Efficient Communication Protocol for Programmable Matter

Jean-Paul A. Yaacoub, Benoit Piranda, Frederic Lassabe and Hassan N. Noura

Abstract Lattice-based modular robots are composed of modules arranged on a lattice and forming 3D shapes, if these robots are small enough and many enough, they form a programmable matter. This work proposes a method for optimising data communication times between modules by compressing the data. We have first analysed the communication delay between the end device modules, then a set of recent lossless compression algorithms was tested to select the optimal one to implement with *Blinky Block*. Based on the results obtained, we propose to add a lossless data compression scheme to reduce the communicated data size and consequently communication delay. We found that the ”Brotli” compression algorithm is the most suitable one for modular robot communication as it achieved a good balance between computing and communication overhead. Then, based on the compression ratio and the communication delay interpolation, a significant gain is achieved by reducing the communication delay by a factor of 5.

1 Introduction

Programmable matter is made of small autonomous building blocks that can be programmed to achieve a wide range of geometric objects and structures with pro-

Jean-Paul A. Yaacoub
Univ. Franche-Comté, FEMTO-ST Inst., CNRS, e-mail: `jean.absyaacoub@femto-st.fr`

Hassan N. Noura
Univ. Franche-Comté, FEMTO-ST Inst., CNRS, e-mail: `Hassan.noura@univ-fcomte.fr`

Frederic Lassabe
UTBM, FEMTO-ST Inst., CNRS, e-mail: `frederic.lassabe@utbm.fr`

Benoit Piranda
Univ. Franche-Comté, FEMTO-ST Inst., CNRS, e-mail: `benoit.piranda@femto-st.fr`

grammable capabilities to change their colour or shape, which leads to the creation of programmable matter [1].

Most of the algorithm use communication capabilities of robots to share local information in order to enlarge global knowledge of the set. These communications are the weak point of distributed algorithms, as they represent the longest processing time. We will show that communication time is mainly due to the size of the data embedded in the messages. Even if the computational capabilities of the robots used in the programmable subject are quite small, the idea developed here is to use these computational capabilities to process the data received in order to reduce the size of the data transmitted.

The context used in this article is central to the problem of self-reconfiguration of programmable matter. Self-reconfiguration consists in programming modular robots so that modules move relatively to each other to change the overall shape of the assembly [2, 3, 4].

The preliminary step in any self-reconfiguration algorithm is to give the modules a way of knowing the final shape to be made. In [5], Tucci et al presented a very efficient 3D scene encoding model for the self-reconfiguration process that describes a 3D models in the form of a Constructive Solid Geometry tree (CSG tree) combining the simple geometric objects placed in the leaves. Combination can be union, intersection and difference of sub-trees. A string code may be generated from a depth search first traversing of the tree.

Blinky Blocks are small cubic modular robots that make up the key component of the Claytronics project to create highly adaptable and reconfigurable objects and environments (cf. Fig. 1). Each *Blinky Block* can be attached with its magnets to form complex geometric shapes, can exchange messages with directly connected neighbours, and react to noise by emitting sounds or/and changing colours. We use these real robots as a test bed to validate some parts of distributed algorithms for programmable matter. Despite the fact that they have no autonomous movement capability, their communication and computing capacity means that algorithms for programmable matter can be implemented on several hundred connected real robots.

The adoption of modular robots in the nature of the Internet of Things (IoT) [6] with the implementation of AI-empowered applications and services [7], can

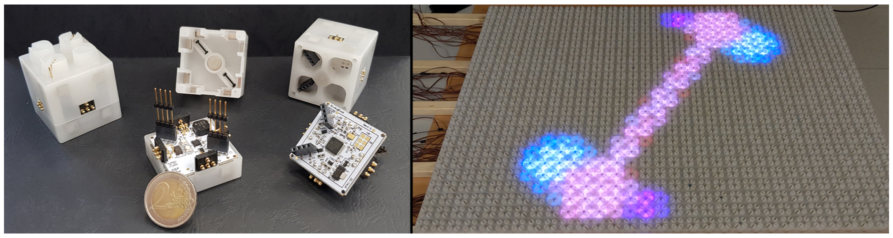


Fig. 1: Left: *Blinky Block* Hardware. Right: a set of 768 *Blinky Blocks* running the same program to visualise a cutting plane of 3D Objects.

reshape the robotics concept [8, 9] into a new modular self-reconfigurable swarm, capable of operating in large numbers synchronously and simultaneously.

