Delft University of Technology
Master's Thesis in Embedded Systems

Robust Downstream Communication and Storage for Computational RFIDs

Jethro Tan

Robust Downstream Communication and Storage for Computational RFIDs

Master's Thesis in Embedded Systems

Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Jethro Tan
j.e.t.tan@student.tudelft.nl

20th August 2015

Author
Jethro Tan (j.e.t.tan@student.tudelft.nl)
Title
Robust Downstream Communication and Storage for Computational RFIDs
MSc presentation
25th August 2015

The content of this thesis was submitted as the article titled *Wisent: Robust Downstream Communication and Storage for Computational RFIDs* by Jethro Tan, Przemysław Pawełczak, Aaron Parks, and Joshua R. Smith to IEEE INFOCOM 2016 conference.

Abstract

Computational RFID (CRFID) devices are emerging platforms that can enable perennial computation and sensing by eliminating the need for batteries. Although much research has been devoted to improving upstream (CRFID to RFID reader) communication rates, the opposite direction has so far been neglected, presumably due to the difficulty in guaranteeing fast and error-free transfer amidst frequent power interruptions at the CRFID. With growing interest in the market where CRFIDs are forever-embedded in various structures, it is necessary for this void to be filled. Therefore, we propose Wisent—a robust downstream communication protocol for CRFIDs that operates on top of the legacy UHF RFID communication protocol: EPC C1G2. The novelty of Wisent is its ability to adaptively change the frame length sent by the reader, based on a length throttling mechanism, to minimize the transfer times at varying channel conditions. We present an implementation of Wisent for the WISP 5 and an off-the-shelf RFID reader. Our experiments show that Wisent allows transfer up to 16 times faster than a baseline, non-adaptive shortest frame case, i.e. single word length, at sub-meter distance. As a case study, we show how Wisent enables wireless CRFID reprogramming, demonstrating the world's first wirelessly reprogrammable (software defined) CRFID.

Preface

In the current era, where a lot of devices take part in the Internet-of-Things, I have always considered the communication between them as a given. Never have I been more wrong when I was given the task to make wireless reprogramming happen on CRFIDs. This thesis is the first step that enabled downstream communication on CRFIDs in the hope that future applications and research will utilize it.

First, I would like to thank my supervisor, Przemysław Pawełczak, for his excellent guidance, support and unfailing optimism throughout this entire project and for keeping me motivated until the very last day. From him I have learned how to conduct research and to keep thinking from a different perspective. Second, I would like to thank the people from the University of Washington in Seattle who collaborated with us on writing an article, and for all their help and patience. In particular I would like to thank Benjamin Ransford, Aaron Parks, and Joshua R. Smith. Furthermore, I want to thank Koen Langendoen for hosting me at the Embedded Software group, Manuel Mazo Jr. for being a member of my committee during a time when most people are still on vacation, Marco Cattani, Soumya Subramanya, Ioannis Protonotarios and Qingzhi Liu for their help and feedback on drafts. I also want thank my colleagues at the 9th floor whom I talked to everyday for necessary distractions and exchanged opinions with. In particular I would like to thank Ivar in 't Veen for sharing his research on the WISP and I wish him good luck on his graduation. Finally, I want to sincerely thank my friends, those who have graduated before me and others who are still busy abroad, and my family for their unconditional support to overcome the difficulties that might or might not have happened throughout this work.

Delft, The Netherlands
20th August 2015

Contents

Preface **v**

1 Introduction **1**
1.1 Problem Statement 2
1.2 Challenges 2
1.3 Contributions 3
1.4 Thesis Organization 3

2 Related Work **5**
2.1 Data Transfer on CRFIDs 5
2.2 Wireless Reprogramming of Communication Platforms 6
2.3 Computation across Power Cycles 6
2.4 Literature Study 6
2.4.1 Introduction to EPC C1G2 7
2.4.2 Upstream Communication using EPC C1G2 7
2.4.3 Modifications and Alternatives to EPC C1G2 8
2.4.4 Overview of Methods to Improve Upstream Communication for CRFIDs 8

3 Downstream CRFID Protocol: Preliminaries **9**
3.1 Host-to-CRFID Communication: System Overview 9
3.2 Reader-to-Tag Communication: EPC C1G2 9
3.3 Host-to-Reader Communication: LLRP 10
3.4 Design Requirements 11

4 Wisent: Design and Implementation **13**
4.1 Wisent Hardware and Software 13
4.2 Wisent Components 14
4.2.1 Message Format 14
4.2.2 Message Definitions 14
4.3 Wisent: Basic Protocol 16
4.4 Wisent EX: Extending Wisent Basic with BlockWrite 18
4.4.1 Implementation of BlockWrite 18
4.4.2 Verification 19

4.4.3 Throttling 19

5 Experiment Results 21

5.1 Evaluation of BlockWrite 21

5.1.1 BlockWrite Performance Metrics 21

5.1.2 BlockWrite Performance Results 23

5.1.3 BlockWrite Performance Metrics Model 23

5.2 Evaluation of Wisent 24

5.2.1 Wisent Performance Metrics 24

5.2.2 Wisent Performance Results 24

5.2.3 Selection of Values for Throttling Parameters 25

5.3 Case Study: CRFID Wireless Reprogramming 27

5.3.1 Experiment Setup and Results 29

5.3.2 Data Transfer Energy Consumption 29

5.3.3 Reprogramming a Tissue-Embedded CRFID 30

6 Conclusion 31

6.1 Future Work 31

Chapter 1

Introduction

Computational RFIDs (CRFIDs) and other wireless, transiently-powered computing devices are emerging platforms that enable sensing, computation and communication without batteries [1]. CRFIDs have been proposed for use in scenarios such as structural health monitoring, in which a device may be embedded in a reinforced concrete structure [2], or medical implants involving tasks such as blood glucose monitoring [3] or pacemaker control [4], where access to the device requires an expensive and/or risky surgery. In these deeply embedded applications, physical access to the device is difficult or impossible, making battery-free operation attractive as the battery maintenance requirement is avoided entirely.

However, as the complexity of use cases for CRFIDs grow, there is another emerging maintenance requirement: the need to patch or replace the firmware of the device, or to alter application parameters including the RFID radio layer controls. In current CRFIDs, maintenance of firmware (due to e.g. errors) requires a physical connection to the CRFID, nulling the main benefit of battery-free operation.

Existing CRFIDs have no means for reliable high-rate downstream (i.e. RFID reader to CRFID) wireless communication and storage. This makes downstream protocols for CRFIDs a potent area of study. Current CRFIDs use UHF RFID standards, such as EPC C1G2 [5], designed to support inventory management applications with minimal computational and data transfer requirements [6, 7]. Any UHF RFID network is built around an interrogator (reader), which provides power to tags in the vicinity, and which can both send/receive data to/from those tags. The uplink from tag to reader is accomplished through backscatter communication, in which the tag modulates its reflection of the reader's carrier signal in order to communicate—with orders of magnitude less power than a conventional radio. The reader is a wall socket-powered device, while the tag is a highly energy-constrained energy-harvesting battery-free platform with frequent power supply breaks, see Fig. 1.1.

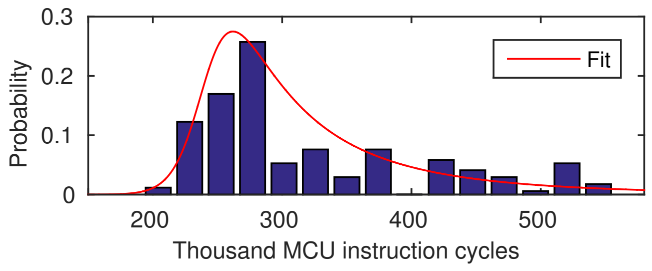


Figure 1.1: Example distribution of CRFID (WISP 5 [8]) available harvested energy between periods of state loss. Energy is mapped into the number of instruction cycles a WISP 5 can achieve per burst with its default hardware configuration. The frequent state loss makes reliable bulk data transfer hard.

1.1 Problem Statement

In CRFIDs, downstream transfer functionality would enable (i) the processing of data sent wirelessly, (ii) CRFID to CRFID data transmission, and most importantly (iii) wireless reconfiguration, including reconfiguration of the communication protocol itself (e.g. demodulation, decoding). In this work, we therefore pose the following research question: *How to enable robust and fast downstream communication for CRFIDs?*

1.2 Challenges

There are two challenges in enabling CRFID downstream.

Challenge 1: Existing CRFIDs rely on the suboptimal downstream communication capability of the EPC C1G2 standard. In fact, EPC C1G2 does not have built-in support for *fast* transfers of large portions of data from the reader to the tag. Building a replacement protocol for reader-to-CRFID communication would indeed improve communication capabilities and performance, however, doing so would reduce compatibility with existing EPC C1G2-compliant RFID systems, which represent a significant amount of existing infrastructure. Therefore, designing a protocol extension *on top of* EPC C1G2 is the natural path (although a difficult one) towards a capable, yet practical downstream CRFID protocol.

Challenge 2: Transient power availability in CRFIDs means that tags will often lose power, and in turn processor state [9] (again see Fig. 1.1). Previous CRFID platforms attempted to alleviate this limitation by the use of non-volatile flash memory. However, not only do rewrite or erase operations on flash memory require the erasure of an entire memory block, but the erasure of flash cells are also slow and energy intensive relative to volatile memory writes [10]. Hence, flash memory on CRFIDs is unsuitable for repetitive tasks involving write operations such as data storage or reprogramming applications.

