Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 3430165, 10 pages
https://doi.org/10.1155/2019/3430165

Research Article

Secure and Smartphone-Assisted Reprogramming for Wireless Sensor Networks Based on Visible Light Communication

Jiefan Qiu,[1] **Chenglin Li**,[1] **and YueRan Li**[2]

[1]*Zhejiang University of Technology, College of Computer Science, Hangzhou, China*
[2]*National Intellectual Property Administration, Jiangsu Center, China*

Correspondence should be addressed to Jiefan Qiu; qiujiefan@zjut.edu.cn

Received 6 October 2018; Revised 15 January 2019; Accepted 24 February 2019; Published 14 March 2019

Academic Editor: Ning Zhang

During the period of over-the-air reprogramming, sensor nodes are easy to eavesdrop and even controlled by unauthorized person. That reminds us that security is key issue for over-the-air reprogramming. Most of previous studies discussed this problem from the aspect of data encryption, but give little consideration to the physical level. In this paper, we attempt to improve the security of reprogramming by changing the physical-level communication mode. We apply unidirectional Visible Light Communication (VLC) to the over-the-air reprogramming and use Commercial Off-The-Shelf device such as smartphone and sensor node to improve applicability. However, the unstable light source and low-cost light sensor make the procedure of reprogramming difficult. For this end, we put forward a novel reprogramming approach named ReVLC, which is twofold: firstly, we design a code block mechanism based on function similarity to reduce transmitting code. Secondly, we use compressing representation to optimize the Dual Header-Pulse Interval Modulation (DH-PIM) to save transmission time. The experiment results illustrate the effectiveness of ReVLC at the cost of extra 49.1% energy overhead compared with a traditional reprogramming approach.

1. Introduction

Wireless sensor networks (WSNs) systems as one kind of edge networks for IoT were always deployed in inaccessible areas for some long-term monitoring tasks. When software upgrades or even has malfunctions, the over-the-air reprogramming is the only way to online debugging or adjusting functionality without retrieving.

With respect to previous over-the-air reprogramming approaches, data transmitting is always completed by radio wave communication (RWC) and its frequency is from 10 KHz to 3000 GHz such as 2.4 GHz (Zigbee and Wi-Fi) or 800Mhz (GPRS), etc. Due to the broadcast nature of RWC, transmitting codes are vulnerable to eavesdropping and tamper. Especially, in the security-sensitive reprogramming case, some of codes or parameters are not suitable to be transmitted by broadcast mode, such as updating data encryption algorithm [1] or secret key [2, 3]. In fact, the previous studies [4–12] consider little in solving information leakage from physical level of communication. Meanwhile, Visible Light Communication (VLC) has unidirectional propagation that makes nonauthoritative person hard to eavesdrop and helps to reduce security risk. In addition, the reprogramming is applied in debugging the lost sensor node which cannot contact with other sensor nodes or sink node. For instance, our early relic-protection WSN system for Forbidden City Museum [13] was tested in laboratory and worked well. Then all nodes were locked in the glass showcase at least 3 months. Once malfunctions happen, debugging the lost node must be carried out by noncontact manner. Traditional RWC-based reprogramming approaches cannot debug the lost sensor node, but VLC-based reprogramming approach makes it possible.

For traditional VLC studies, the dedicated devices of light-signal sending and receiving are required. For example, Fan et al. improved the door control system based on VLC and the system needed a special photosensitive circuit [14]. Wang et al. designed an RGB-LED circuit to

Figure 1: Hardware interaction diagram: TelosB sensor node and smartphone.

increase the transmission rate of VLC [15]. In addition, Adiono et al. attempted to reduce the ambient light noise and develop a dedicated VLC receiver with analog filters [16]. However, the cost of dedicated light receiving device is hard to afford by low-cost sensor nodes. Thus, with respect to reprogramming such nodes, we also explore how to apply VLC in low-cost or Commercial Off-The-Shelf (COTS) devices.

Generally, ambient light sensor equipped by sensor node can sense the change of light strength and can be taken as a potential VLC signal receiver. In addition, the smartphone has potential as a VLC gateway, in the following twofold: (1) LED flashlight built-in smartphone is potential VLC signal transmitter shown in Figure 1; (2) the large memory space and transmission bandwidth guarantee caching the updating codes and forwarding them to sensor node at a little cost.

Using general hardware equipped by smartphone and low-cost sensor node will be very helpful in overcoming the barriers of applying VLC-based reprogramming to post-deployment WSN system. At the same time, using general hardware causes unstable performance of VLC, prolongs the reprogramming procedure, and even fails in reprogramming sensor node. For this end, we propose a novel reprogramming approach called ReVLC. It reduces transmission time of reprogramming from both sides: minimizing transferred data and optimizing modulation.

Previous reprogramming approaches adopted incremental strategy to minimize the transferred data. In these approaches, *delta* script [17] (consist of binary code differences and rebuilding operations) was transmitted. There approaches focus on how to improve the code similarity between the new image and old one, such as the following: Zephyr and Hermes [17, 18] improved the similarity between the old and new versions application by fixing the globe/static variable and function addresses; Li et al [19] designed an update-conscious complier to generate the new image based on the old ones and improve the similarity of both versions.

However, improving code similarity needs to change the program structure which increases difficulty on debug process. Further, such binary code differences are produced through comparing entire program image, but debugging sensor node usually is usually to fix several similar-functionality functions. Thus, we design a code block mechanism. In each block, we maintain function structure and put functionality-relevant functions in one code block. When we conduct reprogramming, these functions in same block are simultaneously replaced. By this way, only certain code blocks need to be transferred rather than entire program image.

We also present a new modulation named Compressing DH-PIM (CDH-PIM for short) based on the Dual Header-Pulse Interval Modulation (DH-PIM) [20]. DH-PIM is designed for VLC and close to the modulation efficiency of OOK. In CDH-PIM, we employ a compressing representation for repetitive bit series in modulation phase. The experiment results illustrate the transmission rate can be improved by about 17.2% compared with DH-PIM.

The rest of this paper is structured as follows: Section 2 shows relative work; Section 3 introduces the design and implementation of ReVLC; Section 4 describes experimental scenarios and evaluates our approach; and Section 5 closes with conclusion.