Communication between similar modular robotic systems as IoT components is essential to perform the intended task. However, this can be delayed due to the message size, which the length of the message can prove to be challenging and result in communication delays. For that, several solutions for Wireless Multimedia Sensor Networks (WMSN) were presented such as in [10] to reduce this redundancy by discarding a certain number of data packets while guaranteeing its integrity (quality). Other solutions include low-overhead data compression techniques [11], Compressed Sensing (CS) algorithms for data compression [12], and data compression and transmission scheme for power reduction in IoT enabled wireless sensors [13].

However, they are prone to delays which can affect their ability to react in real-time which is often caused not only by the *Blinky Blocks* number but rather by the Message Length (ML), which the higher the message, the higher the delay will become. As a result, several experimental results were tested on different compression/decompression algorithms to verify which one is more suitable to be applied to *Blinky Blocks* to mitigate the issue of delay and ensure a higher real-time reaction to users' orders and commands.

The following section presents preliminary work on the study of robots to evaluate their communication and computation capabilities. The next part proposes a study of classical compression models compared to Huffman's method. Finally, our method is presented and completed by an experiment on a real problem applied to a large number of connected robots.

2 *Blinky Block* benchmark and compression models

A preliminary study of *Blinky Blocks* has enabled to assess their communication and pure computing capabilities. *Blinky Blocks* use very standard communication systems (6 UARTs, one on each side of the *Blinky Block*) and a processor very common in embedded systems (ARM Cortex M0 from STMicroelectronics, the STM32F091CB with 32 KB RAM and 128 KB flash memory), which allows us to generalise this study to most distributed multi-robot systems used in the context of programmable matter, such as the *3D Catom* [14].

First, in order to analyse the communication delay on *Blinky Blocks* in terms of the Message Length (M_l) and number of *Blinky Blocks* (N_{BB}), we place N_{BB} *Blinky Blocks* forming a simple line and we compute the communication time of several messages with different size of embedded data using the configuration presented Fig. 2.

Measurement of the total time taken to transfer a message on all *Blinky Blocks* is carried out by a distributed program running on the robots. This program starts with the first extremity A sending a message to its only neighbour, at the local time t_0 stored in A. When the message reaches an internal module with two neighbours, the message received is sent back to the connected opposite port. When the message

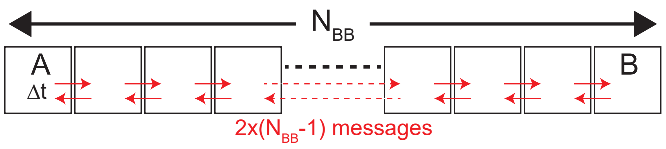
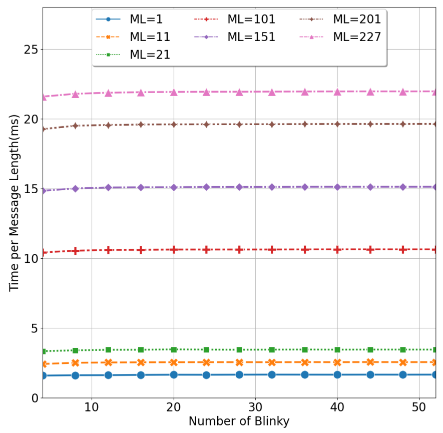


Fig. 2: Experimental network diagram used to measure message propagation times.