1.3 Contributions

While the current focus of CRFID systems lies in improving *upstream* communication, see e.g. [11–13], due to the mentioned challenges, to the best of our knowledge, there has surprisingly been no work focusing on *downstream* communication, where CRFIDs are on the receiving end of large data transfers. To fill this void our contributions are:

Contribution 1: We leverage EPC C1G2 functionality to implement and evaluate multi-word downstream data transfer, i.e. longer than the de facto limit of a 16-bit word (2 bytes). Experimentally we show a threefold improvement of raw downstream throughput. We also show that the error rate of multi-word messages transferred by the reader to the CRFID reduces throughput instantly to zero at long reader-to-CRFID distances.

Contribution 2: We design, implement and experimentally evaluate Wisent[1]—a downstream-oriented protocol for CRFIDs. Wisent allows for an adaptation of the length of the reader message based on the received message rate at the CRFID to keep the transfer speed high, while minimizing the data transfer errors. Furthermore, utilizing FRAM's capability of random-access instantaneous read/write operations without memory block erasure available in the latest CRFID release, namely WISP 5 [8], we enhance Wisent with a mechanism for fast storage and verification of large portions of data.

Contribution 3: We implement and demonstrate the world's first fully wirelessly-reprogrammable CRFID, enabling truly reprogrammable radio stack software-defined CRFID.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Related work is described in Chapter 2, while Chapter 3 introduces the CRFID downstream protocol preliminaries. Chapter 4 outlines the design of Wisent and its implementation. Experimental results and evaluation of CRFID downstream transfer improvements through the EPC C1G2 `BlockWrite` implementation, Wisent itself, and the use of Wisent in wireless reprogramming scenarios are presented in Chapter 5. The thesis is concluded along with discussion of future work in Chapter 6.

Finally, we remark that in the spirit of the research reproducibility, source code for Wisent, measurement data and log files are available upon request or downloadable via http://www.es.ewi.tudelft.nl/papers/2016-Tan-INFOCOM_source_code.zip.

[1]Wisent, i.e. the European Bison, is a wordplay on '**Wi**relessly **sent**'.

Chapter 2

Related Work

As introduced in Chapter 1, CRFIDs are platforms for batteryless sensing, computation and communication. Such a platform was first introduced by Intel in 2006 [?] in the form of the Wireless Identification and Sensing Platform (WISP). The original WISP featured a 3D accelerometer, temperature sensor, LED and 2 KB MCU flash memory that limited its firmware functionalities. Derivations, upgraded or modified versions were released afterwards such as the Moo [7] and the NFC-WISP [14]. CRFIDs soon became the target of many researchers due to the many challenges that were present. Some works that address the challenges related to the work in this thesis are further discussed in the following sections.

2.1 Data Transfer on CRFIDs

As we noted earlier, no existing work, to the best of our knowledge, targets fast and reliable reader-to-CRFID communication. On the other hand various CRFID-related works discuss data schemes for upstream (tag-to-reader) data transfers. Examples of such work include HARMONY [12], a data transfer scheme implemented on the WISP 4.1 with the purpose of reading bulk data, and Flit [13], a coordinated bulk transfer protocol implemented on the Intel WISP. A recent work [11] considers the image data transfer from the camera-enabled WISP to an RFID host. For the interested reader we refer to [15, 16], where system-level evaluation of link layer performance of EPC C1G2 has been investigated. We also refer to BLINK [17]: the only known alternative to EPC C1G2 in the context of CRFID, and to [18], where a modified EPC C1G2 was used to speed up tag access operations.

2.2 Wireless Reprogramming of Communication Platforms

The concept of wireless reprogramming and over-the-air (OTA) firmware update mechanisms is well researched in non-CRFID fields, namely in cellular systems and wireless sensor networks (WSNs). For example, a software redistribution architecture for mobile cellular terminals was discussed first in [19], while code distribution architecture for WSNs has been discussed in [20]. However, wireless reprogramming is completely new to the CRFID field and the only two relevant works we are aware of are Bootie [10] and FirmSwitch [21]. Bootie describes a proof-of-concept bootloader for CRFIDs and a preliminary design of a firmware update protocol that allows Bootie to accept and install firmware updates wirelessly. However, its wireless protocol proposal has not been implemented or tested and no results on its performance exist thus far. FirmSwitch introduced and implemented a wireless firmware switching mechanism for CRFIDs. Unfortunately, FirmSwitch does not deliver the most significant benefit of the reprogramming, as only pre-installed programs can be selected by its CRFID bootloader. Neither OTA firmware updates nor fast/reliable transmission protocols were proposed therein.

2.3 Computation across Power Cycles

Task scheduling mechanisms to enable computation across power cycles are described in [9] and [22]. Mementos [9] is an energy-aware state checkpointing system implemented on the WISP 4.1 DL, while QuickRecall [22] introduces the notion of unified memory by utilizing FRAM non-volatile memory. In unified memory, the program state can be retained by directly mapping the memory to the non-volatile memory instead of the SRAM, resulting in reduced energy consumption and less overhead in time incurred to retain RAM data across power cycles. Instead of task scheduling mechanisms, requiring a program to be idempotent would also enable computation power cycles at the cost of hindering long-running applications by limiting their use of non-volatile storage [**?**].

2.4 Literature Study

To get a more detailed overview in the field of communication of CRFIDs and place this work in context, several works that were mentioned in Section 2.1 will be described more thoroughly. Before we proceed to such detailed descriptions, we briefly explain the EPC C1G2 protocol itself.

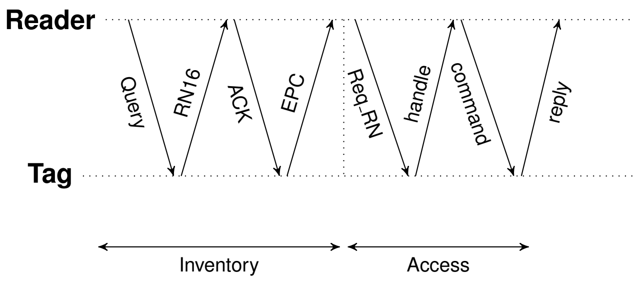


Figure 2.1: Example of tag inventory and access in EPC C1G2 [5, Annex E].

2.4.1 Introduction to EPC C1G2

In EPC C1G2, a communication sequence between a reader and a tag is called an inventory round. Such a sequence can only be initiated by the reader and consists of a series of issued commands starting with the `Query` command, which contains link parameters for the communication [5, Table 6.32]. An example of such an inventory round is depicted in Fig. 2.1. The tag replies to the `Query` by backscattering a 16 bit random number called `RN16`. The reader then acknowledges this `RN16` by repeating it within an `ACK`. If the tag received a valid `RN16` back from the reader, the tag finally responds by backscattering its `EPC` field—a 12 bytes field of data. While the reception of the `EPC` field marks the end of an inventory round, the same round can be extended if the reader wants further *access* (e.g. `Write`, `Read`, `BlockWrite`) to the tag. In this case, the reader issues a `Req_RN` still containing the same `RN16`, which is replied to by the reader using a `handle`. The reader then finally accesses the tag by issuing the corresponding access command (e.g. `Read`, `Write`, `BlockWrite`) with the received `handle`, marking the end of the inventory round.

2.4.2 Upstream Communication using EPC C1G2

There are two ways to achieve upstream communication on EPC C1G2. It can either be achieved by the reader issuing a `Read` command, or through constant modification of the `EPC` field [13]. By constantly modifying the `EPC` field, the reader does not need to issue a command to access the tag, thus reducing the length of each communication sequence and therefore the time to send the data. The authors of Flit [13] also proved that while using the `Read` command seems ideal at first glance and attractive due to the support of variable response lengths, it is in fact inefficient in terms of channel utilization and energy consumption. Additionally, the size of the read messages is limited by factors such as bit error rate and timing drift, which are also present in this work.

The authors of HARMONY [12] improved the throughput of reading

Table 2.1: Methods on Improving Upstream Communication

Name	EPC C1G2 compatibility	Complexity	Speedup[1]
Flit	Compatible	Simple	2x
HARMONY	Compatible	Medium	3x
Persistent Handle	Extension	Simple	5x
BLINK	Not Compatible	Complex	3x

[1] Speedup by throughput gained compared to default EPC C1G2 using a `Read` command.

bulk data from CRFIDs further by utilizing a data transfer scheme. After examinations, they conclude that the root cause of the large latency was caused by the CRC computation overhead that linearly increases with the data size. Therefore, CRC precomputing has been proposed in their data scheme and by exploiting intermediate computations multiple HARMONY data units can be concatenated to increase data transfer efficiency and flexibility.

2.4.3 Modifications and Alternatives to EPC C1G2

Despite a disadvantage of compatibility issues with commodity readers and existing RFID systems, an extension to EPC C1G2 that allows *persistent handles* was proposed and designed in [18]. The design of a persistent handle simply requires the `Handle` to be maintained along other state variables in the CRFID. With such a persistent handle, tags would remember their last transmitted `Handle` for a period of time so they can backscatter data indicated by that handle immediately upon receiving any command that accesses the tag.