2. Related Work

Deluge [21] is the first and sophisticated reprogramming approach applied in wireless sensor networks, but the authors have not considered any security principle when designed it. Because Deluge needs to transmit the entire program image with a long period of reprogramming, sensor nodes are vulnerable to attack. Thus, the early secure reprogramming studies [3, 4] focus on the extensions to Deluge and guarantee data integrity and authenticity. For example, Sluice [3] employs signature and cryptographic hash function to generate the hash image of program image as authentication. It also follows Deluge to divide program image into pages and each page contains the hash image of next pages. Hyun et al [4] presented Seluge to optimize the transmission overhead for Markle hash tree and reduce the page propagation delays.

With secure techniques ongoing, researchers pay more attention to the general attack and excessive security overheads of reprogramming. Park et al [5] proposed a lost packet recovering method by supplementary hash scheme and page digest scheme. DART [6] is proposed by Dong et al. who used time-based authentication to defend pollution attacks and it needs time asymmetry which brings up additional delay. In Sreluge protocol [7], author solved the problem of authenticating data packets by encoded packet using random linear codes. Dong et al. [8] have presented two filtering method to protect from DoS attacks by signature verification. Tan et al. [9] proposed a reprogramming protocol which integrates confidentiality and Dos-attack-resistance countermeasures suitable to WSN. Kim et al. [10] designed three source authentication reprogramming schemes which support dynamic packet size. LDSCD [11]

is a social role-based distributed reprogramming framework against DoS attack and supports multiple authorized tenants. Yang et al. [12] considered excessive overheads for security mechanism and proposed a security enhancement reprogramming approach based on a hierarchical hash tree.

Above approaches considered the security problem of reprogramming by modifying upper-layer software and were not still to totally prevent from eavesdrops due to the broadcast nature of RWC. In this paper, we attempt to change physical-level communication mode and improve the security of reprogramming.

3. Design and Implementation of ReVLC

In this section, we firstly present code block mechanism based on function similarity and then introduce CDH-PIM applied in ReVLC. The former reduces the transmitting data of reprograming and the latter optimizes VLC modulation to make it applied in general devices. At last, we also describe the implementation of ReVLC.

3.1. Code Block Mechanism. The early RWC-based reprogramming approaches [21, 22] need to transfer whole program imager, but the size of program image is clearly overlarge for VLC-based reprogramming. In fact, the bugs exist in several functionality-relevant functions. Thus, the problem functions should be fixed not whole program image.

ReVLC provides a mechanism to divide the program image into several code blocks according into functionality. That means putting the functionality-relevant functions in one code block and transferring corresponding code blocks instead of whole image.

In our previous work [23], we have found that the functionality-relevant functions can be described by the function similarity between independent functions. The function similarity is defined based on an observation that the modification of function has unidirectionality, namely, that caller function will be changed with high possibility if the callee function has been changed; the opposite situation rarely happens. Thus, we firstly defining function similarity degree (FSD) to measure the function similarity based on the collaborative filtering algorithm. Given functions u and v, $N(u)$ and $N(v)$ are, respectively, the callee function in u and v. The FSD S_{uv} between u and v can be described by Jaccard formula:

$$s_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \tag{1}$$

After obtaining each S_{uv} between two functions, we calculate the whole function similarity degree (S) of code block with different weight, as follows:

$$S(f_1, f_2, \ldots, f_i) = \sum_{u=1}^{i} \sum_{v=1}^{i} \beta_{uv} s_{uv}$$
$$\beta_{uv} = \frac{1}{size(u) * size(v)} \tag{2}$$

where β_{uv} is weighted value, which is inversely proportional to the size of function u and v. It ensures that if u and v own smaller size with a high FSD, the whole FSD of code block is high; conversely, if u and v own large size with a high FSD, the whole FSD of code block should be reduced. By calculating the FSD, the program image can be divided into several code blocks.

On the other hand, each code blocks can be migrated to several sensor nodes with different address. The new address of each function is uncertain before reprogramming. If modifying each call instruction according new address, the rebuilding overhead is large and not afforded by sensor node. To simplify this process, we have adopted a register relative addressing to invoke each function in RePage.

In each code block, the `call` instruction must be modified to the register relative addressing and the real entry address of callee function is saved in Function Location Table (FLT) shown in Figure 2. In addition, a hash table (HT) needs maintaining in sensor nodes. The HT contains the real entry addresses of each function and locates in high address of program flash. FLT is fixed in the tail of each code block. It saves hash values and the offset address of each function relative to the code block.

The hash value is generated by hash function *Hash*(fun_name) where function name is taken as hash key. When a code block adds new function, FLT needs correspondingly adding one new item and grows forward the low address like stack behavior.

When a code blocks arrive in a sensor node, the runtime system would load code block with following 3 steps:

(1) Assign a new address to insert the code blocks in program flash and this address is the base address of functions contained in the new code block.

(2) Modify the first `mov` instruction located in the head of each function BY the begin address of HT in local node.

(3) Modify HT according into the hash value saved in FLT and the begin address of HT using the new function addresses, which is figured out by offset+base address.

Before calling the function, the begin address of HT should be assigned to a certain register, for instance, using register *r5* saves the begin address (0xBC00) illustrated in Figure 2. Therefore, in step (2), we must reassign *r5* by `mov` instruction located in the head of each function of new code block.

When a caller function call a callee function based on the register relative addressing, register *r5* combining with relative address which is hash value (0x0054) and point to the item of the HT which save the real entry address of function. Finally, realize the function invoking process.

In Section 3.3, we also discuss that modify `mov` instruction which reassigns *r*5 and the address as parts of each call instruction using hash value. Both of modifications happen in precompiling phase. Certainly, they are transparent to developer or networks owner.

