SeedLight: Hardening LED-to-Camera Communication with Random Linear Coding

Alexis Duque, Razvan Stanica, Hervé Rivano, Adrien Desportes

SeedLight: Hardening LED-to-Camera Communication with Random Linear Coding

Alexis Duque
Rtone, Lyon, France
Univ Lyon, INSA Lyon, Inria, CITI
F-69621, Villeurbanne, France
alexis.duque@insa-lyon.fr

Razvan Stanica
Univ Lyon, INSA Lyon, Inria, CITI
F-69621, Villeurbanne, France
razvan.stanica@insa-lyon.fr

Herve Rivano
Univ Lyon, Inria, INSA Lyon, CITI
F-69621, Villeurbanne, France
herve.rivano@inria.fr

Adrien Desportes
Rtone
Lyon, France
adrien@rtone.fr

ABSTRACT

With the increasing consumer demand for smart objects, LED-to-Camera communication appears as a low-cost alternative to radio to make any conventional device smart. Since LEDs are already on most electronics devices, that is achieved at the cost of negligible hardware modifications. However, as these LEDs are very different from the widely studied ceiling ones, several challenges need to be addressed to make this happen. Among these issues, we note the constrained physical layer data unit (PHY-SDU) length that complicates the use of coding strategies to cope with bits or packets erasure.

To break this limitation, this paper presents *SeedLight*, a coding scheme designed to face the inherent packet losses and enhance line-of-sight LED-to-Camera communication goodput. *SeedLight* leverages random linear coding to provide an efficient redundancy mechanism that works even on PHY-SDU of tens of bits. The key idea of *SeedLight* is to reduce the code overhead by replacing the usual coding coefficients by a seed. Since this work addresses IoT devices with low computational resources, *SeedLight* encoding algorithm complexity remains low.

We develop an implementation of *SeedLight* on a low-cost MCU and a smartphone to evaluate both the communication and algorithmic performances. Experimental results show that *SeedLight* introduces a negligible overhead and can be implemented even on the cheapest MCU, such as the ones used in many IoT devices. The achievable goodput can be up to 2.5kbps, while the gain compared to a trivial retransmissions scheme is up to 100%.

KEYWORDS

VLC; IoT; LED-to-camera; Random Linear Coding

1 INTRODUCTION

With the rise of the IoT, many technological solutions for various applications such as smart consumer electronics, smart lock, smart industry are proposed. Most IoT objects use wireless short-range communication based on radio technologies like NFC or Bluetooth Low Energy, when the compatibility with modern smartphones is a requirement. Nonetheless, these solutions induce non-negligible manufacturing costs and hardware modifications. Recent advances have enabled an alternative to radio, compatible with off-the-shelf smartphones based on Visible Light Communication (VLC) [2]. This solution also called Optical Camera Communication (OCC) takes advantage of the mobile phone camera to offer short range, low data rate communication. By modulating the LED in the 1-10KHz bandwidth and configuring appropriately the camera, a series of dark and bright strips known as the rolling shutter effect artifact appears in the picture and is used to encode information. With this method, a throughput of up to 3kbps can be achieved [21]. Nonetheless, the signal obtained in such way is not continuous since the camera pauses briefly after capturing a frame. During this inter-frame gap (IFG), no signal is captured, inducing packet losses.

OCC channels are mainly categorized as line-of-sight (LOS), when the camera aims directly at the LED, and NON-LOS, when the camera captures the light reflected from a surface illuminated by the emitter. With a NON-LOS channel, the region of interest (ROI), i.e. the part of a picture that contains the rolling shutter stripes, fills all the picture [3, 5]. In such case, bit losses occur only during the IFG and packets for which transmission lasts as long as the camera frame duration can be designed. With a LOS-Channel, the ROI is only a part of the picture and decreases rapidly with the distance between the LED and the camera [4, 10, 21]. Losses are so much larger and intra-frame losses are added to these caused by the IFG. This highly constrains the PHY-SDU length since the PHY-SDU must fit into the ROI to be received.