A complete different alternative to EPC C1G2 called BLINK has been proposed in [17], which includes a rate adaption algorithm to select the optimal bitrate depending on the channel conditions.

2.4.4 Overview of Methods to Improve Upstream Communication for CRFIDs

In Table 2.1 all methods to improve upstream communication on CRFIDs are summarized. We conclude that while the idea of extending EPC C1G2 with a persistent handle leads to compatibility issues on already existing systems, the performance gained in terms of speedup is well worth it in comparison to BLINK that aims to replace EPC C1G2 completely. We also note that Flit and HARMONY could possibly be combined together and further extended with the Persistent Handle to improve performance beyond what has been achieved by their respecive authors.

Chapter 3

Downstream CRFID Protocol: Preliminaries

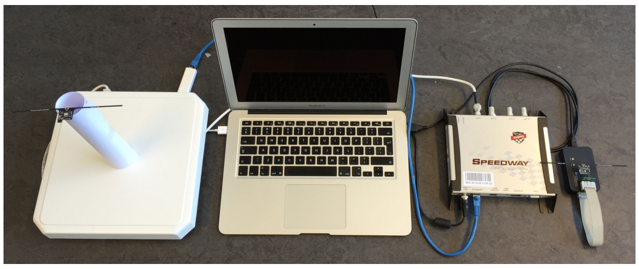
In this chapter, we give an overview of the CRFID system and limitations in it that have to be taken into account in any CRFID protocol design. From these limitations, we also formulate the design requirements for our downstream CRFID protocol.

3.1 Host-to-CRFID Communication: System Overview

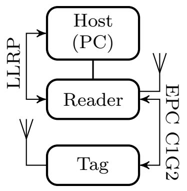
Three fundamental components of a typical CRFID system are (i) a host machine—any device that is able to communicate with the RFID reader over any popular physical interface such as Ethernet or Wi-Fi, (ii) an RFID reader (connected to an antenna) and (iii) a CRFID tag, see Fig. 3.1. All communications from a host to a tag are first sent to the reader. In the context of the EPC C1G2 [5] standard, the Low Level Reader Protocol (LLRP) [23] specifies a network interface for such communication. The reader then issues EPC C1G2 commands corresponding to the LLRP messages it received from the host. Naturally, EPC C1G2 and LLRP communication primitives enforce limitations that have to be taken into account in any CRFID protocol design. We discuss them below.

3.2 Reader-to-Tag Communication: EPC C1G2

There are only two commands in the EPC C1G2 standard with the purpose of sending data from the reader to a tag: `Write` and `BlockWrite` [5]. While `Write` is a mandatory command in the EPC C1G2 specification (i.e. tags conforming to the standard must support these commands [5, pp. 13]), `BlockWrite` is specified as optional, and prior to this work its implementation



(a) CRFID hardware



(b) Protocol layers

Figure 3.1: CRFID system setup: (a) hardware (from left to right) (i) paper tube-protruded WISP 5 on an antenna, (ii) host (PC), (iii) reader—connected to an antenna and the host, (iv) Texas Instruments MSP430 Flash Emulation Tool connected to a WISP and a host; (b) Protocol layers of CRFID communication.

was nonexistent in CRFIDs. Data transfer with `Write` is limited to only one word at a time. To achieve data transmission rates beyond what is supported with `Write`, an implementation of `BlockWrite` for CRFIDs is fundamentally needed. Length-wise, `BlockWrite` can be orders of magnitude longer than `Write`. It is thus far more susceptible to channel errors (induced by e.g. CRFID movement). Therefore, its performance must be well understood to maximize the transfer speeds for CRFID downstream protocols built on top of EPC C1G2.

3.3 Host-to-Reader Communication: LLRP

Despite LLRP's large overhead, it is possible to use it as part of a downstream protocol.

Proposition 1 *Continuous host-to-reader data streaming is enabled in LLRP by issuing a train of* `AccessSpec` *commands.*

Evidence *A host machine can command a reader to start an inventory session on tags only by enabling a* `ROSpec` *[23, Sec. 6]. An* `AccessSpec` *can optionally be included inside a* `ROSpec` *to make a reader issue an access command (e.g.* `Write`*) on tags. To change the current command or its properties (e.g. the number of words or content of a* `BlockWrite`*) issued by the reader, the* `AccessSpec` *must be changed. To achieve this, the host machine needs to tell the reader to first delete the current* `AccessSpec`*, add a new one, and finally enable it (see Fig. 3.2). Alternatively, an* `AccessSpecStopTrigger` *can be specified by the host to make a reader autonomously delete an* `AccessSpec` *once its command has been performed* `OperationCountValue` *times by the tag. Using both principles, it is possible for a host to send multiple* `AccessSpecs` *in a continuous fashion to a reader to effectively enable a data stream.* □

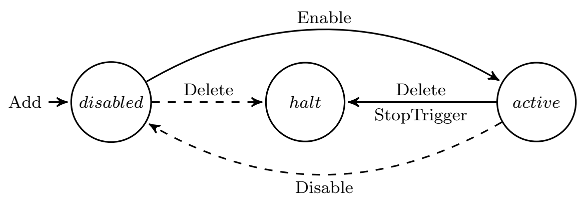


Figure 3.2: LLRP AccessSpec reader state machine [23, Fig. 10] utilized in enabling continuous host-to-reader bulk data transfer following Proposition 1. The dashed lines represent optional state transitions.

3.4 Design Requirements

From the challenges mentioned in Chapter 1 and the limitations found in CRFID systems, we can now formulate the design requirements for a downstream CRFID protocol. It must *(i)* be built on top of EPC C1G2 and LLRP; *(ii)* be able to send large portions of data from a host to a CRFID via a reader; data transmission and storage must be reliable; *(iii)* tolerate interruptions to operating power; *(iv)* tolerate changes of distance of the CRFID platform to the reader's antenna. Additionally, to achieve transmission rates beyond the limitations set by EPC C1G2 `Write`, *(v)* the `BlockWrite` command should be enabled. Furthermore, to alleviate the negative effect of transient power on repetitive tasks in write operations to CRFID's memory, such protocol must use *(vi)* CRFID platforms with non-volatile FRAM memory. We remark that although a CRFID system might consist of multiple tags, we design Wisent for communication with a single tag in mind[1].

[1]Extensions of Wisent for communication with multiple tags will be discussed in Section 6.1.

Chapter 4

Wisent: Design and Implementation

In this chapter, we discuss in detail the design that were made and implementation of two different versions of Wisent — a basic version (Wisent Basic) without `BlockWrite` usage and an extended version (Wisent EX) that takes advantage of the `BlockWrite` command.

4.1 Wisent Hardware and Software

As a host we use an x86-64 computer with an Intel i5-3317U processor running Ubuntu 14.04. As an RFID reader we use an off-the-shelf 915 MHz Impinj Speedway[1] R420 with firmware version 4.8.3 connected to a Laird S9028PCR 8.5 dBic gain antenna. As a CRFID device we use the WISP 5 [8] with TI MSP430FR5969 MCU, which has 64 KB of FRAM[2]. To initially program the WISP with Wisent we use an MSP430 Flash Emulation Tool (FET), in combination with TI Code Composer Studio (CCS), attached to the host. The FET also enables measuring the energy consumption of the WISP and inspecting the non-volatile memory of the WISP during Wisent implementation.

To implement all necessary features of Wisent in LLRP we have extended sllurp [24]: an LLRP control library written in Python. The CRFID-side of Wisent is implemented in C and MSP430 assembly. The complete Wisent implementation amasses to 350 lines of Python and 60 lines of C in addition to 100 instructions of MSP430 assembly. Refer again to Fig. 3.1 for our CRFID system setup used in Wisent experiments.

[1]We have also successfully tested Wisent using the Impinj Speedway R1000, but the experimental results in this work were generated with the R420 reader only.

[2]Although Wisent has been tested exclusively on the WISP 5, it can be targeted towards any CRFID platform using FRAM non-volatile memory.

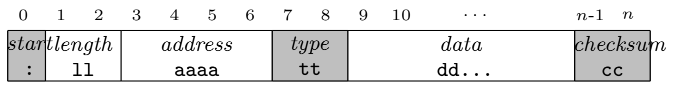


Figure 4.1: Example record of an Intel Hex file. The numbers on top of each field denote the index of the ASCII symbol in the record and the values inside each field denote example content. The length field specifies the byte count in the data field, while the address field represents the destination address for the data in the CRFID memory. Gray fields are not used in Wisent messages.

4.2 Wisent Components

Before proceeding to the protocol definition of Wisent, we first define a message format and various components present in the protocol.

4.2.1 Message Format

To implement fast reader-to-tag transmission, any bulk data that is to be sent to CRFID (e.g. regular file or firmware) should be presented in one specific format. For that purpose we utilize the Intel Hex file format due to its popularity, noting that other formats such as TekHex or Motorola-S [25, Sec. 12.12] can be also supported. An Intel Hex file is divided into lines, called records, and contain fields of information about the data, see Fig. 4.1, which are later parsed for transfer as described below.