3.2. Compressing Dual Header-Pulse Interval Modulation. Different from RWC, signal modulation of VLC depends on switching LED's state. The LED switch rate is faster than fluorescent and filament lamp, so smartphone equipped

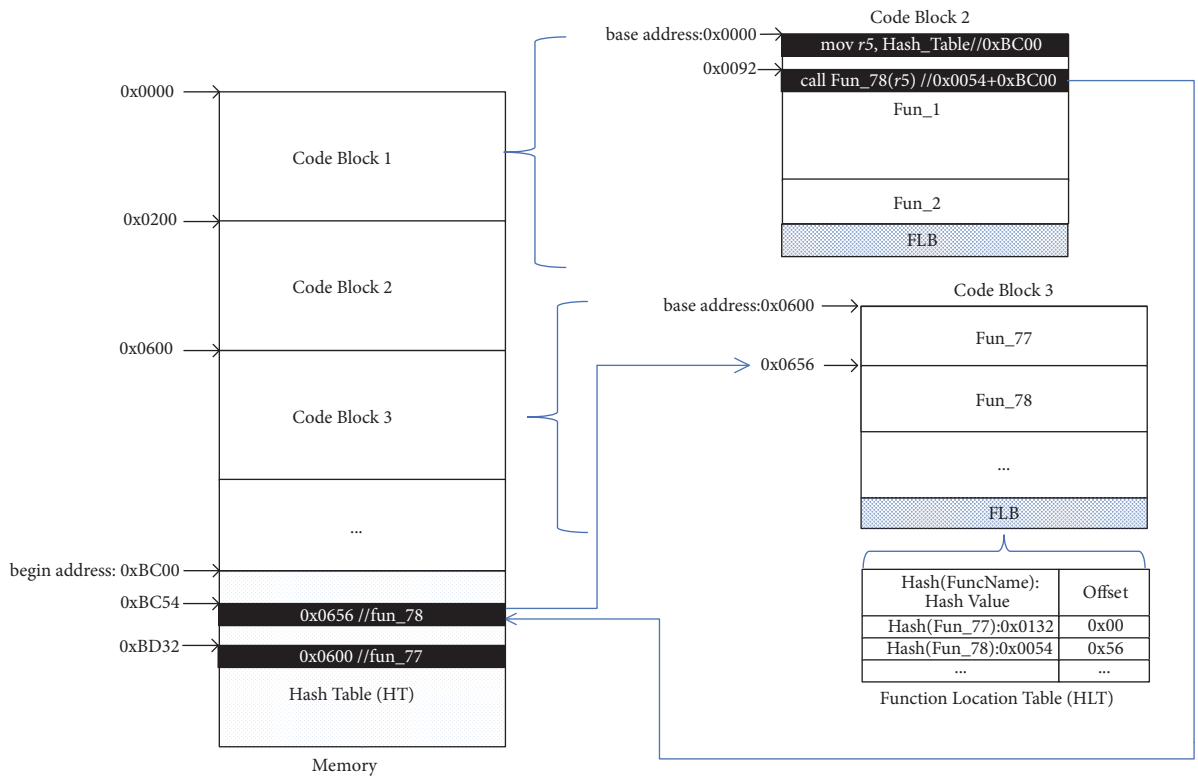


FIGURE 2: Memory layout and function calling process: the entry address of Fun_78 located in code block3 is 0x0656H and saved in hash table with address 0xBC54H. Invoking Fun_78 needs to obtain the hash table address (0xBC54), and then using function entry address (0x0656H) to complete invoking process.

with LED flashlight is a potential light-signal transmitter. However, the dedicated light-signal receivers are usually complex and hard to afford by low-cost sensor nodes. In order to realize VLC between sensor node and smartphone, we try to employ the ambient light sensor built-in sensor node as light-signal receiver.

However, using the ambient light sensor has an unappealing communication performance, because these sensors have a long response latency and opening time. Both of parameters decide the minimum detectable pulse width. We define the double width of minimum detectable pulse width as the one time slot t_r.

In addition, the reprogramming is a bursting procedure. Namely, several code blocks transmit within a short period. Above problems make it difficult that apply directly VLC to over-the-air reprograming.

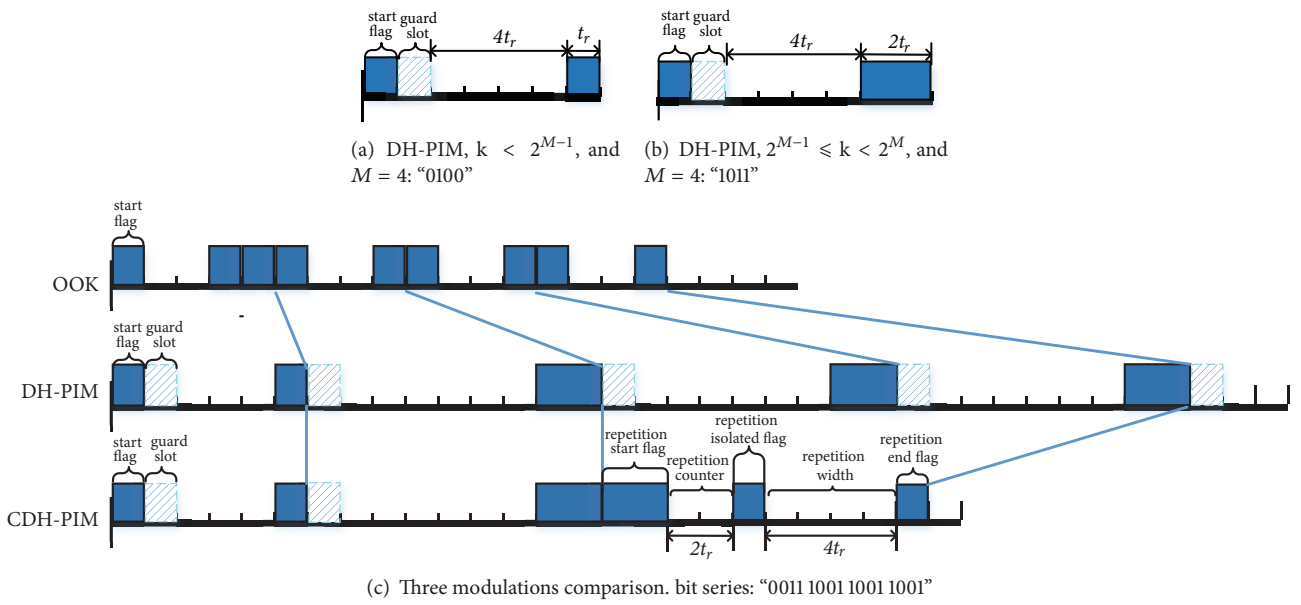
Based on large numbers of experiments, we found a lot of continuously repetitive parts (such as "1010 1010 1010") exist in the bit series of code block. This find lets us consider compressing the continuously repetitive parts and propose a Compressing DH-PIM (Dual Header-Pulse Interval Modulation). The CDH-PIM, for short, divides the continuous bit series to nonrepetitive and repetitive parts according to content.