This paper follows our preliminary work [4] which introduces a LOS LED-to-Camera system that broadcasts the value of a sensor through a cheap color LED put on a small printed circuit board. To get around the issues discussed above, the system in [4] basically retransmits continuously packets. This leads to useless transmissions and makes the sending of large messages arduous. Therefore, this paper focuses on the proposal of an appropriate mechanism to overcome these drawbacks by developing a coding scheme based on random linear coding (RLC) to improve the goodput and reduce the number of retransmissions in LED-to-Camera broadcast.

Several similar approaches are presented in Sec. 2 below but our work is distinguished through the following contributions: (1) We provide the first RLC coding scheme optimized for VLC and OCC. (2) We reduce the RLC overhead by using a pseudo-random linear coding (PRLC) method. (3) The code complexity enables its implementation on cheap MCUs.

We provide practical engineering insights and conduct experimental evaluation on low-end IoT devices putting in evidence two points: (1) The goodput is up to 100% higher than a trivial retransmission approach. (2) A carefully considered coding scheme implementation does not affect the communication performance, even on the MCUs used in IoT devices.

The rest of this paper is organized as follow. Sec. 2 presents previous works and spotlights the remaining challenges this work addresses. We then detail the coding scheme that is the main contribution of this paper in Sec. 3, before evaluating its performances by experimentations in Sec. 4. Sec. 5 focuses on the algorithm implementation and its performances evaluation on low-end IoT devices before concluding in Sec. 6.

2 RELATED WORK

To overcome the loss and asynchronous communications between emitter and receiver in LED-to-Camera broadcast, several methods have been studied in the literature.

[5] proposes to repeat consecutively every packets R times to ensure that each of them can be captured, regardless of the misalignment between the LED and the camera. It is clear that this scheme leads a reduced data rate. Moreover, the smartphone must receive all the packets independently to decode the frame. A single missing or erroneous packet hence imposes to wait for a whole frame retransmission. More recently, [17] proposes a combination of Hamming codes and bit interleaving to avoid packet retransmission. Similarly, [9] uses Reed-Solomon encoding to recover both IFG losses and inter-symbol interferences. These approaches assume that missing bits occur only during the IFG, and that the whole picture contains information. These all require a channel estimation, to set the appropriate number of retransmissions R, Hamming blocks or redundancy bits. As a consequence, this makes them unsuitable for LOS scenario, where the bit loss highly increases with the distance between the emitter and the receiver.

Recovering the loss in LOS channel is more challenging. Fountains codes [13] have been employed to improve the data rate and the link reliability in LED-to-Camera broadcast. [21] leverages Raptor Code [18] and proposes an experimental evaluation using ceiling LEDs. Their approach reduces the latency at the cost of 25% overhead. In a close way, [3] studies LT-code[11] and proves their implementation can reach 1.6kbps of goodput. Nonetheless, the overhead induced by these approaches would be unacceptably large for the extremely small PHY-SDU size we use. Another flaw of LT-code and Raptor code is the coding complexity at the emitter side that [3, 21] set aside performing the encoding on a computer.

Another class of rateless codes called RLC are widely used in the context of network coding (NC)[16]. However, RLC has never been proposed yet for VLC. RLC can be applied within a NC framework but also as a good packet-level error correction solution: they are simple to implement at the emitter side and perform as optimal erasure codes for sufficiently large finite field used for creating linear combinations of source symbols. The major obstacle in the application of RLC is the decoding complexity of Gaussian elimination decoding. In [15] the authors compare RLC and Raptor code performances showing that, for moderate loss values, the performance of both codes is almost the same, but with higher loss rates, RLC has a better performance.

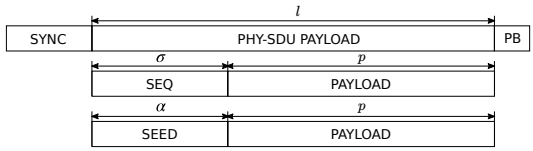

Figure 1: Packet sent without coding (middle) and chunck format generated by PRLC (bottom) encapsulated into a PHY-SDU (top).

