit flips moves dramatically to the left, i.e., lower number of bit flips. But, this is not the case for the Toyota data set. This confirms the results in Fig. 14 where we observed that GD provides AoI close to the lower bound ($\Delta_{\min}$) for the Toyota data set but not for the Ford data set.

Compression Without Side Information: Fig. 20 depicts the compression gain and $\Delta_{\text{ave}}(N)$ for having two source nodes

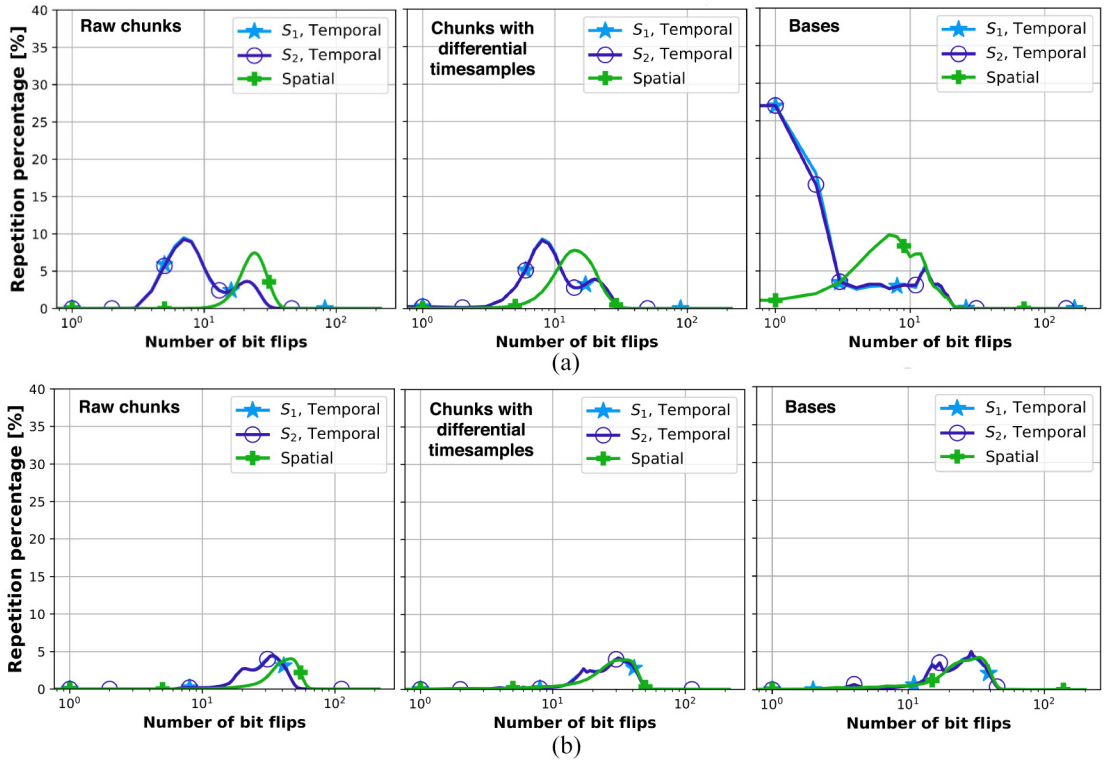

Fig. 19. Temporal and Spatial distribution of the number of bit flips for the raw chunks, chunks with differential time samples, and the bases for the (a) Toyota data set and (b) Ford data set.