For nonrepetitive parts, CDH-PIM adopts DH-PIM designed for VLC [20]. DH-PIM is an anisochronous pulse time modulation in which data are encoded as discrete time slots between adjacent pulses. A symbol which encodes M bits of data is represented by k slots of low power and followed by one pulse of constant power, where $0 \leqslant k \leqslant L/2\text{-}1$ and $L = 2^M$. The pulse of constant power will last t_r or $2*t_r$ for two numbers which are radix-minus-one complement. For example, the number "0100" has 4 slots of low power and 1 slot pulse of constant power. The number "1011" is radix-minus-one complement of "0100" and has 4 slots of low power and 2 slots pulse of constant power.

For repetitive parts, CDH-PIM adopts a compressing representation to reduce the number of slots. The repetitive part can be represented by three parts: first nonrepetition bits, repetition counter, and width. The latter two parts need special symbols to mark.

In practice, given M=4, the one slot pulse and two slot pulse were used to represent for $k<2^{M-1}$ and $2^{M-1} \leqslant k < 2^M$ shown in Figures 3(a) and 3(b). It needs 3 repetition flags in order to mark and identify the repetition counter and width as shown in Figure 3(c). The successive 2 slots pulses are repetition start flag and followed by the normal DH-PIM's 1 slot pulse and taken as start flag for the repetition counter. The next one pulse is repetition isolated flag to divide the repetition counter and width. The last one slot pulse is the repetition end flag of repetition width. The repetition counter and width are represented by the number of slots of zero power.

As illustrated in Figure 3(c), to represent the repetition part, it needs 4 slots pulses and 6 slots of zero power for expression, compared with 4 slots pulses and 14 slots of zero



(a) DH-PIM, k < 2^{M-1}, and M = 4: "0100"

(b) DH-PIM, $2^{M-1} \leqslant$ k < 2^M, and M = 4: "1011"

(c) Three modulations comparison. bit series: "0011 1001 1001 1001"

Figure 3: Modulation comparison: (1) OOK modulation; (2) Dual Header-Pulse Interval Modulation; (3) Compressing Dual Header-Pulse Interval Modulation.

Table 1: Function table.

Fun Name	Fun Size	Calling Relationship
Fun_2	96 B	Fun_78,Fun_1,...

power of DH-PIM. The transmission time reduces 44%. In such case, the 3 repetition flags need extra 4 slot pulse.

It should be noted that some cases cannot be applied compressing representation. For example, a bit series "0000 0000 0000" is modulated by DH-IPM and only needs 3 pulses and 3 slots of zero power compared with using CDH-PIM with 5 pulses and 7 slots of zero power. Thus, to deal with the repetitive part, the extra cost caused by tree types of repetition flags needs to be considered before modulation.

3.3. Implementation of ReVLC. We run ReVLC on a popular sensor network OS TinyOS. The running phase of ReVLC can be divided to three parts: the compilation phase (compile the program image), transmitting phase (transmitting code block by VLC), and the loading phase (deploy code block).

As shown in Figure 4(a), compilation phase occurs in smartphone or cloud. In ReVLC, it needs to perform twice compilations for code block. In the first compilation, we make a modification on dead code elimination of TinyOS and keep all the functions existing in image program file [24]. Then, establish the function table through reading the symbol table of executable (.exe). The function table comprises function name, function size, and the calling relationship which records all callee functions by current function. The function table is as shown in Table 1.

The function table is generated after the first compilation. Using the calling relationship, the function similarity degree (FSD) between functions can be calculated and saved in function table. In terms of FSD, ReVLC readjusts all positions of functions and puts functions into each code block. It also reserves space in each code block for FLT.

In the second compilation, the hash values are figured out by *Hash*(fun_name). The hash values are filled into the FLT. In the precompilation phase, we modify the addressing mode of calling function and add the register assignment instructions (`mov` instruction) in the head of each function. The source file needs compiling again according to the new readjusted function addresses.

Figure 4(b) shows transmitting and loading phase. Firstly, ReVLC figures out all the repetitive parts in the bit series of code block. Then evaluate and compare the cost of using compressing representation with not using. Then the modulated bit series are packaged into one or several frames and transmitted by VLC. In target sensor nodes, the transmitted bit series are demodulated.

Once code block arrives in sensor node, runtime system provides new spaces for the new code block. Then, it rewrites the local hash table in terms of the FLT of new code block and modifies the source operand (The begin address of HT) of the `mov` instruction in each function of the new code block. Finally, the runtime system starts software reboot procedure to restart target node.

4. Evaluation

In this section, we conduct a series of experiments in continuous updating cases. We firstly introduce the experiment methodology and then evaluate ReVLC.