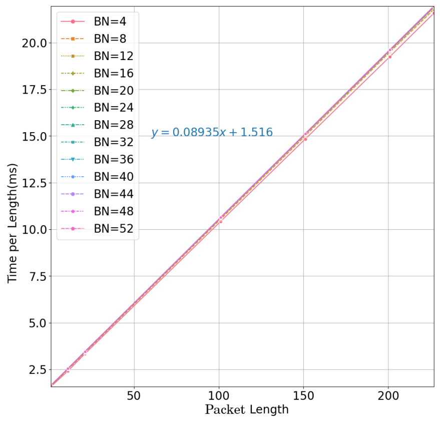
reaches the extremity B (which has only one neighbour connected), the message is sent back to the receiving port. When the back message reaches A at local time t_1, the time $\Delta_t = t_1 - t_0$ gives the average duration of $2 \times (N_{BB} - 1)$ message transfers where N_{BB} varies from 4 to 52 respectively.
We repeat this operation 1000 times to deduce the average duration of the transmission of messages (T_{ML}). The benchmark tests were performed on a series of 52 *Blinky Block* with each set being tested for a message of N bytes, with N taking 7 values in $[2..227]$. Based on the obtained results (see Fig. 3a and 3b), we found that only the message length has an effect on the communication time. Finally, we propose a linear approximation of the duration of the message depending on its length:

$$t = 0.08935 \times M_l + 1.516 \tag{1}$$

In a second study, we carried out a number of calculations on each *Blinky Block* set, such as mathematical operations and decompression using Huffman's method. This study led us to the conclusion that all the computations required in distributed algorithms for programmable matter were negligible compared with communication time. Here, for example, decompressing a Huffman code of 1061 bytes takes



(a) Function of the number of *Blinky Blocks*.



(b) Function of message length

Fig. 3: Variation of communication delay

15 ms, which is comparable to sending 150 bytes from a *Blinky Block* to a neighbour.

Thirdly, we studied the various compression algorithms available and compared them with the Huffman method implemented on our *Blinky Blocks*. Data compression algorithms can be divided into two classes: Lossless Compression which allows the original data to be fully reconstructed from the compressed data and with no information loss, and Lossy Compression which is especially used for multimedia data such as images and audio. It allows the original data to be reconstructed with a certain loss of information, but it can achieve better data reduction compared to lossless compression as it allows more space to be freed up.
In our case, the type of compression is message (textual data) compression and consequently, the required compression time should also be lossless, due to its ability to prevent the loss of any data during the compression/decompression process to avoid any modification to the original sent message. On the other hand, Fig. 4 represents a taxonomy of existing lossless data compression schemes. In this paper, a lossless set of these compression schemes was tested to confirm whether they are suitable to be implemented with lattice-based modular robots or not.

A description of the most known and widely used lossless compression algorithms is presented in Fig. 4. We tried different kinds of compression methods to give a brief description of each of the widely selected lossless data compression algorithms [15]:

- **DEFLATE:** is a lossless compression algorithm widely used in many popular compression utilities like gzip, zip, and PNG. It uses a combination of Huffman coding and LZ77 sliding window compression to compress text data [16].
- **LZ77:** is a lossless compression algorithm that uses a sliding window technique to compress textual data. It works by identifying repeated patterns in the input text and replaces them with references to previous occurrences of the same pattern [17].

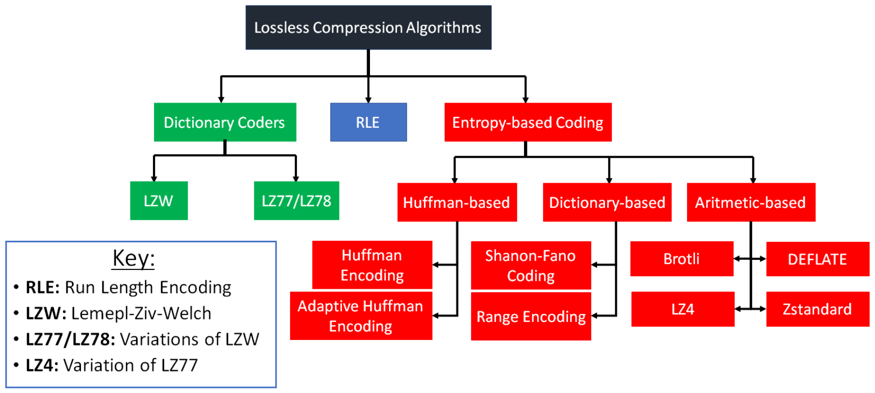
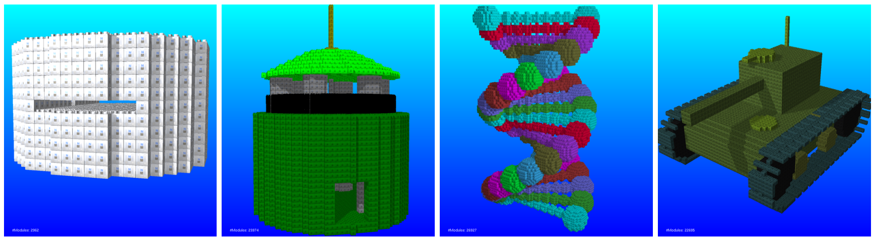