3 SEEDLIGHT DESIGN

These solutions already proposed for VLC in the literature can hardly apply to our system since we use small color LEDs that constrain the PHY-SDU length, as explained in Sec. 1. Also, implementing these codes on a low-resources MCU, as those we use is not feasible. Therefore, this section presents a coding scheme that overcomes these limitations and improves LED-to-Camera reliability and goodput compared to a systematic retransmission method.

SeedLight implements a Pseudo-RLC (PRLC) and considers the following requirements: (a) the code overhead must remain as low as possible, (b) the complexity should be low on the emitter side, since the end-devices are low-cost embedded devices with limited computational, memory and energy capabilities, (c) application layer message size cannot be larger than a few hundreds of bytes as consequence of (b).

We begin introducing RLC that is the root of *SeedLight* and few notations used in this paper, before proposing our alternative implementation to reduce the code overhead.

3.1 Random Linear Coding

We assume that an emitter Tx wants to send a message of G bytes to a receiver Rx, split into N blocks $b_1, \ldots, b_N$ that will be encapsulated in a PHY-SDU with payload size p. In the RLC terminology, G is usually termed as the generation size and chunks designate the coded blocks. RLC performs the coding over blocks using a Galois Field $GF(q)$ where $q = 2^\gamma, \gamma \in \mathbb{Z}^+$ is the field size. GF is a key parameter in RLC since it significantly impacts the code performance: a small field size will result in linear dependent chunks, lower overhead and lower computation cost. Because linearly dependent chunks contain the same information, their number must remain as low as possible. Also, [12] demonstrated that a field size of 2^8 is large enough to have a nearly non-linear dependency.

Following the RLC method further detailed in [16], a set of N coefficients of γ bits $c = (c_1, \ldots, c_N), c_i \in GF(q)$, is used to generate the coding vector v by linearly combining uncoded blocks as follow:

$$v = \sum_{i=1}^{N} (c_i \times b_i) \tag{1}$$

For successfully decoding and solving the linear equations, Rx must know the coding coefficient c for each coding vector v. This

is usually achieved by building a chunk $C = [c, v]$ appending c to v, which results in an overhead of $N * \gamma$ bits. Considering GF(256), a 50B message and a 2B payload, this adds an overhead of 25×8 bits coefficients for each chunk. That does not fit into a PHY-SDU which size l is only 24 bits.

3.2 Compressing the coding vector

To avoid transmitting the coding coefficients c, our method is influenced by [19] and uses a seed of a pseudo-random generator (PRNG) to determine c. Our PRLC scheme works as follows.

Encoding: For each chunk C, E randomly generates a seed $s \in [0 \dots 2^{\alpha}]$, where α is the seed length.

Then, a Mersenne-Twister PRNG that follows the implementation proposed in [14] and initialized with s is used to produce the N coefficients $c_1, \dots, c_N$.

Since $c_i \in GF(q)$ and $\gamma \leq \alpha$, c_i is built with the γ lowest significant bits of the PRNG results. The coding vector v is then computed using Eq. (1) and sent with the seed s in a single chunk $C = [s, v]$, following the format shown on the bottom of Fig. 1.

In this way, the overhead remains low: only α bits to describe the seed, which replaces a sequence number no further needed with *SeedLight* approach.

Decoding: The decoding step on the receiver side is straightforward and works iteratively. Each time a chunk is received, *Rx* reconstructs c using the seed s included in C, using the same PRNG described above. Then, c and v are added to the decoding unit matrix and used to solve the linear equations by Gaussian elimination : $\begin{bmatrix} c & v \end{bmatrix} \begin{bmatrix} c^{-1} \end{bmatrix} = \begin{bmatrix} 1 & b \end{bmatrix}$.