4.1. Experiment Methodology. We implement ReVLC based on a Samsung N7100 smartphone powered by Android and TelosB sensor node [25] installed TinyOS-2.x. As shown in Figure 5, TelosB node equips with 2.4 GHz Zigbee RF

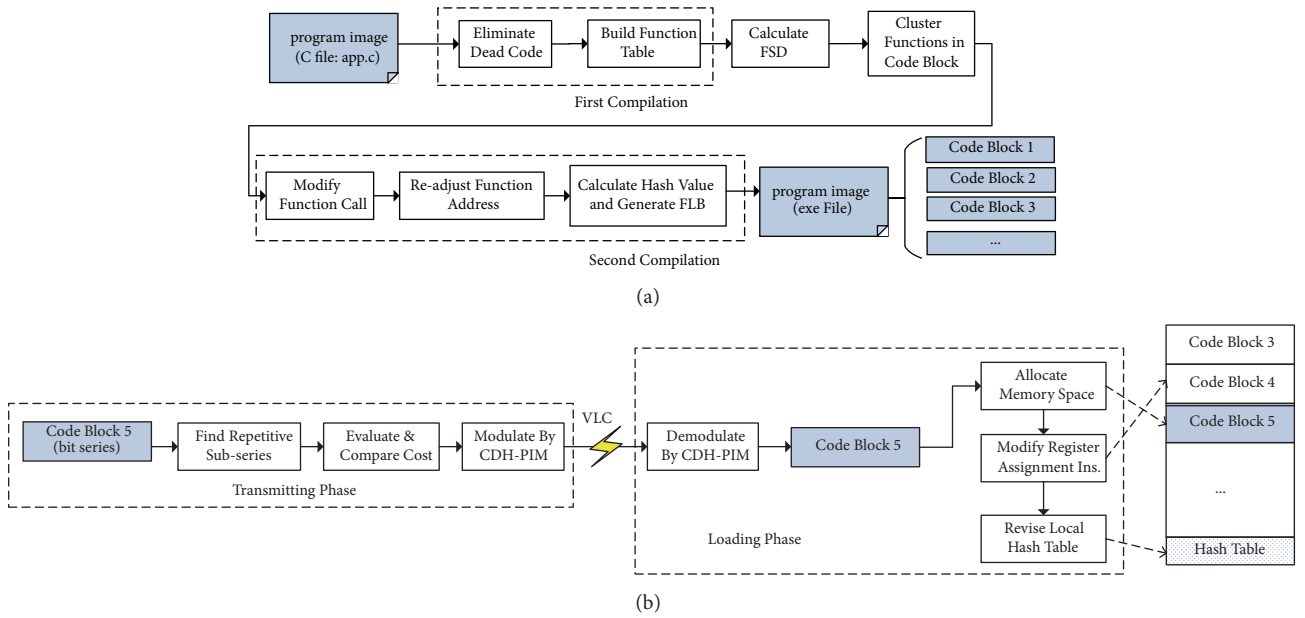


Figure 4: Implementation of ReVLC: (a) compilation phase; (b) transmitting and loading phase.

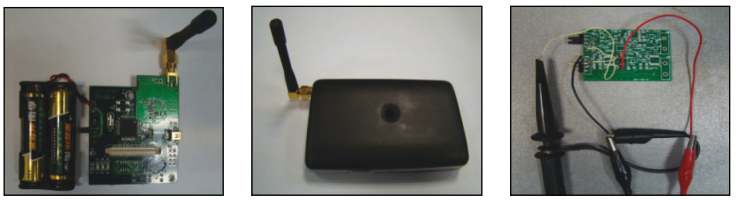


Figure 5: Sensor node and current-monitoring circuit: (a) TelosB sensor node; (b) packaged TelosB with light hole; (c) current-monitoring circuit with MAX9928.

transceiver CC2420, a 16-bit MCU MSP430F1611 (10kB RAM and 48kB program flash), and an ambient light sensor TSL2561. We evaluate the energy overhead and measure current change by a dedicated current-monitoring circuit equipped current-sense amplifiers MAX9928 [26] as shown in Figure 5(c).

4.1.1. Experimental Basic Setup

(1) The sensor node memory space: 1kB program flash (address: 0xFBFF-0xFFFF) for HT and 13kB program flash (address: 0xC708 to 0xFBFE) for ReVLC's runtime system.

(2) The max size of frame is 2048 bits, and size of header of each package is 30 bits, including start flag (2 bits), frame size (20 bits), and code block ID (8 bits).

(3) The size of code block is 256 bytes. FLT is located at the end of each code block. Each item of FLT is 5 bytes with hash value (3 bytes) and location information (2 bytes).

Updating Case Setup. We describe eight updating cases in the real world. In the first updating case, the target sensor node installed a standard routines program named Oscilloscope which is contained in TinyOS application library. The other five versions of programs are named EasiRoute (version from v0.1 to v0.3). They are developed for our relic-protection WSN system which has been deployed in the Forbidden Palace Museum [13]. Each version means a major upgrade. As shown in Figure 6, six programs are continuously updated according to an alphabetical order:

Updating Case A. Oscilloscope executes a standard data collection program. By adding a routing functionality, we update it to EasiRoute_v0. 1.

Updating Case B. Update EasiRoute from v0.1 to v0.11. After this update, a sensing-data storing module is added, and it realizes storing data in external flash chip.

Updating Case C. Directly update Oscilloscope to v0.11, and add routing and storing modules in target node.

Table 2: Comparison of transmitting data among ReVLC, RePage, and Tiny Module-link (Bytes).

	Case A	Case B	Case C	Case D	Case E	Case F	Case G	Case H
Tiny Module-link	876	225	656	296	1844	52	2678	2924
RePage	924	368	674	312	1766	52	2536	3014
ReVLC	1024	512	768	512	1536	256	2348	3328

Table 3: Comparison of memory space among ReVLC, RePage, and Tiny Module-link (Bytes).

		Case A	Case B	Case C	Case D	Case E	Case F	Case G	Case H
Tiny Module-link	Program Flash	14758	14796	15036	15720	15456	15456	18356	18704
	RAM	——	——	——	——	——	——	——	——
RePage	Program Flash	12256	12256	12256	12256	12256	12256	12256	12256
	RAM	5120	5120	5632	5632	5632	6144	6144	6144
ReVLC	Program Flash	13558	13558	13558	13558	13558	13558	13558	13558
	RAM	——	——	——	——	——	——	——	——

Updating Case D. Update museum monitoring program from v0.11 to v0.18. After the update, a bug will be fixed. This bug leads that sensor node to send data to sink node too often when museum is powered off at night.

Updating Case E. Update v0.11 to v0.21. We add a sleep mechanism to balance load based on the battery energy remaining.

Updating Case F. Update v0.18 to v0.21. The new version will remove the sensing-relevant functions from the node. It only retains routing function and forwards data from other ones.