Fig. 4: Taxonomy of Existing Lossless Compression Algorithm Types.



(a) Bunker (b) Watch tower (c) DNA Chain (d) Battle tank

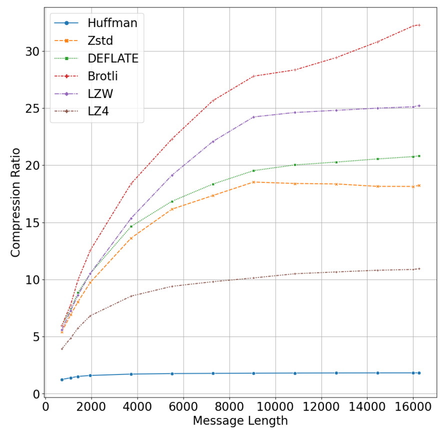
Fig. 5: VisibleSim view of 3D models used for experiments.

- **LZW:** stands for Lemepl-Ziv-Welch, is a dictionary-based lossless compression algorithm that is used in several popular file formats like GIF and TIFF. It works by building a dictionary of frequently occurring patterns in the input text and replaces them with shorter codes [18].
- **Brotli:** is a relatively new compression algorithm that was developed by Google. It uses a combination of a modern variant of the LZ77 algorithm, Huffman coding, and second-order context modelling to achieve higher compression ratios compared to other algorithms like DEFLATE [19].
- **Zstd:** stands short for Zstandard, and is a compression algorithm developed by Facebook. It uses a combination of Huffman coding, Finite State Entropy (FSE) compression, and a fast dictionary search algorithm to achieve high compression ratios and fast decompression speeds [20].

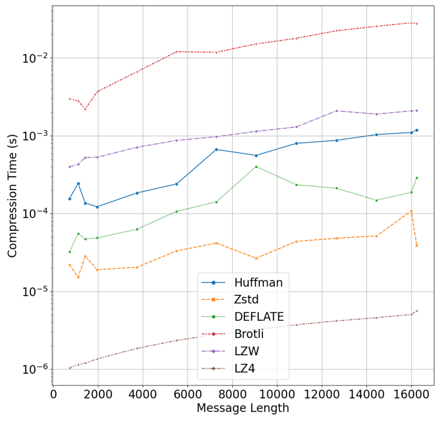
To test the effectiveness of our proposed solution, we decided to create more or less complex shapes to get different message length. Therefore, we designed four 3D models on *OpenSCAD* [21] modeller: (a) a ”Bunker”, (b) a fortified ”Watchtower”, (c) an ”ADN” and (d) a ”Battle Tank” and integrated them on the VisibleSim simulator [22] to create a set of *Blinky Blocks* that fills the models as shown in Fig. 5. Our vectorial description language alphabet being made of 32 different characters, the description models are encoded into a list of 5-bit codes building the CSG tree. In fact, Table 1 shows the code size for each one of them. this code can be compressed before sending it in the graph of modules and is locally decoded inside each module (without storing the model), before being integrated into our simulator and applied using Huffman decompression.