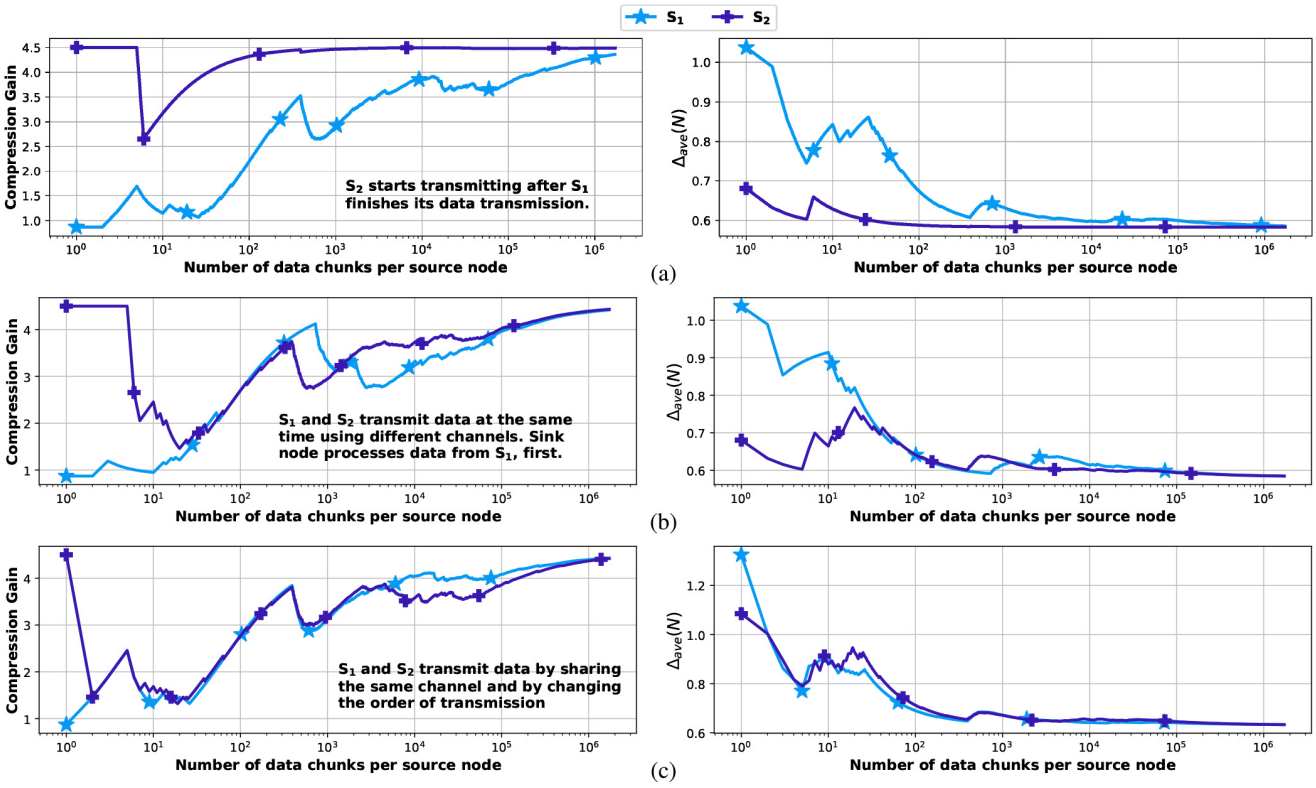

Fig. 20. Compression gain and $\Delta_{ave}(N)$ for having two source nodes considering the Toyota data set for (a) sequential node activity, (b) simultaneous node activity, independent communication channels, and (c) simultaneous node activity, shared communication channel.

and considering the Toyota data set for different transmission scenarios. We use half of the Toyota data set for one source node and the other half for the second source node. Fig. 20(a) shows the performance for the case of *sequential node activity* where the second source node, S_2, starts transmitting data after S_1 finishes its data transmission. Since S_2 can take advantage

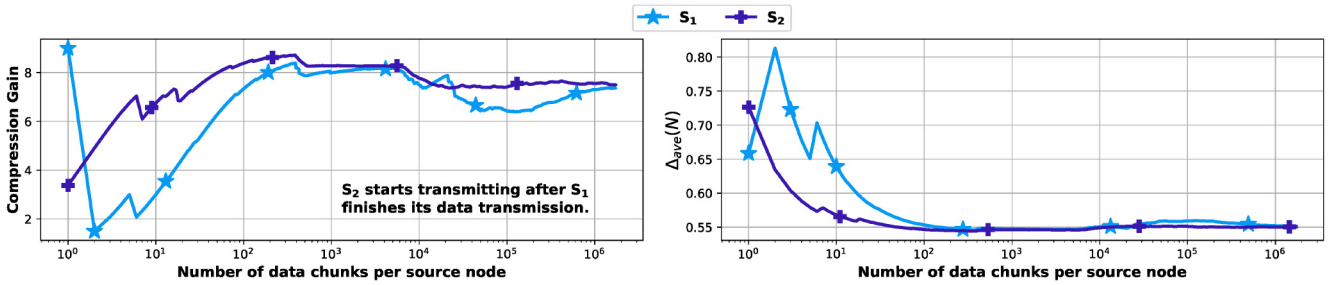

Fig. 21. Compression gain and $\Delta_{\text{ave}}(N)$ for having two source nodes considering the Toyota data set for sequential node activity and considering 10-kB RAM and 40-kB flash memory.

of all the fingerprints available at the sink node transmitted by S_1, the compression gain starts at 4.5 even in early transmissions. The compression gain for S_1 starts from 0.8 as we assume no side information and no bases stored in the sink node. $\Delta_{\text{ave}}(N)$ follows similar behavior where the starting point for S_2 is as low as 0.68 while the starting point for S_1 is 1.04.