Updating Case G. Update v0.18 to v0.3. After this update, node will install the UDP protocol which supports the fact that users directly interact with the nodes through web browser.

Updating Case H. Update v0.21 to v0.3.

We will test the performance of ReVLC in three aspects: transmission and memory overhead, rebuilding overhead, and the performance of VLC applied in reprogramming.

4.2. Transmission and Memory Overhead. Different from the incremental reprogramming approaches [17–19, 23, 27], transmitting data of ReVLC only consist of code blocks. We compare ReVLC with other incremental reprograming approaches such as RePage [23] and Tiny Module-link [27]. In RePage, program image is divided into each function page like the code block and modifies each page according to *delta* script. Tiny Module-link directly modifies function codes, but the updated functions need storing in tail of code segment (.text segment).

Table 2 gives details about transmitting data under 6 cases. Clearly, in six cases, the RePage and Tiny Module-link have advantage over ReVLC, because both of approaches can modify the image using smaller-size *delta* script.

For example, in case F, the *delta* script is mainly comprised by rebuilding operations to remove the sensing-relevant functions about RePage and Tiny Module-link. ReVLC still

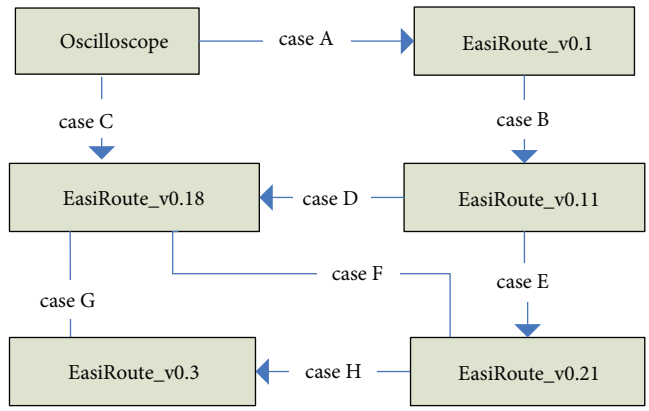


Figure 6: Roadmap of continuously updating.

transfers one code block at four times over both. But case C and case G are special. In both cases, the modified processes are complex. It means that several functions were changed, and *delta* script contains about 430 bytes of rebuilding operations. ReVLC only transmits new code blocks without these operations, and thus, its transmitting data are conversely less than the two others. Finally, the average of transmitting data of ReVLC is 7.12% and 6.2% higher than Tiny Module-link and RePage.

Memory overhead is mainly brought up by extra codes and the runtime system. As shown in Table 3, three runtime systems occupy 12256B, 13558B, and 14124B program flash space with respect to RePage, ReVLC, and Tiny Module-link. ReVLC's runtime system needs a demodulation module for CDH-PIM. Therefore, the size of ReVLC's runtime system is larger than RePage. In addition, due to writing external flash, the runtime system of Tiny Module-link needs 1.4 kB and its size is largest among three approaches.

With respect to extra codes, Tiny Module-link puts the size-increasing functions in tail of code segment (.text segment) without RAM participation, so the extra codes will be increasingly stored in program flash. On the other hands, RePage caches the frequently updating function pages

in low-power RAM to reduce rebuilding energy, so with reprograming ongoing, the cache area in RAM grows until fulling cache (6144 bytes). Conversely, the ReVLC directly replaces the old code block with new one stored in flash program. Therefore, there is no extra codes existing in RAM or program flash.

4.3. Rebuilding Overhead. The energy overhead of over-the-air reprogramming mainly comes from the code transmission and program rebuilding when read/write memory [28]. In fact, rebuilding overhead in volatile memory (such as RAM) is far less than that the nonvolatile memory (such as program flash). Take the TelosB nodes as example and the energy overhead of writing and reading (wt/rd for short) operations on the program flash is 14.6 times than on RAM. Therefore, when studying rebuilding overhead of over-the-air reprogramming, we focus on the number of wt/rd operations on program flash.

We compare ReVLC with RePage [23] under eight continuously updating scenarios. Like ReVLC, RePage adopts a paging mechanism to cluster the similar function, while it, as one incremental reprograming approach, only transfers the *delta* script between old and new function pages. RePage needs to store new page in cache (RAM) or program flash. Thus, in RePage, it necessary to write program flash to replace pages and read program flash to cache page in RAM or rebuild new page. Meanwhile, ReVLC transfers entire cold block and directly writes new block into program flash, so it is unnecessary to read program flash.

Figure 7 gives the number of wt/rd program flashes. In case A-case D, the performance of RePage is closed to ReVLC and the difference of wt/rd operations between RePage and ReVLC is 9.8%, because of no replacement happening. In case E, one-function pages are replaced in RePage. That means one page needs writing into program flash and the other one needs to reading from program flash. Such exchanging 512 bytes of function page brings up increasing 31.9% rebuilding overhead than ReVLC. In case F, using RePage, although only inserting or revising operations are fewest in all cases, the deleting operations lead to move codes to new position.

Meanwhile, in ReVLC, it only overwrites the old code blocks. In case G and H, several functions are inserted in program due to adding UDP protocol. In RePage, several function pages saved in program flash are cached in RAM and 2 pages replaced from cache. Finally, the wt/rd operations of RePage exceed 4.5 kB compared with 2.3kB and 3.3kB using ReVLC in case G and case H.

4.4. Performance of VLC Applied in Over-the-Air Reprogramming. In ReVLC, we optimize a current DH-PIM by bit-level compressing representation and propose Compressing DH-PIM (CDH-PIM). With respect to sensing hardware, the ambient light sensor TSL2561 is equipped by sensor nodes. Its minimum detectable pulse width is about 10ms. That means only the light pulse exceeds 10ms, and the sensor can stably measure light strength.

Therefore, the slots tr are set to 20ms and M=4 of CDH-PIM. Figures 8 and 9 show the completion time and transmission rate with DH-PIM and CDH-PIM in eight cases. The average transmission rate of CDH-PIM is 54.8 bit/s compared with 45.4 bit/s of DH-PIM. The completion time of CDH-PIM also reduces 16.3% than DH-PIM. This also suggests that a lot of bit continuous repetitions exist in transferred code blocks.