Rx can then successfully decodes G if it has received at least N linear independent packets. Complexity and implementation are discussed in Sec. 5.

4 EXPERIMENTAL EVALUATION

In this section, we evaluate experimentally and present the achievable performance of *SeedLight* LED-to-camera broadcast under realistic conditions and compare this with a conventional retransmission method. We mainly focus on three aspects: 1) the coding parameters, namely the seed length α and the generation size G, 2) the distance between Tx and Rx and 3) the PHY-SDU length.

4.1 Testbed

We use a slightly modified version of the testbed we described in [4]. We implement the VLC emitter on the low cost STM32L051, an MCU similar to those already integrated in most household appliances. Following the recommendations of previous studies [2, 5], an OOK modulation scheme is used. We consider a symbol rate of 8KHz, which is suitable for receivers using the rolling shutter effect [2]. To avoid any flickering effect, we use the Manchester coding proposed in the literature [2, 5]. The signal is received by the smartphone camera and decoded by an Android application we have developed. We use an LG Nexus 5 smartphone running the unmodified Android Marshmallow version number 6.0.1 which 8 megapixels 1080p CMOS sensor can capture up to 30 frames per second. Our Android application sets up the camera parameters to observe the rolling shutter effect produced by the modulating LED, based on the work in [5]. The information is divided and transported

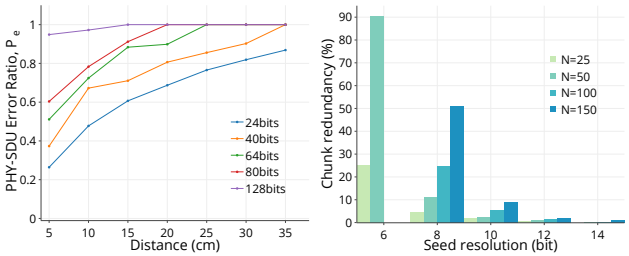

Figure 2: PHY-SDU error probability vs the distance (left). Chunk redundancy introduced by duplicated chunks vs the seed length and N (right).

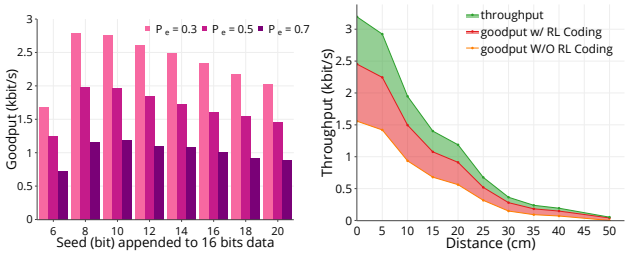

Figure 3: Goodput with G=100B for some PHY-SDU error probability vs the seed length (left). Throughput compared with the goodput vs the distance, with and without coding for G=100B (right).

in PHY-SDUs with a size between 24 to 128 bits, following the structure shown in Fig.1.

4.2 PHY-SDU size

An important property of the LED-to-camera communication is the fact that information losses are unavoidable and almost regular. Indeed, even when functioning at the maximum frame rate of 30 frames per second, the pictures taken by the smartphone do not represent a continuous view of the LED, as of the IFG. To cope with these losses, the data needs to be segmented into packets, which can be easily reassembled and retransmitted. However, the size of the packet has a significant impact on the packet loss probability. This is shown in the left plot in Fig. 2, for a packet size varying between 24 bits and 128 bits. We notice that the packet loss ratio becomes unmanageable as the packet size increases, e.g. almost all the 128-bits packets are lost, even at a few centimeters from the LED. Even for small 24-bits packets, the loss ratio is important (around 25% close to the emitter); this is the consequence of the packets that are transmitted during the IFG, which do not or partially appear in any picture. These results expose that the seed length and the overhead ratio have to be carefully designed.

4.3 Seed and Generation size

The results in Sec. 4.2 indicate that the PHY-SDU size is roughly limited. As a consequence, the seed length is a major influence factor of *SeedLight* performance.