Fig. 20(b) shows the performance for the case of *simultaneous node activity, independent communication channels* where each source node has its channel and both of them can use the data already transmitted by the other source node. In general, similar behavior is expected for both source nodes. However, since the sink node processes the data from S_1 first, S_2 has a slightly higher performance advantage. For instance, in the first transmission of both these sources the bases happen to be equal. Both source nodes send the same fingerprint to identify the basis. The sink node asks S_1 to send the basis. Thus, it does not need to ask S_2 for the associated basis. This is the reason for the better performance of S_2 for the number of transmitted chunks per source node less than 30. Afterward, both sources have similar performance, which is expected.

Fig. 20(c) depicts the performance for the case of *simultaneous node activity, shared communication channel*, where S_1 and S_2 share the same channel and they change the order of transmission. This means half of the time S_1 transmits first and half of the time S_2 transmits first, as described in Section III-D. Thus, the performance of both nodes is similar during the entire process, except for the first transmission, where a similar situation to that described for independent communication channels occurs.

Compression With Side Information: Fig. 21 depicts the compression gain and $\Delta_{\text{ave}}(N)$ for two source nodes considering the Toyota data set, for sequential node activity and considering 10-kB RAM and 40-kB flash memory. As can be seen, introducing flash memory and RAM has improved the compression gain beyond a factor of 8 (best case without side information) and has reduced the final value for $\Delta_{\text{ave}}(N)$ to 0.55. Since S_2 can take advantage of the data previously sent by S_1, it has better performance. The fingerprint for the first chunk for S_1 is available at the flash memory, which provides a good starting point for it.

IX. Conclusion

This article analyzes the AoI for a family of lossless, multisource compression protocols. We show that protocols built using the emergent concept of GD can improve compression and AoI significantly at the same time, while, state-of-the-art compressors introduce a tradeoff between these two metrics. In other words, we show that our techniques remove the tradeoff between these two metrics. We show that the use of preset and dynamically generated dictionaries in the source nodes can improve both these metrics. We also demonstrate that each source node in our scheme can benefit not only from their own transmissions but from transmissions of other sources to strictly improve their individual performance. Our evaluation with real data sets show that compression can be improved by up to an order of magnitude while simultaneously reducing the AoI by a factor of up to 2.3 compared to uncompressed data, DEFLATE and LZW. Our future work will focus on studying scenarios with lossy communication channels that can generate additional transmission delays. This randomness will require us to consider the effects of buffers and memory in our AoI analysis as well as the stable throughput region of our scheme. We will also study the potential of our proposal to advance research in real-time systems. These could benefit from providing strict guarantees on the certain data sources, while using the idle time left by our compression technique to serve best effort traffic in the system and allow for more opportunities for scheduling, e.g., as an addition to FTT-CAN [42] and other CAN bus protocols. Although compression of CAN Bus data logs is common, online compression to influence scheduling decisions is not. To the best of our knowledge, only the work in [43] and [44] have attempted to address this issue with simple techniques.

Acknowledgment

The authors thank Christian Steiniche and Martin Falch from CSS Electronics for their support in there data collection process.

References

[1] N. Yazdani and D. E. Lucani, "Age of information analysis for instantly decompressible IoT protocols," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, 2020, pp. 1–6.
[2] M. Khabbaz, J. Antoun, and C. Assi, "Modeling and performance analysis of UAV-assisted vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8384–8396, Sep. 2019.
[3] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *Proc. IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 2731–2735.
[4] S. Kaul, M. Gruteser, V. Rai, and J. Kenney, "Minimizing age of information in vehicular networks," in *Proc. 8th Annu. IEEE Commun. Soc. Conf. Sens. Mesh Ad Hoc Commun. Netw.*, Salt Lake City, UT, USA, 2011, pp. 350–358.
[5] J. Zhong and R. D. Yates, "Timeliness in lossless block coding," in *Proc. Data Compression Conf. (DCC)*, Snowbird, UT, USA, 2016, pp. 339–348.