Table 1: **Size of the Designed Data Models.**

3D model	**Brut Size**	**5-Bit Coded**	**Huffman Header**	**Huffman Body**
Bunker	116 Bytes	580 bits	139 bits	429 bits
Watchtower	397 Bytes	249 bits	167 bits	1422 bits
DNA chain	3722 Bytes	18610 bits	139 bits	12147 bits
Tank	3986 Bytes	19930 bits	188 bits	14432 bits

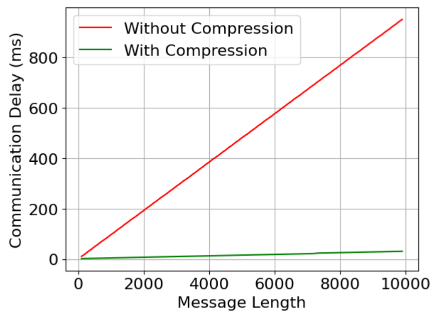


(a) Comparison of compression ratio versus message length.



(b) Variation of the compression time versus message length.

Fig. 6: Two efficiency comparisons for different lossless compression algorithms.



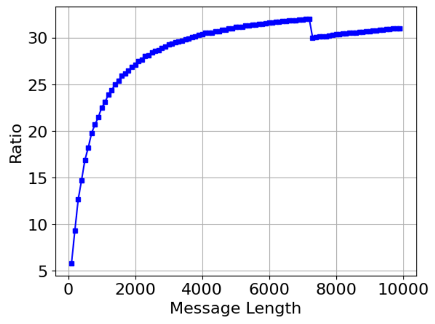


Fig. 7: Variation of Communication Delay (a) and Time Ratio (b) in Terms of Message Length with/without Brotli Compression.

3 Method and experiments

To verify which compression algorithm is most suitable for both compression and decompression, a comparison was made in terms of the compression ratio of the data and the compression/decompression time.

The comparison was made between these lossless compression algorithms in terms of data compression ratio (see Fig. 6a), data compression, and data decompression times (see Fig. 6b). In fact, the testing was done on real messages that *Blinky Blocks* can use. Therefore, one can conclude that even though Brotli does not have the fastest compression and decompression times, except that it achieves the best result when it comes to message size compression by reducing the original

Table 2: Numerical Example between different lossless compression algorithms using different message data sizes.

3D Model	Original Size (Bytes)	Huffman	Zstd	DEFLATE	Brotli	LZW	LZ4
Bunker	116	241	88	84	82	99	126
Tower	397	432	180	162	165	190	263
ADN Chain	3772	2252	288	266	229	253	443
Tank	3986	2577	569	491	444	507	1046

message size as seen in Table 2, and data by 55%. Thus, it appears to be the best lossless compression algorithm for both data compression and decompression time, as seen in Fig. 6b, and it is a suitable candidate for implementation with *Blinky Blocks*.
To be more precise, the compression process was performed only once on the master side to prepare data to be flooded into the network, while the decompression process was done on each *Blinky Block* at each computation of its colour.

The data transmitted in our application is used to describe a 3D configuration (i.e. the shape to be occupied by the set of robots). These vector data are used to determine whether or not a grid position (occupied by a *Blinky Block*) is inside this shape. Compression is performed once by the external server, and then the compressed message (size N_{comp}) is sent to all robots via a module connected to the server. The decompression process, made in each *Blinky Block* in parallel, can be carried out by a single traversal of all received data. Then, data is 'decoded' on the fly, without storing a decompressed version of the message. This results in a complexity decompression algorithm $O(N_{comp})$.