4.2.2 Message Definitions

We define m: a *Wisent message*, i.e. message sent from the host to CRFID as content for a `Write` or `BlockWrite` command that is constructed by the host as follows. First, information is taken from the length, address and data fields of each Intel Hex file record i, being the only necessary fields for our implementation. We denote the data that is taken this way as a matrix I, from which $I(i, j)$ is the j-th chunk in row i of size S_p words. The value of S_p depends on which of the two commands, leveraged by the two distinct versions of Wisent described in subsequent Section 4.3 and Section 4.4, the content is constructed for. To a constructed $I(i, j)$, we add a header containing information to identify and instruct the CRFID as to how the remaining part of the message, i.e. the payload, should be handled. Details on the header and payload content, including payload handling, shall also be discussed in Section 4.3 and Section 4.4, as they are also Wisent-version dependent.

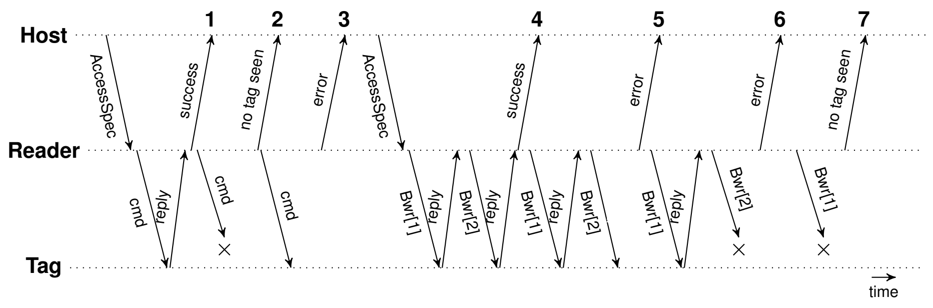


Figure 4.2: Difference of using EPC C1G2 commands as specified by the standard [5] (**1**–**3**) and `BlockWrite` with `WordCount=2` as observed at Impinj readers (**4**–**7**). The reader reports a no tag seen (**2**,**7**) if the tag did not receive the command at all. An error is reported instead if the tag received the command but processed it incorrectly (**3**,**5**,**6**). Only success reports (**1**,**4**) are counted towards the `OperationCountValue` of individual `AccessSpecs`.

The next message to be sent is defined as

$$m_{\text{next}} = \begin{cases} I(1,1) & \text{if } m = \varepsilon, \\ I(i, j+1) & \text{if } j \neq \text{EOL}, \\ I(i+1, 1) & \text{if } j = \text{EOL}, \\ \emptyset & \text{if } i = \text{EOF}, \end{cases} \tag{4.1}$$

where ε represents the undefined message and EOL and EOF denote the end of the record and end of file, respectively.

The backscattered `EPC` field of the CRFID is utilized by Wisent for message verification purposes. During the construction of each message, the host generates verification data that is compared with the `EPC` in a tag report from the reader. However, since the `EPC` is backscattered before a command that accesses the tag is executed [5, Annex E] (e.g. `Write`, `BlockWrite`), an `EPC` after that command is needed to verify the current message. The host receives a *positive acknowledgment* (ACK) of a message if this `EPC` matches the verification data and a *negative acknowledgement* (NACK) otherwise.

The `OperationCountValue` (`OCV`), described in Section 3.3, acts as an upper bound of the *operation frame* in which a message m is actively transmitted by the reader through a command operation. Unlike ACKs and NACKs, which depend on following `EPC` fields, the result of a command operation is reported before the next `EPC` arrives, and only successful commands count towards the `OCV`, see Fig. 4.2. However, such a report contains an `EPC` regardless of the result and therefore, the report of a successful operation can be a NACK and contrarily a report of an erroneous operation can be an ACK.

The *message frame* of a message m starts as soon as the host sends that message to the reader. When m is acknowledged by the CRFID, the

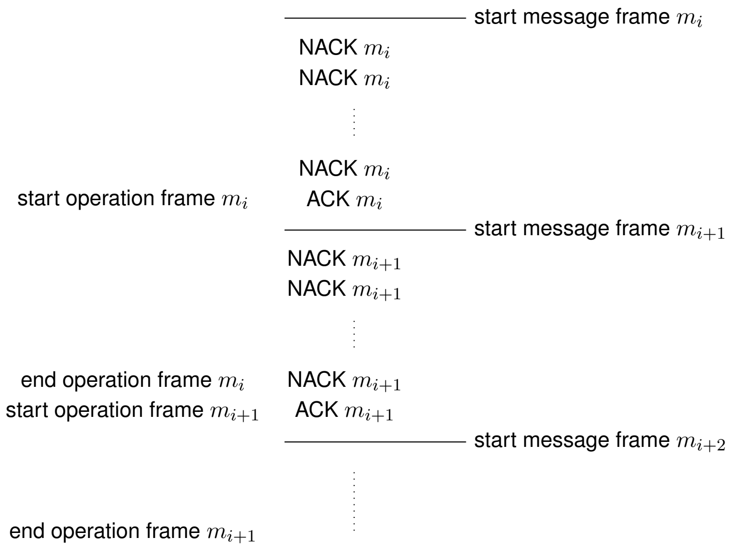

start message frame m_i
NACK m_i
NACK m_i
NACK m_i
start operation frame m_i
ACK m_i
start message frame m_{i+1}
NACK m_{i+1}
NACK m_{i+1}
end operation frame m_i
NACK m_{i+1}
start operation frame m_{i+1}
ACK m_{i+1}
start message frame m_{i+2}
end operation frame m_{i+1}

Figure 4.3: Example message sequence with ACKs and NACKs in Wisent as observed by the host. A large portion of the operation frame of message m_i overlaps with the message frame of message m_{i+1} causing lot of ACKs for message m_i to be NACKs for message m_{i+1}.

message frame of m ends and the host moves on to the message frame of m_{next}, commanding the reader to delete the `AccessSpec` containing m before sending m_{next}. However, there is a delay before this deletion is processed by the reader and thus a delay before the start of the operation frame of m_{next}. Because of this, the operation frame of m overlaps with the message frame of m_{next} causing ACKs for m in the overlap to be NACKs for m_{next}, see Fig. 4.3.

If too many NACKs, defined by $N_{\text{threshold}}$, are observed, a *timeout* is generated and the message will be resent up to a maximum, R_{max}, number of times. If after the maximum amount of resends, the message is still not received, the Wisent transfer is aborted and ends up in a failure. Even though power failures as a result of transient power (see Fig. 1.1) are accounted for by NACKs, separation of the CRFID from the antenna for a prolonged period of time (which results in a transfer failure) are not. This issue will be discussed in Section 6.1.

4.3 Wisent: Basic Protocol

The `Write` command limits a message to only one word of content. Due to the small size of the message, we propose a copy of the message to be kept by the host as verification data to compare with the `EPC`. By fitting a single byte header in such a message, only one byte remains for the payload, i.e.

Protocol 1 Wisent Basic

```
1: $R_{\text{count}} \leftarrow 0$                          ▷ **Host events**
2: $m \leftarrow m_{\text{next}}$                           ▷ See eq. (4.1)
3: SEND($m$)
4: while $R_{\text{count}} < R_{\max}$
5:     upon ACK:
6:         $R_{\text{count}} \leftarrow 0$
7:         $m \leftarrow m_{\text{next}}$                   ▷ See eq. (4.1)
8:         SEND($m$)
9:     upon TIMEOUT:
10:        $R_{\text{count}} \leftarrow R_{\text{count}} + 1$
11:        SEND($m$)
12: upon RECEIVE($m$):                                      ▷ **Tag events**
13:     EPC ← HANDLE($m$)
14:     BACKSCATTER(EPC)
```

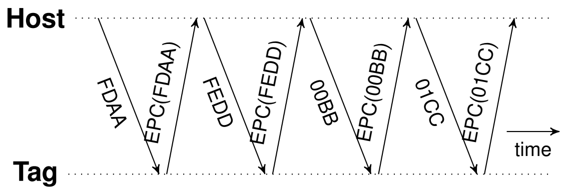


Figure 4.4: Example of a Wisent Basic communication sequence of messages constructed from a record with content `:02AADD00BBCCF0`.

$S_{\text{p}} = 0.5$ words. We propose a non-ambiguous header that describes the payload with data $I(i, j)$ obtained from only the address and data fields. For each row i, two messages should be sent, each with a payload containing half of the address field and a unique header to identify which half of the address field the message contains. The following messages should include a byte from the data field as payload and a header that represents an offset of the byte to the base address. The proposed header definitions are: *(i)* `FD`: new line (address low byte), *(ii)* `FE`: address high byte, and *(iii)* `00`–`20` data byte with offset. For ease of explanation, an example message transfer sequence is shown in Fig. 4.4.