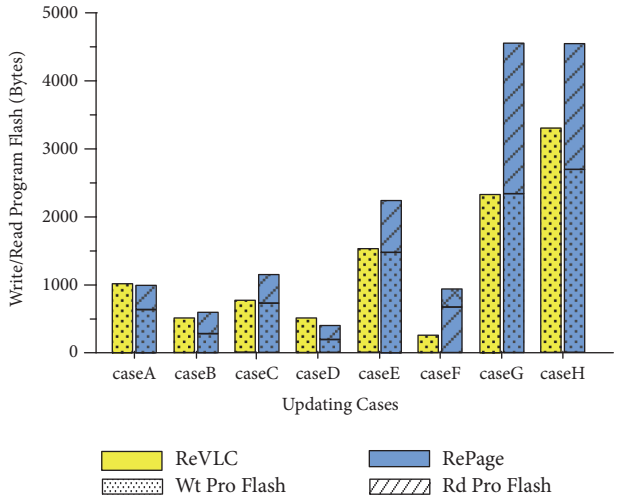


Figure 7: Rebuilding overhead.

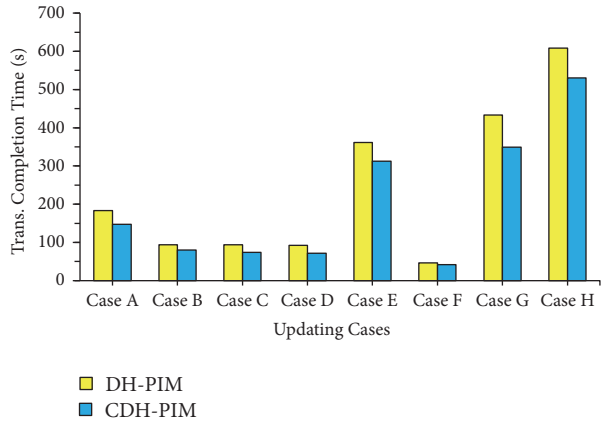


Figure 8: Transmission completion time (s).

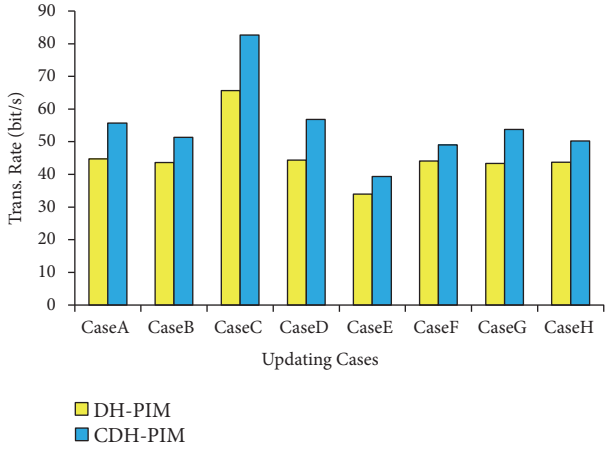


Figure 9: Transmission rate (bit/s).

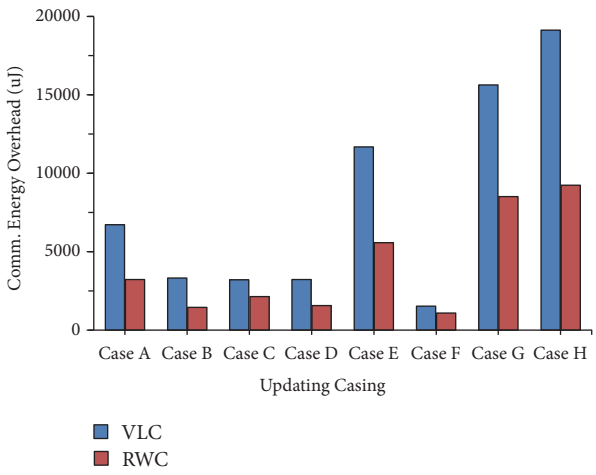


Figure 10: Communication energy overhead (uJ).

Further, we compare VLC with RWC based on energy overhead which is important for battery-power sensor nodes. We conduct a reprogramming procedure on one sensor node and then send block codes from sink node by RWC and smartphone by VLC, respectively. Figure 10 gives the energy overhead of the sensor node using VLC and RWC. The energy overhead is measured by current-monitoring circuit. The energy overhead of VLC mainly comes from turning on TSL2561 during transmission period. Obviously, the power of ambient light sensor is higher than the traditional RF transceiver CC2420. The high-power light sensor leads to the fact that the average energy overhead of VLC increases 49.1% than RWC. However, in most of security-sensitive cases, this energy overhead increasing is acceptable. In addition, given that lost sensor node is or offline from network, the ReVLC may be only viable reprogramming solution to fix sensor node.

5. Conclusion

Most of current reprogramming approaches solve the security problem in upper-level software such as adding complex security mechanism. In this paper, we attempt to reduce the security risk by changing the physical-level communication mode. For this end, we propose a VLC-based reprogramming approach, ReVLC applied in resource-limited sensor node. In this approach, we directly use low-cost and general sensor nodes without hardware modification and a CTOS smartphone as VLC gateway to save and forward codes. Meanwhile, using the ambient light sensor equipped by sensor node as signal receiver leads to unappealing transmission performance. So, in ReVLC, we propose a code block mechanism to minimize transferred data and CDH-PIM with compressing representation to optimize modulation of VLC. The experiment results prove the availability of ReVLC.

In future work, we will continuously study the VLC modulation from the more compressing representation and also introduce the probability coding such as Huffman Coding into modulation.

Data Availability

Figures 7–10 and Tables 2-3 which are used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research work is supported by China Natural Science Foundation (NSF) under Grant no. 61502427 and by the Zhejiang Provincial Natural Science Foundation of China under Grant no. LY16F020034.

References

[1] Y. C. Luo, J. G. Wu, Z. K. Zhang, W. J. Shi, and Y. Q. Miu, "Online algorithm for secure task offloading in dynamic networks," in *Proceedings of IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, China, 2017.