Despite Brotli having a high compression time, it also has the lowest decompression time. However, since we only need to compress the message once and decompress it every single time per *Blinky Block*, we found that Brotli seems to be the most ideal solution for this. Based on the obtained experimental results, we have shown how Brotli outperforms the other lossless compression algorithms in terms of compression ratio (as shown in Fig. 6a).
As a result, one can clearly deduce the effectiveness of the Brotli lossless compression algorithm in terms of both data compression and decompression and the reduction of message length. Thus, it proves to be a very effective method to mitigate the delay problem and effectively reduce its computation and execution time. Its appliance on *Blinky Blocks* comes as a novel solution for, to our knowledge, we are the first to propose applying lossless compression algorithms to a set of modular robots *Blinky Block* in terms of "Programmable Matter" and select the best one. Regarding Fig. 6a, the experimental validation of the given remarks was applied. On the left side, the communication delay of messages with different lengths is compared in both compressed (using Brotli) and original (non-compressed) versions. These graphs clearly show the gains made from the use of Brotli compression to transmit messages. The second graph (right), shows the link between the compressed

and non-compressed message lengths for different message sizes. This experience confirms that the gain is very important whenever we have a higher message length.

Therefore, Brotli is a versatile compression algorithm that offers excellent compression ratios, especially for *Blinky Blocks* messages, while still maintaining reasonable compression and decompression speeds. In fact, to further confirm the accuracy of our presented work, we tested it on *VisibleSim*, which is a software tool for simulating and programming modular robots (*Blinky Blocks*) and compared it with the already obtained results (see Fig. 6a and Fig. 6b) to show how close these results are and that the executed code remains the same wherever it is tested. Thus, it shows that the proposed algorithm has no compatibility or coding issues since it operates on the size of the message and not on the *Blinky Block* configuration.

After several tests on real data models and having it compared with other lossless compression algorithms in terms of compression/decompression time and compression ratio, Huffman will be replaced with the Brotli compression algorithm. Thus, offering the highest known compression ratio with far fewer compression and decompression times compared to Huffman.

Finally, we propose a more practical experiment to validate the complete process, consisting of compressing the 3D model, distributing the data code to a large set of connected *Blinky Blocks*, and decompressing many time the stored code in each *Blinky Block* to use the 3D data to set their colour. Figure 8 shows a picture of the setup of this experiment, also used to produce the video[1]. The setup shown on the left side of Fig. 8, includes a laptop connected to a grid of 768 *Blinky Blocks* (32×24). The laptop first sends the coordinates to each *Blinky Block* then sends the compressed model to the *Blinky Blocks*. At launch, the *Blinky Blocks* create a common coordinate system to obtain a position (cx, cy, cz) relative to the module in the lower left corner, by applying the algorithm proposed in [23]. The spanning tree created for this purpose will be used to distribute the code to all the blocks.
In this application we use a Huffman encoding algorithm which we ran on the laptop to create the code from the 3D model (the DNA model presented on the right side of Fig. 8), and sending it to one of the *Blinky Blocks*. To check that the data is well-received and uncompressed by each *Blinky Block*, after reception we repeat 60 rounds that compute the colour of an horizontal plane at level cz crossing the 3D scene.
At each stage, each *Blinky Block* analyses the encoding chain eight times to calculate the colour at eight different positions of the space inside the block. This method allows to create anti-aliasing effects. Positions are $(cx \pm 0.25 \times l, cy \pm 0.25 \times l, cz \pm 0.25 \times l)$ where l is the width the cubic *Blinky Block*. After half a second, each *Blinky Block* switches to the next stage by increasing its cz position by $0.25 \times l$ and recomputing a new colour.

[1] Video of Real time decompression on *Blinky Blocks*: https://youtu.be/xjAKxByAElI

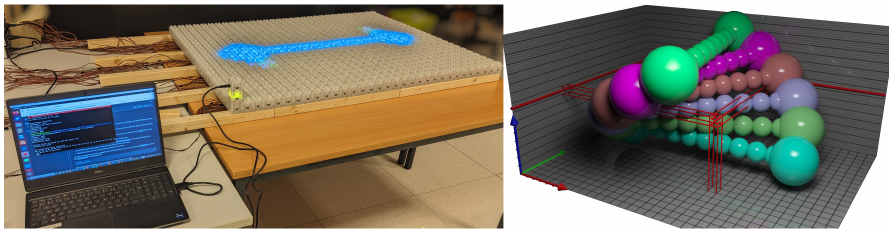


Fig. 8: An example of *Blinky Blocks* application with transmission and decompression of a 3D description model (the short DNA chain presented in the right picture).