Messages received by the CRFID are handled by a HANDLE(m) function, see Protocol 1 (line 13), as follows. First, the integrity of a message is checked and the CRC16 checksum over the data of the `Write` command is calculated [5, Sec. 6]. However, data written to non-volatile memory might still be corrupted in case of a power failure. Therefore, the address to which the byte was written is immediately read to verify the content. Afterwards, the `EPC` of the CRFID is set to the header of the received message along with

Protocol 2 Wisent EX

```
 1: $R_{\text{count}} \leftarrow 0$                              ▷ **Host events**
 2: $M_{\text{count}} \leftarrow 0$
 3: $S_{\text{p}} \leftarrow S_{\text{max}}$
 4: $m \leftarrow m_{\text{next}}$                               ▷ See eq. (4.1)
 5: SEND($m$)
 6: while $R_{\text{count}} < R_{\text{max}}$
 7:     upon ACK:
 8:         $R_{\text{count}} \leftarrow 0$
 9:         if $M_{\text{count}} > M_{\text{threshold}}$ then
10:             THROTTLE($S_{\text{p}}$, up)                     ▷ See eq. (4.3)
11:             $M_{\text{count}} \leftarrow 0$
12:         else
13:             $M_{\text{count}} \leftarrow M_{\text{count}} + 1$
14:         $m \leftarrow m_{\text{next}}$                       ▷ See eq. (4.1)
15:         SEND($m$)
16:     upon TIMEOUT:
17:         $R_{\text{count}} \leftarrow R_{\text{count}} + 1$
18:         $M_{\text{count}} \leftarrow 0$
19:         THROTTLE($S_{\text{p}}$, down)                       ▷ See eq. (4.3)
20:         $m \leftarrow I(i, j)$                               ▷ See eq. (4.1)
21:         SEND($m$)
22: upon RECEIVE($m$):                                         ▷ **Tag events**
23:     EPC ← HANDLE($m$)
24:     BACKSCATTER(EPC)
```

the read byte. Pseudocode of Wisent Basic can be found in Protocol 1.

4.4 Wisent EX: Extending Wisent Basic with Block-Write

To overcome the single word limit imposed by the `Write` command, we have extended Wisent to make use of `BlockWrite`, denoted as Wisent EX, which introduces the use of a `WordCount` parameter. `WordCount` specifies the size of the `BlockWrite` payload.

4.4.1 Implementation of BlockWrite

We have observed that the Impinj RFID readers do not issue the `BlockWrite` command as specified by EPC C1G2 [5, pp. 92]. Rather than issuing one `BlockWrite` command with data containing all the words, the reader issues a *series* of commands, see Fig. 4.2, each containing one word and a sequential

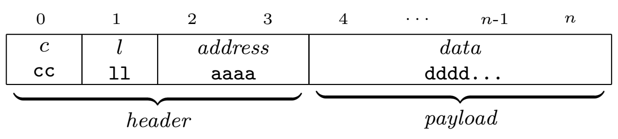


Figure 4.5: Proposed Wisent EX message format. The numbers on top of each field denote the byte index.

increasing address pointer to store the word in the memory of the tag[3]. Because of this, the original `BlockWrite` from the host to the reader is not known to the CRFID, while the reader still considers the series as the instructed `BlockWrite`. A command operation of `BlockWrite` will only be reported as successful if and only if the CRFID processes each of the individual commands in the series and replies to the reader for each of these commands. To improve performance on the CRFID side, the CRC16 checksum of each command in the series is not calculated.

4.4.2 Verification

Because the CRC16 checksum was not calculated for `BlockWrite`, the communication channel is not reliable anymore. An alternative verification mechanism and checksum is necessary to secure the robustness of the communication channel and verify the data written to the non-volatile memory. We propose the use of a single byte checksum for each message that is calculated by taking the least significant byte of the sum[4] of all bytes in message m. This checksum is calculated to check the integrity of the message when received by the CRFID and after writing data to the non-volatile memory.

With messages sizes that can exceed the `EPC` length, using a copy of the message as a means for verification is no longer a viable solution. Nevertheless, a larger message size allows a header of more than one byte, that can be used instead for verification purposes and should be set as `EPC` by the CRFID. In such a header, we propose to include (i) the message checksum c; (ii) the length of the payload in bytes l; and (iii) the destination address of the payload. Consequently, the payload should contain data $I(i, j)$ obtained only from the data field of an Intel Hex file record, see Fig. 4.5.

4.4.3 Throttling

We first introduce the following observation.

[3]Although Wisent EX is built utilizing this non-standard `BlockWrite` behavior, it would also work in combination with RFID readers that *do* issue `BlockWrite` as specified in the EPC C1G2 standard.

[4]Our approach follows the same method used in Intel Hex files, but we are naturally aware of alternative error correction methods. Nevertheless, we will introduce another bootloader specific layer in Section 5.3.

Lemma 1 *A larger* `WordCount` *does not always correspond to faster bulk transfer rates.*

The intuitive reasoning is an existance of a tradeoff between performance in terms of message size and error rate in the communication channel.

Proof 1 *We prove this observation by contradiction. For a* `BlockWrite` *command of length* L *bits the transmission overhead is [5, Table 6.43]* $H = 51$ *bits. Assuming a general complementary error function relating bit error to distance* d *[26, Eq. (3)]* $p_e(d) = \mathrm{erfc}(1/d)$ *the normalized* `BlockWrite` *throughput is* $T(L, d) = L/(L + H)(1 - p_e(d)^{L+H}$ *(compare with [26, Eq. (1)]). Now, for* `BlockWrite` *commands with data lengths* $L_1 = 16$ *bytes and* $L_2 = 32$ *bytes we have* $T(L_1, d) < T(L_2, d)$ *for* $d = 0.2$*, while for* $d = 0.5$ *the opposite holds, which completes the proof.*

Following the above observation we propose the use of a throttling mechanism to adjust S_{p}, the payload size of messages, which is initialized to a user defined value S_{max}. We are aware of the additive increase, exponential decrease throttling mechanism present in other protocols. However, because a Wisent message cannot contain data from two different data rows $I(i)$ and $I(i + 1)$, such a throttling mechanism would not benefit Wisent as the amount of data in each data row is limited. Therefore, we propose a throttling mechanism as described below. Given the number of words S_{r} in data row $I(i)$, we construct a set T_t for S_{p} values as

$$T_t = \left\{ S_{\mathrm{p}} : S_{\mathrm{p}} = \left\lceil \frac{S_{\mathrm{r}}}{n} \right\rceil \right\}, \quad n, S_{\mathrm{r}} \in \mathbb{N}^+, n \leq S_{\mathrm{r}}, \tag{4.2}$$

where n represents the number of data chunks, and in turn messages, in which that row should be split. The throttling function is then defined as

$$\textsc{throttle}(S_{\mathrm{p}}, \rho) = S_{\mathrm{p+1}} \in T_t, \tag{4.3}$$

where $\rho = \{\mathrm{up}, \mathrm{down}\}$, with $S_{\mathrm{p+1}} > S_{\mathrm{p}}$ if throttling up, and $S_{\mathrm{p+1}} < S_{\mathrm{p}}$ if throttling down.

If the need for a resend is perceived, S_{p} is throttled down to decrease the load on the communication channel. On the other hand, S_{p} is increased upon $M_{\mathrm{threshold}}$, a threshold of consecutive successful messages. With this, the full description of the protocol is complete, refer to pseudocode in Protocol 2.

Chapter 5

Experiment Results

In this chapter, we show results of experiments of the `BlockWrite` command in isolation from Wisent, data transfer using Wisent and wireless reprogramming usig Wisent. The experiment setup is as shown in Fig. 3.1 and explained in Section 4.1. The whole measurement setup was located inside of a university laboratory for the `BlockWrite` and wireless reprogramming experiments while the data transfer experiments took place in a university office. As a means for verifiying reproducibility of the results, each experiment has been conducted five times.

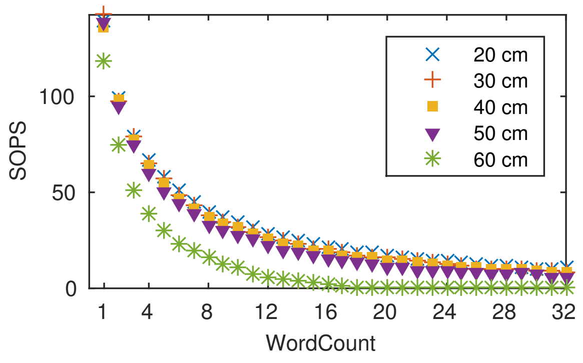
5.1 Evaluation of BlockWrite

EPC C1G2 implies a maximum value of 255 words for `WordCount` by the 8-bit length of the `WordCount` header field. We have tested multiple `WordCount` values and found that an R420 reader is not able to issue more than 32 words at a time.

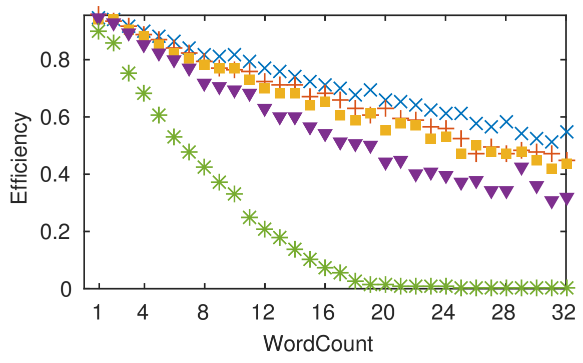
5.1.1 BlockWrite Performance Metrics

During our experiments we let the reader issue a single `BlockWrite` with a set `WordCount` for each experiment. Such operation is repeated for a duration of ten seconds, i.e. the host sends an `AccessSpec` with a `BlockWrite` to the reader without an `AccessSpecStopTrigger` and halts after ten seconds, over multiple distances $d = \{20, 30, \dots, 60\}$ cm from the tag to the antenna.