[6] Y. Inoue, H. Masuyama, T. Takine, and T. Tanaka, "A general formula for the stationary distribution of the age of information and its application to single-server queues," *IEEE Trans. Inf. Theory*, vol. 65, no. 12, pp. 8305–8324, Dec. 2019.
[7] I. Krikidis, "Average age of information in wireless powered sensor networks," *IEEE Wireless Commun. Lett.*, vol. 8, no. 2, pp. 628–631, Apr. 2019.
[8] R. D. Yates, "Status updates through networks of parallel servers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, USA, 2018, pp. 2281–2285.
[9] Y. Sun, E. Uysal-Biyikoglu, and S. Kompella, "Age-optimal updates of multiple information flows," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Honolulu, HI, USA, 2018, pp. 136–141.
[10] B. Yin, S. Zhang, and Y. Cheng, "Application-oriented scheduling for optimizing the age of correlated information: A deep-reinforcement-learning-based approach," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8748–8759, Sep. 2020.
[11] A. Javani, M. Zorgui, and Z. Wang, "Age of information in multiple sensing," in *Proc. IEEE Inf. Theory Appl. Workshop (ITA)*, Waikoloa, HI, USA, 2020, pp. 1–10.
[12] Y. Inoue, "Analysis of the age of information with packet deadline and infinite buffer capacity," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, USA, 2018, pp. 2639–2643.
[13] Z. Jiang, B. Krishnamachari, X. Zheng, S. Zhou, and Z. Niu, "Timely status update in wireless uplinks: Analytical solutions with asymptotic optimality," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3885–3898, Apr. 2019.
[14] H. Tang, J. Wang, L. Song, and J. Song, "Minimizing age of information with power constraints: Multi-user opportunistic scheduling in multi-state time-varying channels," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 5, pp. 854–868, May 2020.
[15] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Minimizing the age of information in broadcast wireless networks," in *Proc. IEEE 54th Annu. Allerton Conf. Commun. Control Comput. (Allerton)*, Monticello, IL, USA, 2016, pp. 844–851.
[16] B. Zhou and W. Saad, "Minimum age of information in the Internet of Things with non-uniform status packet sizes," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 1933–1947, Mar. 2020.
[17] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, "Update or wait: How to keep your data fresh," *IEEE Trans. Inf. Theory*, vol. 63, no. 11, pp. 7492–7508, Nov. 2017.
[18] Q. He, G. Dan, and V. Fodor, "Minimizing age of correlated information for wireless camera networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Honolulu, HI, USA, 2018, pp. 547–552.
[19] J. Zhong, R. D. Yates, and E. Soljanin, "Backlog-adaptive compression: Age of information," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, 2017, pp. 566–570.
[20] S. Poojary, S. Bhambay, and P. Parag, "Real-time status updates for Markov source," *IEEE Trans. Inf. Theory*, vol. 65, no. 9, pp. 5737–5749, Sep. 2019.
[21] J. Hribar, M. Costa, N. Kaminski, and L. A. DaSilva, "Using correlated information to extend device lifetime," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2439–2448, Apr. 2019.
[22] A. E. Kalør and P. Popovski, "Minimizing the age of information from sensors with common observations," *IEEE Wireless Commun. Lett.*, vol. 8, no. 5, pp. 1390–1393, Oct. 2019.
[23] B. Zhou and W. Saad, "On the age of information in Internet of Things systems with correlated devices," 2020. [Online]. Available: arXiv:2001.11162.
[24] R. Vestergaard, Q. Zhang, and D. E. Lucani, "Generalized deduplication: Bounds, convergence, and asymptotic properties," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1–6.
[25] L. Nielsen, R. Vestergaard, N. Yazdani, S. Talasila, D. L. Rötter, and M. Sipos, "Alexandria: A proof-of-concept implementation and evaluation of generalised data deduplication," *Proc. IEEE Globecom Workshops (GC Wkshps)*, Waikoloa, HI, USA, Dec. 2019, pp. 1–6.
[26] R. Vestergaard, D. E. Lucani, and Q. Zhang, "Generalized deduplication: Lossless compression for large amounts of small IoT data," in *Proc. 25th Eur. Wireless Conf.*, Aarhus, Denmark, 2019, pp. 1–5.
[27] N. Yazdani and D. E. Lucani, "Protocols to reduce CPS sensor traffic using smart indexing and edge computing support," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Waikoloa, HI, USA, Dec. 2019, pp. 1–6.
[28] R. Vestergaard, Q. Zhang, and D. E. Lucani, "Lossless compression of time series data with generalized deduplication," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1–6.
[29] R. Vestergaard, D. E. Lucani, and Q. Zhang, "A randomly accessible lossless compression scheme for time-series data," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Toronto, ON, Canada, 2020, pp. 2145–2154.
[30] P. R. Kumar and S. P. Meyn, "Duality and linear programs for stability and performance analysis of queuing networks and scheduling policies," *IEEE Trans. Autom. Control*, vol. 41, no. 1, pp. 4–17, Jan. 1996.
[31] J. C. Moreira and P. G. Farrell, *Essentials of Error-Control Coding*. Chichester, U.K.: Wiley, 2006.
[32] N. Yazdani, L. Nielsen, and D. E. Lucani, "Memory-aware online compression of CAN bus data for future vehicular systems," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Taipei, Taiwan, 2020, pp. 1–6.
[33] *Can Specification Version 2.0*, Robert Bosch GmbH, Gerlingen, Germany, 1991.
[34] P. Deutsch, "DEFLATE compressed data format specification version 1.3," IETF, RFC 1951, 1996. [Online]. Available: https://tools.ietf.org/html/rfc1951
[35] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.
[36] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[37] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984.
[38] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, Sep. 1978.
[39] *Intel Lab Data*. Accessed: Jun. 15, 2019. [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html
[40] *CSS Electronics*. Accessed: Apr. 18, 2020. [Online]. Available: https://www.csselectronics.com/
[41] *ASAM MDF*. Accessed: Apr. 15, 2020. [Online]. Available: https://www.asam.net/standards/detail/mdf/wiki/
[42] L. Almeida, P. Pedreiras, and J. A. G. Fonseca, "The FTT-CAN protocol: Why and how," *IEEE Trans. Ind. Electron.*, vol. 49, no. 6, pp. 1189–1201, Dec. 2002.
[43] S. Kelkar and R. Kamal, "Boundary of fifteen compression algorithm for controller area network based automotive applications," in *Proc. Int. Conf. Circuit Syst. Commun. Inf. Technol. Appl. (CSCITA)*, 2014, pp. 162–167.
[44] Y. Wu, J.-G. Chung, and M. H. Sunwoo, "Design and implementation of CAN data compression algorithm," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Melbourne, VIC, Australia, 2014, pp. 582–585.