Therefore, we evaluate the impact of the seed length (α) on the ratio of linearly dependent chunks produced when the minimum number of blocks N, required to recover the message, increases. Results in the right plot in Fig. 2 highlight that, to get less than 10% duplicated chunks, 8 bits are necessary for $N = 50$ while 10 bits are needed for N between 100 and 150. We also see that less than 8 bits are not enough, even when N is small. The duplicate ratio become negligible for seeds larger than 10 bits and for values of N higher than 100.

Left plot in Fig. 3 shows the goodput when varying the seed length for a PHY-SDU error rate P_e of 0.3, 0.5 and 0.7, respectively at 5, 10 and 20 centimeter distance, taking a generation size $G = 100B$. The best goodput is always obtained with a 8 bits seed and decreases progressively when the seed length increases. This is mostly because the overhead rises, even if the amount of linearly dependent blocks highly reduces. As of the duplicated chunks ratio with a 6 bits seed is larger than that of 8 bits, the goodput is in such case lower.

4.4 Communication goodput

We now evaluate the LED-to-camera communication performance as a function of the distance between the LED and the smartphone. We set the emitter symbol rate to 8KHz and place it in standard indoor illumination conditions, near a window and illuminated with neon lights. The illuminance has been measured with a luxmeter around 650 lux. The PHY-SDU size is 24 bits and is build as depicted in Fig. 1. The emitter broadcasts continuously a 100B message, which is transformed with *SeedLight* in a set of chunks, built with a 8 bits seed or, without *SeedLight*, in 50 different packets containing a 8 bits sequence number. For the case with *SeedLight*, the encoding of the set of such is done before the transmission starts. Thus, the computation does not affect the communication performance.

We compute the throughput as the number of received bits per second, error free, including duplicated packets or linearly dependent chunks. The goodput is calculated by removing duplicates and transmitting until the smartphone receives all the 100B message.

Right plot in Fig. 3 compares the throughput and the goodput for transmissions with and without the benefits of *SeedLight*. The 3 curves follow the same tendency: the throughput drops from 3kbps at 5cm to less than 0.2kbps at 40cm. We notice two sharp decrease, respectively at 5cm and 15cm. From 0 to near 5cm, the LED lightens the full surface of the CMOS sensor and the loss is mainly due to the SNR decrease and the blooming effect [1]. The second, between 15 to 20cm, is less straightforward and we make few assumptions to explain the phenomena. First, at this distance, the ambient light, that has been measured at 650 lux, starts interfering with the LED signal. Then, the ROI becomes small enough that no more than one packet is received at once and a non negligible amount of frames do not contain any information. The red curve shows that, at 5cm, the goodput with *SeedLight* is 2.25kbps while it is 1.4kbps without, bringing out a gain of 60%. Also, we can see that the relative gain grows with the distance: a gain of 90% is noticed at distance of 30cm.

Method	Operation	Mem.
(1) Mult. LT	1 LK	$2.2^{2\gamma}$
(2) Log. LT	3 LK + 2 BR + 1 MOD + 1 ADD	2.2^{γ}
(3) No LT	γ . (2 XOR + 2 SHIFT + 1 AND)	0

Table 1: Several multiplication implementations in GF(256). The operations are abbreviated LK for table lookup, BR for branch and MOD for modulus.

5 IMPLEMENTATION DISCUSSION

We show that *SeedLight* improves the LED-to-Camera transmission goodput compared to a systematic retransmission method. However, this is achieved at the cost of an additional coding step before transmission. Nonetheless, as explained in Sec. 3, *SeedLight* focuses on low-end IoT devices for which available resources are very limited. In such situation, software development and computation intensive algorithm optimization is complicated. Therefore, in this section, we discuss *SeedLight* implementation threats on aforesaid embedded platforms and evaluate the coding impact on two distinct MCUs and smartphones.

5.1 PRLC implementation on IoT devices

Several RLC algorithm benchmarks have been implemented previously. These studies were performed on computers [20], smartphones [6] or high-end embedded devices, at least able to run a UNIX operating system [8]. There is not implementation on low powerful platform like ours.