[2] Y. Rao, J. Wang, R. Tian, and F. Zhu, "Dynamic updating based key management algorithm for wireless sensor networks," in *Proceedings of the 2011 International Conference on Wireless Communications and Signal Processing, WCSP 2011*, China, November 2011.

[3] P. E. Lanigan, R. Gandhi, and P. Narasimhan, "Sluice: secure dissemination of code updates in sensor networks," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS '06)*, July 2006.

[4] S. Hyun, P. Ning, A. Liu, and W. Du, "Seluge: Secure and DoS-resistant code dissemination in wireless sensor networks," in *Proceedings of the 2008 International Conference on Information Processing in Sensor Networks, IPSN 2008*, pp. 445–456, April 2008.

[5] K. Park, J. H. Lee, T. Y. Kwon, and J. Song, "Secure dynamic network reprogramming using supplementary hash in wireless sensor networks," in *Proceedings of the Fourth International Conference on Ubiquitous Intelligence and Computing*, pp. 653–662, Hong Kong, 2007.

[6] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in wireless network coding," *ACM Transactions on Information and System Security*, vol. 14, no. 1, 2011.

[7] Y. W. Law, Y. Zhang, J. Jin, M. Palaniswami, and P. Havinga, "Secure rateless deluge: pollution-resistant reprogramming and data dissemination for wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2011, Article ID 685219, 22 pages, 2011.

[8] Q. Dong, D. Liu, and P. Ning, "Pre-authentication filters: Providing DoS resistance for signature-based broadcast authentication in sensor networks," in *Proceedings of the WiSec'08: 1st ACM Conference on Wireless Network Security*, pp. 2–13, USA, April 2008.

[9] H. Tan, D. Ostry, J. Zic, and S. Jha, "A confidential and DoS-resistant multi-hop code dissemination protocol for Wireless Sensor Networks," in *Proceedings of the 2nd ACM Conference on*

Wireless Network Security, WiSec'09, pp. 245–252, Switzerland, March 2009.

[10] D. Kim, D. Kim, and S. An, "Source authentication for code dissemination supporting dynamic packet size in wireless sensor networks," *Sensors*, vol. 16, no. 7, 2016.

[11] M. Xie, U. Bhanja, J. Shao, G. Zhang, and G. Wei, "LDSCD: A loss and DoS resistant secure code dissemination algorithm supporting multiple authorized tenants," *Information Sciences*, vol. 420, pp. 37–48, 2017.

[12] L. Yang, S. Li, Z. Xiong, and M. Qiu, "HHT-based security enhancement approach with low overhead for coding-based reprogramming protocols in wireless sensor networks," *Journal of Signal Processing Systems*, vol. 89, no. 1, pp. 13–25, 2017.

[13] D. Li, W. Liu, and L. Cui, "EasiDesign: An improved ant colony algorithm for sensor deployment in real sensor network system," in *Proceedings of the 53rd IEEE Global Communications Conference, GLOBECOM 2010*, USA, December 2010.

[14] L. Fan, Q. Liu, C. Jiang et al., "Visible light communication using the flash light LED of the smart phone as a light source and its application in the access control system," in *Proceedings of the 2016 IEEE MTT-S International Wireless Symposium, IWS 2016*, China, March 2016.

[15] Y. Wang and N. Chi, "A high-speed bi-directional visible light communication system based on RGB-LED," *China Communications*, vol. 11, no. 3, pp. 40–44, 2014.

[16] T. Adiono, A. Pradana, R. V. W. Putra, and S. Fuada, "Analog filters design in VLC analog front-end receiver for reducing indoor ambient light noise," in *Proceedings of the 2016 IEEE Asia Pacific Conference on Circuits and Systems, APCCAS 2016*, pp. 581–584, Republic of Korea, October 2016.

[17] R. K. Panta, S. Bagchi, and S. P. Midkiff, "Zephyr: efficient incremental reprogramming of sensor nodes using function call indirections and difference computation," in *Proceedings of USENIX Annual Technical Conference*, 2009.

[18] R. K. Panta and S. Bagchi, "Hermes: Fast and energy efficient incremental code updates for wireless sensor networks," in *Proceedings of the 28th Conference on Computer Communications, IEEE INFOCOM '09*, pp. 639–647, Brazil, April 2009.

[19] W. Li, Y. Zhang, J. Yang, and J. Zheng, "UCC: Update-conscious compilation for energy efficiency in wireless sensor networks," in *Proceedings of the PLDI'07: 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 383–393, USA, June 2007.

[20] N. M. Aldibbiat, Z. Ghassemlooy, and R. McLaughlin, "Error performance of dual header pulse interval modulation (DH-PIM) in optical wireless communications," *IEE Proceedings—Optoelectronics*, vol. 148, no. 2, pp. 91–96, 2001.

[21] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, pp. 81–94, ACM, November 2004.

[22] R. K. Panta, I. Khalil, and S. Bagchi, "Stream: Low overhead wireless reprogramming for sensor networks," in *Proceedings of the 26th IEEE International Conference on Computer Communications (IEEE INFOCOM '07)*, pp. 928–936, USA, May 2007.

[23] J. Qiu, S. Li, and B. Cao, "RePage: A novel over-air reprogramming approach based on paging mechanism applied in fog computing," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[24] W. Dong, Y. Liu, C. Chen, L. Gu, and X. Wu, "Elon: Enabling efficient and long-term reprogramming for wireless sensor networks," *ACM Transactions on Embedded Computing Systems*, vol. 13, no. 4, article no. 77, 2014.

[25] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN '05)*, pp. 364–369, April 2005.

[26] MAX9928 introduction, available in: http://www.maximintegrated.com/datasheet/index.mvp/id/5753.

[27] S.-K. Kim, J.-H. Lee, K. Hur, K.-I. Hwang, and D.-S. Eom, "Tiny module-linking for energy-efficient reprogramming in wireless sensor networks," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 1914–1920, 2009.

[28] K. Lorincz, B.-R. Chen, and G. W. Challen, "Mercury: a wearable sensor network platform for high-fidelity motion analysis," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 183–196, ACM, November 2009.