Niloofar Yazdani (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the K. N. Toosi University of Technology, Tehran, Iran, in 2011 and 2013, respectively.

She is a Ph.D. Fellow with the Department of Engineering, Aarhus University, Aarhus, Denmark. She has been a Visiting Researcher with TU Delft, Delft, The Netherlands, since February 2021. From 2012 to 2013, she was with the K. N. Toosi University of Technology and the University of Allameh-Majlesi, Qazvin, Iran, as an Laboratory Instructor and an Instructor. From 2014 to 2017, she worked with Research and Development Department, Mega-Motor Company,Tehran, as an Electronic System Designer. She was also with the Integrated Circuits and Systems Laboratory, K. N. Toosi University of Technology as an Associate Researcher. Her research interests include communications systems, network coding, IoT, and data compression.

Dr. Yazdani is a recipient of the Best Paper Award from the IEEE Wireless Africa Conference in 2019.

Daniel E. Lucani (Senior Member, IEEE) received the B.S. (*summa cum laude*) and M.S. (Hons.) degrees in electronics engineering from Universidad Simon Bolvar, Caracas, Venezuela, in 2005 and 2006, respectively, and the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2010.

He has been an Associate Professor with the Department of Engineering, Aarhus University, Aarhus, Denmark, since April 2017 as well as the CEO and a Lead Scientist of the start-up company Chocolate Cloud ApS, Aalborg, Denmark, since June 2014. He was an Associate Professor with Aalborg University, Aalborg, from 2012 to 2017, and an Assistant Professor with the University of Porto, Porto, Portugal, from 2010 to 2012. He has published more than 180 scientific papers in international journals and top-ranked international conferences as well as eight patents and patent applications. His research interests include communications and networks, network coding, information theory, coding theory, distributed storage and computation, and their applications to cloud computing technologies necessary to enable IoT, BigData, and 5G applications and services.

Dr. Lucani is a recipient of the IEEE ComSoc Outstanding Young Researcher Award for the EMEA region in 2015 and the Danish Free Research Foundation's Sapere Aude Starting Grant. He was the General Co-Chair of the 2014 International Symposium on Network Coding (NetCod2014). He is an Associate Editor of the *EURASIP Journal on Wireless Communications and Networking*.