4 Conclusion and Future Works

In this paper, we propose a study of the efficiency of a set of *Blinky Blocks* robots in terms of communication delay and computation time.

Based on the obtained results, we show that the communication delay linearly depends on the size of the message, and presented Huffman as a lossless compression algorithm as a novel method, which was substituted by Brotli as an ideal solution.

To reduce the communication delay, we propose to add a lossless compression algorithm. We compare a set of recent efficient algorithms with the Huffman coding method. We express the compression ratio and compression/decompression execution time for each of them. Moreover, the obtained results show that the Brotli algorithm requires the minimum overhead in terms of execution time and can achieve the maximum compression ratio. Therefore, this work indicates that the Brotli algorithm should be introduced at *Blinky Block* to reach a minimum communication delay.

In the future, this work will further extend to cover three main points:

- First, the adoption of Huffman as the first compression mechanism that can perform compression on *Blinky Blocks* proved to be a success. However, it cannot compress large messages within the accepted range of *Blinky Blocks*' message length, which varies from 1 to 227 bytes. Therefore, based on the presented results above, Brotli will be introduced as a successor to replace Huffman's compression.
- The constant integration of *Blinky Blocks* into the IoT domain [24, 25] and its interaction with different IoT devices will surely require not only textual data to be exchanged, but also audio, video, and even images. Therefore, other compression algorithms will be tested, depending on the changing nature of *Blinky Blocks* and the structure of the integrated data to reduce the communication delays between *Blinky Blocks*.
- Compression is surely an important mechanism to reduce communication delays. However, it is important to ensure that this communication is not intercepted by a malicious/non-malicious party. Therefore, a very lightweight cryptographic solution that takes into consideration the resource-constrained nature of *Blinky*

Blocks is required and will be integrated with the compression mechanism to ensure the first crypto-compression solution for *Blinky Blocks* that reduces delays and secures the communication by preventing the interception of the compressed messages.

ACKNOWLEDGMENT

This work has been supported by the EIPHI Graduate School (contract ”ANR-17-EURE-0002”).

References

1. J. Bourgeois, B. Piranda, A. Naz, N. Boillot, H. Mabed, D. Dhoutaut, T. Knychala Tucci, and H. Lakhlef, “Programmable matter as a cyber-physical conjugation,” in *IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016)*, IEEE. Budapest, Hungary: IEEE, oct 2016, pp. 002 942 – 002 947. [Online]. Available: https://publiweb.femto-st.fr/tntnet/entries/13257/documents/author/data
2. M. Rubenstein, A. Cornejo, and R. Nagpal, “Programmable self-assembly in a thousand-robot swarm,” *Science*, vol. 345, no. 6198, pp. 795–799, 2014.
3. K. Støy and R. Nagpal, “Self-Reconfiguration Using Directed Growth,” in *Distributed Autonomous Robotic Systems 6*, 2007, pp. 3–12.
4. J. Bassil, B. Piranda, A. Makhoul, and J. Bourgeois, “Repost: Distributed self-reconfiguration algorithm for modular robots based on porous structure,” in *IEEE RSJ International Conference on Intelligent Robots and Systems (IROS 2022)*, Kyoto, Japan, oct 2022.
5. T. Knychala Tucci, B. Piranda, and J. Bourgeois, “Efficient scene encoding for programmable matter self-reconfiguration algorithms,” in *32nd Annual Symposium on Applied Computing (SAC 2017)*, ser. ACM International Conference Proceedings, vol. Morroco, Marrakesh, Marrakech, Morocco, apr 2017, pp. 256 – 261. [Online]. Available: https://publiweb.femto-st.fr/tntnet/entries/13417/documents/author/data
6. J.-P. A. Yaacoub, H. N. Noura, and B. Piranda, “The internet of modular robotic things: Issues, limitations, challenges, & solutions,” *Internet of Things*, p. 100886, 2023.
7. J.-P. A. Yaacoub, H. N. Noura, and O. Salman, “Security of federated learning with iot systems: Issues, limitations, challenges, and solutions,” *Internet of Things and Cyber-Physical Systems*, 2023.
8. J.-P. A. Yaacoub, H. N. Noura, O. Salman, and A. Chehab, “Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations,” *International Journal of Information Security*, pp. 1–44, 2022.
9. J.-P. Yaacoub, H. Noura, O. Salman, and A. Chehab, “Security analysis of drones systems: Attacks, limitations, and recommendations,” *Internet of Things*, vol. 11, p. 100218, 2020.
10. E. F. Tagne, H. M. Kamdjou, A. E. Amraoui, and A. Nzeukou, “A lossless distributed data compression and aggregation methods for low resources wireless sensors platforms,” *Wireless Personal Communications*, vol. 128, no. 1, pp. 621–643, 2023.
11. R. Banerjee and S. Das Bit, “An energy efficient image compression scheme for wireless multimedia sensor network using curve fitting technique,” *Wireless Networks*, vol. 25, pp. 167–183, 2019.
12. F. Chen, A. P. Chandrakasan, and V. M. Stojanovic, “Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors,” *IEEE journal of solid-state circuits*, vol. 47, no. 3, pp. 744–756, 2012.