As shown in Sec. 3, PRLC relies on the widely used GF(256): its optimization has been discussed in [7], where the authors propose several solutions optimized for speed or memory. Tab. 1 shows the number of operations and memory requirement for three of those solutions that comply with our system. (3) requires the largest number of operations and will be restricted to cases in which the low available RAM forbids the use of lookup tables (LTs). (1) is the fastest method but its LTs are larger than our MCU flash size. (2) is well balanced between memory requirement and complexity for Tx. On the contrary, method (1) is preferred for the decoding considering the 1GB non-volatile memory available on the Nexus 5. According to Tab. 1 for (2) and the PRLC scheme proposed in Sec. 3, the code complexity to produce N chunks of P symbols is then $O(N^2P)$. In the same way, the complexity to decode a matrix of $N \times (N + P)$ bytes is given by $O(N^3 + N^2P)$.

5.2 Evaluation

We now evaluate the PRLC algorithm introduced in Sec. 3 and implementations discussed in Sec. 5.1 in our testbed. We benchmark the algorithm on the 8MHz Cortex-M0+ MCU and the more powerful Cortex-M4 with a clock configured at 50 and 100MHz. Both MCUs are often used in consumer IoT. For the receiver, we use an LG Nexus 5 with a Snapdragon 800 4-Cores 2,26 GHz CPU and the newer Huawei Nexus 6P with a Snapdragon 810 8-Cores 2.0 GHz CPU. The LG Nexus 5 runs Android Marshmallow version number 6.0.1 while the Nexus 6P runs Android Nougat version number 7.1.2. Both operating system are unmodified.

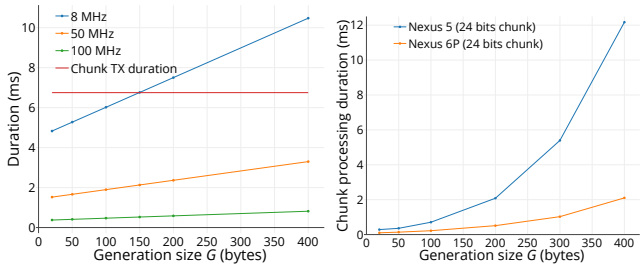

Figure 4: Computation time on a MCU to build 1 chunk (left). Computation time on the Nexus 5 and 6P to process 1 chunk and perform decoding (right).

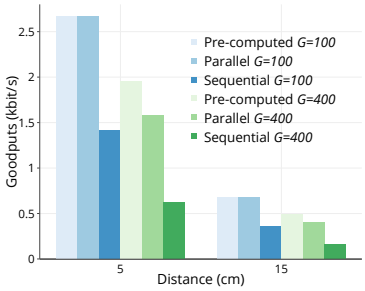

Figure 5: Goodput using different implementations for G=100B and G=400B.

Algorithm benchmark: Considering the optimizations envisaged in the previous section, Tx uses log and anti-log LTs to perform GF(256) arithmetic operations, while Rx uses a full multiplication LT.

The left plot in Fig.4 shows that the encoding computation time per chunk increases linearly with the generation size. For the 8MHz MCU (blue line), the time remains below the chunk transmission time when the generation size is smaller than 150B and is about 5ms with a 25B generation and it doubles when $G = 400$. This emphasizes that, with an appropriate mechanism, performing coding in parallel with modulating the LED is feasible, even for a low MCU frequency, when less than 150B are transmitted.

The right plot in Fig. 4 shows the decoding and Gaussian elimination computation time for each chunk. The evaluation was performed while the smartphone was capturing and processing camera frames using the VLC decoding algorithm introduced in Sec.4.1. Unlike the encoding algorithm, the increase is cubic. The benefits of the Nexus 6P is high and the chunk processing remains below 2ms even with a generation size of 400B. Since the camera frame rate is 30 fps and about 6 chunks are decoded in each, this shows the operations can be done in real-time. Also, further improvements are left pending, as developing the decoding algorithm in a C library to replace the current Java implementation.