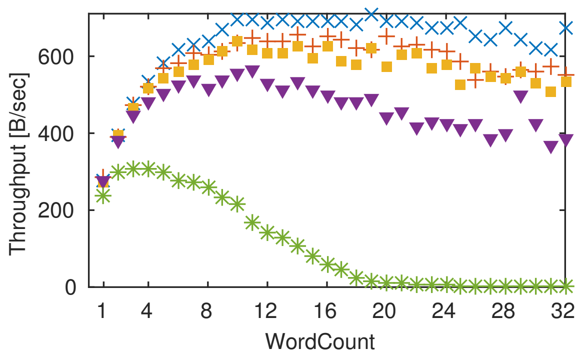
We propose the following set of performance criteria of the `BlockWrite`: (i) number of *success operations per second* (SOPS) defined as $\psi_\mathrm{s} = n_\mathrm{s}/t$ where n_s is the number of success reports and t (sec) is the duration of the experiment; (ii) the *number of total operations per second* (TOPS) defined as $\psi_\mathrm{t} = n_\mathrm{t}/t$, (iii) *efficiency* defined as $\eta = \psi_\mathrm{s}/\psi_\mathrm{t}$, and (iv) *throughput* defined as $\theta = 2x\psi_\mathrm{s}$ (B/sec), where x is the `WordCount` value (since each word is 2 bytes long).



(a) SOPS (ψ_s)



(b) Efficiency (η)



(c) Throughput (θ)

Figure 5.1: Success operations per second, efficiency and throughput of a single `BlockWrite` as functions of `WordCount` over various distances.

Table 5.1: Function parameters for $f_{\psi_t}(x)$ and $g_\eta(x)$ and R^2 for $h_\eta(x)$

d (cm)	a_η	b_η	a_2	b_2	c_2	R^2
20	0.0138	0.9448	170.3735	0.4184	–22.1623	0.9176
30	0.0163	0.9401	166.3176	0.4523	–16.6735	0.8627
40	0.0168	0.9270	164.6218	0.4341	–18.7205	0.7716
50	0.0204	0.9056	158.3378	0.4909	–10.9255	0.4504
60	0.0503	0.8710	122.2697	0.7347	11.0553	0.8553

5.1.2 BlockWrite Performance Results

In Fig. 5.1(a) we show that we are able to improve the throughput of reader-to-CRFID communication manyfold compared to a single word length. While there is only minor efficiency degradation between 20 and 50 cm (Fig. 5.1(b)), the effect of issuing `BlockWrite` with large `WordCount` can clearly be seen if the WISP is moved too far away from the antenna. The efficiency is impacted much more at 60 cm, possibly preventing further usage of such `WordCount` values in Wisent EX messages.

5.1.3 BlockWrite Performance Metrics Model

From the measured data presented in Fig. 5.1 we conjecture that simple functions can be used to describe all experimentally obtained statistics, which can be used in future analytical studies. For this purpose we propose a model for which we introduce $f_{\psi_t}(x)$ and $g_\eta(x)$, which describe the relations between selected word size x and ψ_t and η, respectively. From these functions, $f_{\psi_s}(x)$ and $h_\theta(x)$ follow, describing the respective relations for ψ_s and θ. We then have

$$f_{\psi_t}(x) = \frac{a_2}{x^{b_2}} + c_2, \tag{5.1}$$

$$g_\eta(x) = \frac{f_{\psi_s}(x)}{f_{\psi_t}(x)} = -a_\eta x + b_\eta, \tag{5.2}$$

$$h_\theta(x) = 2x f_{\psi_s}(x), \tag{5.3}$$

with all associated parameters given in Table 5.1. Using MATLAB R2015a we have calculated the above fit accuracy via coefficient of determination, R^2, for all fitted functions over all measured distances d. For $f_{\psi_t}(x)$ the mean value of R^2 for all distances is $\mu_{R^2}(f_{\psi_t}(x)) = 0.9981$ with its variance of $\sigma^2_{R^2}(f_{\psi_t}(x)) = 5.423 \times 10^{-6}$ and $\mu_{R^2}(g_\eta(x)) = 0.9749$ with $\sigma^2_{R^2}(g_\eta(x)) = 1.8622 \times 10^{-4}$ indicating very good fits for the data. The R^2 for $h_\theta(x)$ obtains lower accuracy due to outliers found in the measured data for θ. Due to the lower fit accuracy for $h_\theta(x)$ individual values of R^2 are given in Table 5.1 for inspection.

5.2 Evaluation of Wisent

We now present the results for the complete Wisent protocol, but we shall only proceed with the evaluation of Wisent EX due to its generic nature. We will use the same experiment setup as in Section 5.1, unless stated otherwise.

As in the case of evaluating `BlockWrite` performance we use tag-to-reader antenna distance d as a parameter to evaluate Wisent. However, instead of `WordCount`, we take message payload size S_{p} as the second parameter.

5.2.1 Wisent Performance Metrics

In Section 5.1 the experiments were executed with a single uninterrupted `BlockWrite` of a set duration, which in Wisent is equivalent to a single message. In Wisent, however, each of the previously used metrics also depends on the number of messages per second the RFID reader is able to process, i.e. the overhead discussed in Section 3.3. Furthermore, a Wisent log file provides information on events that occurred between messages, rather than events that occurred within a predefined time. Therefore, we add t, the runtime of the transfer session, as a variable.

We introduce the following metrics: (i) the number of Wisent messages per second defined as $v = m_{\mathrm{t}}/t$, where m_{t} is the total number of messages sent during the Wisent transfer session; (ii) the number of success operations per message (SOPM) defined as $\psi_{\mathrm{sm}} = n_{\mathrm{s}}/m_{\mathrm{t}}$, where $\psi_{\mathrm{s}} = v\psi_{\mathrm{sm}}$; (iii) the number of total operations per message defined as $\psi_{\mathrm{tm}} = n_{\mathrm{t}}/m_{\mathrm{t}}$ with $\psi_{\mathrm{t}} = v\psi_{\mathrm{sm}}$; (iv) throughput defined as $\theta = 2S_{\mathrm{p}}v$ (B/sec), (v) the resend rate defined as $p_{\mathrm{r}} = \frac{m_{\mathrm{r}}}{m_{\mathrm{r}}+m_{\mathrm{s}}} = \frac{m_{\mathrm{r}}}{m_{\mathrm{t}}}$ where m_{r} and m_{s} are the number of messages resent and sent, respectively, and (vi) η, whose definition is given in Section 5.1.

5.2.2 Wisent Performance Results

We have experimented sending as many messages as possible and observed a value for $v \approx 3.8$ messages per second when using the Impinj R420 RFID reader and only half of that value with an Impinj R1000. This result is not affected by setting different values for `OCV`, which should change the size of the operation frame, and in turn lead to a smaller or bigger message frame. This has been confirmed by testing multiple values for `OCV` $= \{5, 10, 15, 20, 25, 30\}$. For `OCV` $= \{5, 10\}$, the host is observed occasionally not receiving an ACK in each message frame before the operation frame of that message ends and the message is deleted. For `OCV` $= 20$, the deletion of a message commanded by the host, explained in Section 4.2.2, collides with the `AccessSpecStopTrigger` after executing 20 operations causing the reader to misbehave and cease the bitstream. Higher values for `OCV` cause the next message frame of each message to be flooded with NACKs and forces the resend of the message. Only for the value `OCV` $= 15$, the bitstream remained operational. In all cases, however, the observed value for v did not change and therefore we set `OCV` to 15.

Proposition 2 *`OCV` $\leq N_{threshold}$ should hold when selecting $N_{threshold}$ to increase the probability of message ACKs.*

Proof 2 *Let m_i and m_{i+1} be messages with operation frames of length $f_o(m_i)$ and $f_o(m_{i+1})$, respectively. When m_i is acknowledged, i.e. operation frame of m_i has started, the message frame of m_{i+1} starts, causing the remainder of the operation frame of m_i to be NACKs in the message frame of m_{i+1}. Since `OCV` acts as upper bound for $f_o(m_i)$ and $f_o(m_{i+1})$, the message frame of m_{i+1} is now flooded with up to $f_o(m_i) - 1$ NACKs. $N_{threshold} <$ `OCV` would imply that the probability of m_{i+1} getting acknowledged depends on $f_o(m_i)$ not reaching the upper bound.*

For all our experiments, we set $N_{\text{threshold}}$ to 20 noting that proper investigation is required to get an optimal value.

As a first experiment with Wisent EX, now taken in a university *office* instead of laboratory, we transferred Intel Hex files without the throttling mechanism described in Section 4.4.3, containing 5120 bytes of random data, which is around the same number of bytes as an Intel Hex file of a WISP firmware generated by TI CCS. However, files generated by TI CCS have a maximum data length of 16 words per record that cannot be specified by the user. This forces the record to be split into messages of possibly different S_{p} values other than the S_{p} selected for the experiment. Therefore, we created Intel Hex files for our experiments ourselves in such a way that the records in each of the files hold the same number of words as the value of S_{p} in the experiment. The results of each of these experiments with different set S_{p} over multiple distances d are shown in Fig. 5.2. Assuming a rate of 3.8 messages/sec, the runtime of Wisent Basic would be over two times that of Wisent EX using S_{p} of one word (Wisent Basic uses $S_{\text{p}} = 0.5$ and an additional two messages overhead for sending over the address field for each record). The speedup gained by Wisent EX this way only grows larger, the greater value of S_{p} is used. However, a peculiarity in Fig. 5.2 is the operation of Wisent EX at $d = 60$ cm, an unstable operating distance in Fig. 5.1. Instead, this unstable operating distance is shifted to 80 cm, which can be explained by the presence of more metal objects found in the laboratory than in the office.