13. C. J. Deepu, C.-H. Heng, and Y. Lian, “A hybrid data compression scheme for power reduction in wireless sensors for iot,” *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 2, pp. 245–254, 2016.
14. Y. Peng, G. Carichner, Y. Kim, L.-Y. Chen, R. Tribhout, B. Piranda, J. Bourgeois, D. Blaauw, and D. Sylvester, “A high-voltage generator and multiplexer for electrostatic actuation in programmable matter,” *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 915–928, 2023.
15. J. Alakuijala, E. Kliuchnikov, Z. Szabadka, and L. Vandevenne, “Comparison of brotli, deflate, zopfli, lzma, lzham and bzip2 compression algorithms,” *Google Inc*, pp. 1–6, 2015.
16. S. Oswal, A. Singh, and K. Kumari, “Deflate compression algorithm,” *International Journal of Engineering Research and General Science*, vol. 4, no. 1, pp. 430–436, 2016.
17. J. Ziv, “The universal lz77 compression algorithm is essentially optimal for individual finite-length *n*-blocks,” *IEEE transactions on information theory*, vol. 55, no. 5, pp. 1941–1944, 2009.
18. H. Dheemanth, “Lzw data compression,” *American Journal of Engineering Research*, vol. 3, no. 2, pp. 22–26, 2014.
19. J. Alakuijala, A. Farruggia, P. Ferragina, E. Kliuchnikov, R. Obryk, Z. Szabadka, and L. Vandevenne, “Brotli: A general-purpose data compressor,” *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 1, pp. 1–30, 2018.
20. Y. Collet and M. S. Kucherawy, “Zstandard compression and the application/zstd media type,” *RFC*, vol. 8878, pp. 1–45, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52962805
21. S. Büttrich, “3d modeling with openscad-part 1,” *LOW-COST 3D PRINTING*, p. 83, 2018.
22. P. Thalamy, B. Piranda, A. Naz, and J. Bourgeois, “Visiblesim: A behavioral simulation framework for lattice modular robots,” *Robotics and Autonomous Systems*, p. 103913, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889021001986
23. B. Piranda, F. Lassabe, and J. Bourgeois, “Disco: A multiagent 3d coordinate system for lattice based modular self-reconfigurable robots,” in *IEEE International Conference on Robotics and Automation (ICRA 2023)*, London, England, may 2023.
24. J.-P. A. Yaacoub, M. Noura, H. N. Noura, O. Salman, E. Yaacoub, R. Couturier, and A. Chehab, “Securing internet of medical things systems: Limitations, issues and recommendations,” *Future Generation Computer Systems*, vol. 105, pp. 581–606, 2020.
25. J.-P. A. Yaacoub, O. Salman, H. N. Noura, N. Kaaniche, A. Chehab, and M. Malli, “Cyber-physical systems security: Limitations, issues and future trends,” *Microprocessors and microsystems*, vol. 77, p. 103201, 2020.