MCU mode: The main functionality of consumer IoT MCU is not usually communication. This can be gathering data from sensors or performing main application logic algorithms. Radio transmission and signal generation can be realized by a dedicated chip but that is not always the case and, as a consequence, the MCU resources must be shared. In such cases, communication happens during reserved time slots or concurrently in a multi-tasking approach. To conform with these needs, we propose 3 scheduling strategies for PRLC and transmission: (1) **pre-computed** mode computes a set of chunks before transmitting them in a row, as in Sec. 4. This increases the needed RAM. (2) **sequential** mode computes a chunk and transmits it right away. This increases the delay, since the transmission is interrupted between each chunk. (3) **parallel** mode takes advantage of multi-tasking to compute a chunk when the previous one is transmitting. If the chunk computation time is larger than the transmission time, the transmission will stop during this time gap. This one has the highest impact on then CPU load. (1) is relevant when the data source, e.g. a sensor, stores the data immediately as encoded chunks, (2) when a prioritized task is running at the same time and (3) when the communication is prioritized regarding the main application task or when the MCU speed is hight enough to run both.

We now focus on the end-to-end goodput evaluation, considering these three implementations to depict the real use cases of IoT devices.

Fig. 5 shows the goodput transmitting a message of 100B (blue) and 400B (green) with *SeedLight* ($\alpha = 8$) using the software implementations discussed above. We place Tx and Rx at 5cm and 15cm distance. We note that the goodput falls as the message size increases, mainly because of the amount of linear dependent chunks transmitted. The plot highlights that, when working with a generation size of 100B, the chunk computation can be achieved in real time even at 8MHz and both transmission and chunk computation can be paralleled without throughput loss compared to the *Pre-computed* implementation. The *Sequential* mode for the 100B message is half the values of the others since it adds a delay to compute the chunk before its transmission. With a generation size of 400B, the goodput decreases between the *Pre-computed* and *Parallel* modes. This highlights that, in such case the MCU reaches its limit and a small gap between chunk transmission appears. Nonetheless, *SeedLight* still boosts the goodput set by the side with a systematic retransmissions scheme.

These results conform quiet well with Fig. 4 and bring out that *SeedLight* can be implemented even on the cheapest MCU with a minor impact on the communication performance. Furthermore, *SeedLight* will not increase the MCU power consumption, since this can be done without any modification on the MCU clock and during the same time interval the device modulates the LED. Moreover, since it reduces the number of transmissions that are necessary to receive the full message, *SeedLight* hence reduces the energy consumed.

6 CONCLUSIONS

This paper introduces *SeedLight*, a novel lightweight erasure coding method that leverages RLC theory to face the inevitable and large packet losses in LOS LED-to-Camera broadcast. The key idea of *SeedLight* is to take advantage of a PRNG to avoid the systematic transmission of the mandatory coding header introduced by RLC.

The experimental evaluation shows that this approach increases the goodput from 1.5 to 2.5kbps set side by side with a primary packet retransmission mechanism. Compared to the erasure coding

scheme recently proposed for OCC [3, 21], *SeedLight* introduces a much smaller overhead that makes it suitable when the PHY-SDU can not be larger than few tens of bits.

The other benefits of *SeedLight* is the low computational overhead on the emitter side. A careful implementation of *SeedLight* on low-cost MCUs like those often used in IoT devices and smarthpones brings out that it does not affect the communication performance and is compatible with such applications.

REFERENCES

[1] Chi-Wai Chow, Chung-Yen Chen, and Shih-Hao Chen. 2015. Enhancement of Signal Performance in LED Visible Light Communications Using Mobile Phone Camera. *IEEE Photonics J.* 7, 5 (oct 2015).

[2] Christos Danakis, Mostafa Afgani, Gordon Povey, Ian Underwood, and Harald Haas. 2012. Using a CMOS camera sensor for visible light communication. *2012 IEEE Globecom Work.* (2012).