5.2.3 Selection of Values for Throttling Parameters

We now proceed to discuss the procedure of selecting values for the throttling parameters in Wisent EX.

For the throttling mechanism, recall that $S_{\text{p}}, S_{\text{p}+1} \in T_t$ should hold. Let $T_t(S_{\text{p}})$ and $T_t(S_{\text{p}+1})$ be the indices of S_{p} and $S_{\text{p}+1}$ in T_t. Now, let T_{U}, T_{DE}

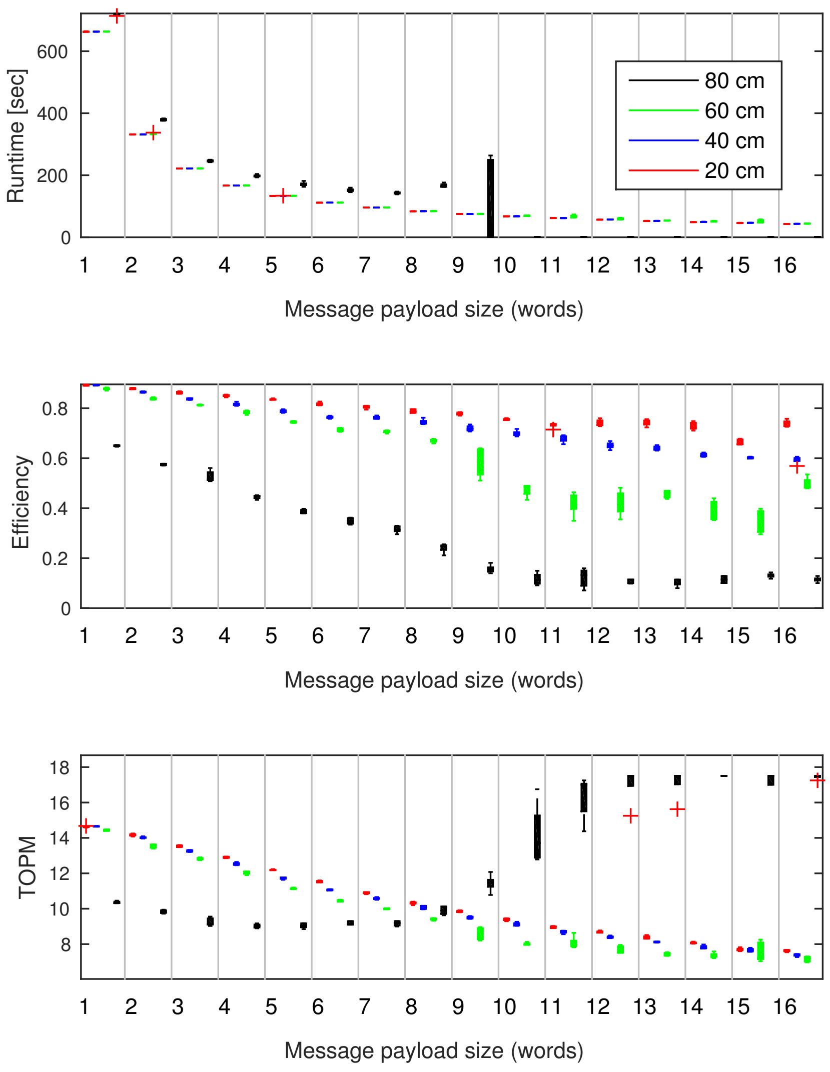


Figure 5.2: Runtime, efficiency and total operations per message during a Wisent EX transfer as a function of S_p over various distances.

Table 5.2: Selected Values for Wisent EX Throttling Parameters

Parameter	T_{U}	T_{DE}	T_{DL}	$R_{\max}$	$M_{\text{threshold}}$
Value	1	–2	–3	3	10

and T_{DL} be denoted as $T_{\text{X}} = T_t(S_{\text{p}+1}) - T_t(S_{\text{p}})$, i.e. the index difference between $S_{\text{p}+1}$ and S_{p}, where T_{U} is used for throttling up, T_{DL} for throttling down in cases where timeouts are caused by NACKs due the reader losing sight of the CRFID (possibly a more severe case than e.g. NACKs due to a mismatched `EPC`), and T_{DE} for throttling down in cases where timeouts are caused by NACKs due to any other reason. We propose the following condition that should hold when selecting values for T_{X}:

$$T_{\text{U}} < |T_{\text{DE}}| \leq |T_{\text{DL}}|\,, \qquad |T_{\text{X}}| \in \mathbb{N}^+, T_{\text{DE}}, T_{\text{DL}} < 0. \tag{5.4}$$

Furthermore, for a message resent for the $R_{\max}$-th time, its S_{p} should be the minimum possible value, i.e. $S_{\text{p}} = T(1)$, even if before any resend of a message, i.e. $R_{\text{count}} = 0$, S_{p} was at its maximum possible value, i.e. $S_{\text{p}} = S_{\text{r}}$. $R_{\max}$ is then found by solving R_{count} in $|T_t| - T_{\text{DE}} R_{\text{count}} = 1$.

The selected values we have chosen for all system parameters in Wisent EX can be found in Table 5.2. We conjecture that the value of $M_{\text{threshold}}$ is related to $|T_t|$, since it decides the speed of which S_{p} converges to a steady state during a set period of time where the communication channel is stable (i.e. the distance from CRFID to antenna is the same for that period). We note that further analysis should be done to reason about the optimal value for $M_{\text{threshold}}$. We nevertheless feel that a value of 10 supports the rest of values we have chosen for throttling.

5.3 Case Study: CRFID Wireless Reprogramming

As final evaluation, we demonstrate the ability to wirelessly reprogram the WISP by using Wisent EX with the selected parameter values in Section 5.2.3. For this, we created a bootloader that is manually programmed to the WISP and is not overwritten after the wireless programming, see Fig. 5.3.

To initiate the wireless programming, the host sends a transfer initialization message as a `Write` command to enter programming mode. This command is recognized by the CRFID so that messages are handled correctly until the programming session is finished. When all data of the application is transferred, the CRC16 checksum over the whole application is sent to validate the firmware. If the checksum matches the calculated one on the CRFID, the programming session was successful and the bootloader will start the application.

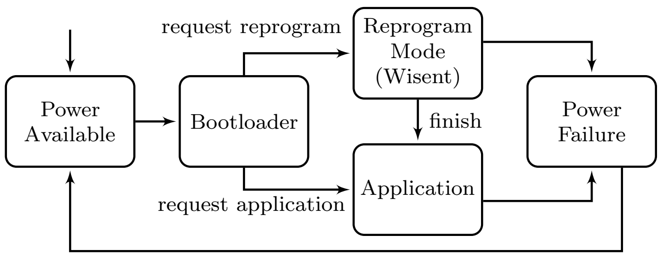
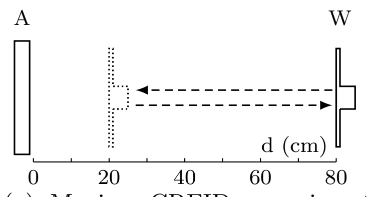
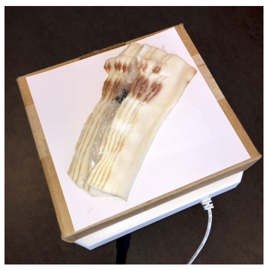


Figure 5.3: Flowchart of the proposed and implemented CRFID bootloader used for wireless reprogramming using Wisent. Compare this design with [10, Fig. 3] and its $n + 1$-way switch to determine which firmware program to run.



(a) Moving CRFID experiment setup



(b) CRFID ex vivo

Figure 5.4: Schematic overview of two experiments used to evaluate downstream data transfer with Wisent: (a) moving CRFID (A: antenna, W: WISP), (b) CRFID ex vivo placed between layers of meat (bacon) on RFID antenna.

Table 5.3: WISP Reprogrammed Firmwares

Firmware	Size (bytes)	Runtime[1](sec)
WISP 5 (base)	5387	54.467
WISPCam	6442	65.097
FM0 modulation[2]	1199	—
Wisent functionality[2]	528	—

[1] From static distance $d = 20\,\text{cm}$ and with $S_\text{p} = 16$.
[2] Difference of sizes between base firmware with and without this functionality. Runtime omitted as patches were not made.

Table 5.4: Comparison of WISP 5 Base Firmware Transfer Performance

S_p	t (sec)	p_r	No. resends	No. transfers completed
throttle	248.448	0.0344	24.4	5/5
1	810.000	0.0004	1.20	5/5
2	462.585	0.0096	13.2	5/5
3	403.974	0.0411	44.0	4/5
4	351.899	0.0951	73.0	2/5
6*	—	—	—	0/5

* t, p_r and no. resends were ommited as all transfers failed due too many resends. $S_\text{p} = \{8, 16\}$ were ommited as they show the same result as for $S_\text{p} = 6$.

5.3.1 Experiment Setup and Results

As experiments, we have adapted a measurement scenario to mimic movement of CRFID and its effect on channel quality using an in-house developed automaton for repeated indoor mobility [27]. For this the WISP is attached to nylon wires, which are then wound up/released by stepper motors (controlled by an Arduino) with a speed of approximately 0.1 m/s to move the WISP and change its distance to the antenna between 20 cm and 90 cm (i.e. between the optimal distance and a distance where the channel is unreliable) in a repeatable manner, see Fig. 5.4(a). Results of these experiments using the constructed bootloader are listed in Table 5.3 and Table 5.4, which justifies the use of the throttling mechanism as proven in Section 4.4.3. In comparison with the case of a set $S_\text{p} = 4$ the throttling mechanism cuts the transfer time of the base firmware down by approximately 100 seconds and even reduces the resend rate with almost three times while finishing all transfer attempts.

5.3.2 Data Transfer Energy Consumption

For result completeness and comparison to typical state-of-the-art WSN devices concerning the energy consumption used in downstream communication, we have used a Monsoon Power Monitor to measure the energy consumption of a Tmote Sky node storing 5387 bytes of data received from another node (the WISP 5 base firmware length, see Table 5.3). To represent

Wisent EX messages in our experiment, the data was sent in chunks of 36 bytes with the X-MAC protocol present in Contiki OS. The measured energy consumption of the Tmote Sky is 256.35 mJ, while the WISP consumed a total of 81.70 mJ. This proves that despite Wisent not being designed with energy efficiency in mind, CRFID downstream transfer is at least three times more energy efficient than corresponding WSN downstream by sacrificing operating distance, which is orders of magnitude better for WSN.

5.3.3 Reprogramming a Tissue-Embedded CRFID

As an ultimate experiment, we demonstrate the ability to wirelessly reprogram a tissue-implanted CRFID. To emulate such a scenario we placed a cling film-wrapped WISP between 5 and 6 layers of meat (bacon) at the top and bottom of the WISP, respectively, placed on an antenna separated by a 6 cm paper box, see Fig. 5.4(b). This experiment followed similar ex vivo experiments emulating implantable sensor scenarios [4, Fig. 8]. We were able to successfully reprogram the WISP with the complete RFID stack within 63.55 sec despite attenuation of the backscatter signals by the meat.

Chapter 6

Conclusion

In this thesis, we have designed and implemented a protocol for robust downstream communication for transiently powered computers, i.e. CRFIDs. Our protocol, called Wisent, allows to transfer bulk data from host to CRFID in a fast and robust manner while maintaining compatibility with existing RFID ecosystems relying on EPC C1G2. Wisent allows to store and verify data despite power interruptions at the CRFID thanks to the use of non-volatile FRAM memory and a simple error verification mechanism. In addition, through the introduction of large frame sizes (sent by the RFID reader) and its length adaptations depending on the channel conditions, Wisent's `BlockWrite` improves the raw throughput threefold in comparison to single word message size supported by the EPC C1G2 standard and allows transfer up to 16 times faster than a baseline using single word messages at sub-meter distance. Finally, implementation of Wisent allowed to introduce and experimentally verify the world's first wirelessly reprogrammable (software defined) CRFID.

6.1 Future Work

Wisent forms a fundamental baseline for further extensions and experiments on efficient downstream CRFID communication. Further features that could be implemented are:

1. *Transfer to multiple tags at the same time*: although Wisent has been designed for transfers to a single CRFID tag, an extension for multiple tags is necessary;

2. *Resume a downstream transfer after a failure*: At this moment no mechanism has been implemented in Wisent to resumes transfer after a failure (whether complete power failure [of either CRFID or an RFID reader], or a permanent movement of a tag away from the antenna). This mechanism should be implemented via the tracking of state of a

file transfer by the RFID reader.

3. *Data compression*: to increase data transferred within the same runtime or reduce runtime with an equal amount of data sent by the reader, a compression mechanism must be implemented at the cost of computational power at the CRFID side for decompression.

4. *Security*: to deploy reprogrammable CRFIDs, data transfer needs to be secured. Wisent as of now has no mechanism to prevent message spoofing. We argue that this is the most urgent feature missing in Wisent.

5. *Energy efficiency*: while Wisent is able to handle the effects of transient power, no design considerations were taken to make it as energy efficient as possible. We argue that improving the energy efficiency could lead to better performance in terms of speedup as was the case with works that improved upstream communication discussed in Section 2.4.

Bibliography

[1] S. Gollakota, M. Reynolds, J. Smith, and D. Wetherall, "The emergence of RF-powered computing," *IEEE Computer*, vol. 47, no. 1, 2014.

[2] S. Jiang and S. V. Georgakopoulos, "Optimum wireless power transmission through reinforced concrete structure," in *Proc. IEEE RFID*, 2011.

[3] Z. Xiao, X. Tan, X. Chen, S. Chen, Z. Zhang, H. Zhang, J. Wang, Y. Huang, P. Zhang, L. Zheng, and H. Min, "An implantable RFID sensor tag toward continuous glucose monitoring," *IEEE J. Biomed. Health Inform.*, vol. 19, no. 3, 2015.

[4] D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *Proc. IEEE Symposium on Security and Privacy*, 2008.

[5] (2015) EPC radio-frequency identity protocols generation-2 UHF RFID. Version 2.0.1. [Online]. Available: http://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf

[6] A. P. Sample and J. R. Smith, "The wireless identification and sensing platform," in *Wirelessly Powered Sensor Networks and Computational RFID*, J. R. Smith, Ed. Springer, 2013, pp. 33–56.

[7] H. Zhang, J. Gummeson, B. Ransford, and K. Fu, "Moo: A batteryless computational RFID and sensing platform," UMass Amherst, Tech. Rep. UM-CS-2011-020, 2011.

[8] (2014) WISP 5.0 firmware git. [Online]. Available: https://github.com/wisp/wisp5

[9] B. Ransford, J. Sorber, and K. Fu, "Mementos: System support for long-running computation on RFID-scale devices," in *Proc. ACM ASPLOS*, 2011.

[10] B. Ransford, "A rudimentary bootloader for computational RFIDs," UMass Amherst, Tech. Rep. UM-CS-2010-061, 2010.

[11] S. Naderiparizi, A. N. Parks, Z. Kapetanovic, B. Ransford, and J. R. Smith, "WISPCam: A battery-free RFID camera," in *Proc. IEEE RFID*, 2015.

[12] Y. Zheng and M. Li, "Read bulk data from computational RFIDs," in *Proc. IEEE INFOCOM*, 2014.

[13] J. Gummeson, P. Zhang, and D. Ganesan, "Flit: A bulk transmission protocol for RFID-scale sensors," in *Proc. ACM MobiSys*, 2012.

[14] Y. Zhao, J. R. Smith, and A. Sample, "NFC-WISP: A sensing and computationally enhanced near-field RFID platform," in *Proc. IEEE RFID*, 2015.

[15] M. Buettner and D. Wetherall, "An Empirical Study of UHF RFID Performance," in *Proc. ACM MobiCom*, 2008.

[16] M. Mohaisen, H. S. Yoon, and K. H. Chang, "Radio Transmission Performance of EPCglobal Gen-2 RFID System," in *Proc IEEE ICACT*, 2008.

[17] P. Zhang, J. Gummeson, and D. Ganesan, "BLINK: A High Throughput Link Layer for Backscatter Communication," in *Proc. ACM Mobisys*, 2012.

[18] M. Buettner and D. Wetherall, "A software radio-based UHF RFID reader for PHY/MAC experimentation," in *Proc. IEEE RFID*, 2011.

[19] M. Dillinger and R. Becher, "Decentralized software distribution for SDR terminals," *IEEE Commun. Mag.*, vol. 9, no. 2, 2002.

[20] N. Reijers and K. Langendoen, "Efficient code distribution in wireless sensor networks," in *Proc. ACM WSNA*, 2003.

[21] W. Yang, D. Wu, M. J. Hussain, and L. Lu, "Wireless Firmware Execution Control in Computation RFID Systems," in *Proc. IEEE RFID*, 2015.

[22] H. Jayakumar, A. Raha, and V. Raghunathan, "Quickrecall: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers," in *Proc. International Conference on Embedded Systems Design*, 2014.

[23] (2015) EPCglobal low level reader protocol. Version 1.1. [Online]. Available: http://www.gs1.org/sites/default/files/docs/epc/llrp_1_1-standard-20101013.pdf

[24] (2014) sllurp git. [Online]. Available: https://github.com/ransford/sllurp

[25] (2015) MSP430 Assembly Language Tools. Version 4.4. [Online]. Available: http://www.ti.com/lit/ug/slau131j/slau131j.pdf

[26] P. Lettieri and M. B. Srivastava, "Adaptive frame length control for improving wireless link throughput, range, and energy efficiency," in *Proc. IEEE INFOCOM*, 1998.

[27] M. Cattani and I. Protonotarios, "Gondola: a parametric robot infrastructure for repeatable mobile experiments," 2015. [Online]. Available: http://arxiv.org/abs/1601.07457