[3] Haohua Du, Junze Han, Xuesi Jian, Taeho Jung, Cheng Bo, Yu Wang, and Xiang-Yang Li. 2017. Martian: Message Broadcast via LED Lights to Heterogeneous Smartphones. *IEEE J. Sel. Areas Commun.* 35, 5 (may 2017).

[4] Alexis Duque, Razvan Stanica, Herve Rivano, and Adrien Desportes. 2016. Unleashing the power of LED-to-camera communications for IoT devices. In *Proc. ACM VLCS '16.*

[5] Julia Ferrandiz-Lahuerta, Daniel Camps-Mur, and Josep Paradells-Aspas. 2015. A Reliable Asynchronous Protocol for VLC Communications Based on the Rolling Shutter Effect. In *2015 IEEE Glob. Commun. Conf.*

[6] Frank H.P. Fitzek, Morten V Pedersen, Janus Heide, and Muriel Medard. 2010. Network Coding: Applications and Implementations on Mobile Devices. In *Proc. ACM PM2HW2N '10.*

[7] Kevin M Greenan, Ethan L Miller, and S.J. Thomas J. E. Schwarz. 2008. Optimizing Galois Field Arithmetic for Diverse Processor Architectures and Applications. In *2008 IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*

[8] Néstor Hernández Marcano, Chres Sørensen, and Juan Cabrera G. 2016. On Goodput and Energy Measurements of Network Coding Schemes in the Raspberry Pi. *Electronics* 5, 4 (2016).

[9] Pengfei Hu, Parth H. Pathak, and Xiaotao Feng. 2015. ColorBars. In *Proc. 11th ACM Conex.'15.*

[10] Hui-Yu Lee, Hao-Min Lin, Yu-Lin Wei, and Hsin-I Wu. 2015. RollingLight. In *Proc. ACM MobiSys '15.*

[11] Michael Luby. 2002. LT codes. In *43rd Annu. IEEE Symp. Found. Comput. Sci. 2002. Proceedings.*

[12] Daniel E Lucani, Muriel Medard, and Milica Stojanovic. 2009. Random Linear Network Coding for Time-Division Duplexing: Field Size Considerations. In *IEEE GLOBECOM 2009.*

[13] D.J.C. MacKay. 2005. Fountain codes. *IEE Proc. - Commun.* 152, 6 (2005).

[14] Makoto Matsumoto and Takuji Nishimura. 1998. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.* 8, 1 (jan 1998).

[15] Sajid Nazir, Dejan Vukobratovic, and Vladimir Stankovic. 2011. Performance evaluation of Raptor and Random Linear Codes for H.264/AVC video transmission over DVB-H networks. In *2011 IEEE Int. Conf. Acoust. Speech Signal Process.*

[16] Dong Nguyen, Tuan Tran, Thinh Nguyen, and Bella Bose. 2009. Wireless broadcast using network coding. *IEEE Trans. Veh. Technol.* 58, 2 (2009).

[17] Duy Thong Nguyen and Youngil Park. 2017. Data rate enhancement of optical camera communications by compensating inter-frame gaps. *Opt. Commun.* 394 (jul 2017).

[18] Amin Shokrollahi. 2006. Raptor codes. *IEEE Trans. Inf. Theory* 52, 6 (2006).

[19] Nikolaos Thomos and Pascal Frossard. 2012. Toward One Symbol Network Coding Vectors. *IEEE Commun. Lett.* 16, 11 (nov 2012).

[20] Huanlai Xing, Rong Qu, Lin Bai, and Yuefeng Ji. 2014. On minimizing coding operations in network coding based multicast: an evolutionary algorithm. *Appl. Intell.* 41, 3 (oct 2014).

[21] Yanbing Yang, Jie Hao, and Jun Luo. 2017. CeilingTalk. *IEEE Trans. Mob. Comput.* April (2017).