Hardware Implementation of BitSurfing Communication Logic

Dimitrios G. Karagkounis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisors: Prof. *Evangelos Markatos*, Dr. *Sotiris Ioannidis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Hardware Implementation of BitSurfing Communication Logic

Thesis submitted by
Dimitrios G. Karagkounis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Dimitrios G. Karagkounis

Committee approvals: _____
Evangelos Markatos
Professor, Thesis Supervisor

_____
Sotiris Ioannidis
Research Director,Thesis Advisor,Committee Member

_____
Polyvios Pratikakis
Assistant Professor, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, November 2019

Abstract

The demand for realistic, low energy consumption communication protocols continuously increases in the era of IoT. IoT deployments are expected to expand even more in year to come, in order to serve multiple diverse field applications. Reliability and longevity (long autonomy) are two, undeniably, important factors. Problems arising due to the lack of the later, are among the most difficult to address.

In order to increase the reliability and lifespan of IoT devices many research teams have proposed protocols and architectures especially designed for IoT applications.

Recently, a novel theoretical communication paradigm was proposed, called BitSurfing, which can simplify significantly the IoT transceiver hardware, with potential for major benefits in energy-efficiency and security.

The present work demonstrates a proof-of-concept implementation of the BitSurfing logic using low cost Raspberry Pi hardware. BitSurfing nodes take advantage of packets broadcasted in their network and utilize them to enable intra-network communications. Every node maintains a FIFO buffer, in which it stores broadcasted packets. If a node wants to send a message, it waits until the message appears in the buffer and then it sends a low energy 1-bit pulse to inform neighboring nodes. BitSurfing cuts down the required IoT device energy consumption and thus it has a potential use in battery-less systems. In this study, a proof-of-concept deployment on real hardware is presented and the results of BitSurfing evaluation suggest that the proposed logic can be feasibly used for data transfer among network nodes. This study shows that BitSurfing is implementable even with very common hardware and defines an empirical model for its operation.

Furthermore, a multi-threaded BitSurfing simulator is created and tested for larger network sizes and congestion levels. The results derived from the various simulations also suggest that BitSurfing can deliver messages among network devices without using any MAC layer functionality, a characteristic reducing the energy consumption requirements even more.

In conclusion, two specific BitSurfing use cases are presented and promising research directions are highlighted.

Περίληψη

Η ανάγκη για ρεαλιστικά και με μειωμένες απαιτήσεις σε ενεργειακούς πόρους πρωτόκολλα επικοινωνίας ολοένα και αυξάνεται στην εποχή του IoT. Η σχεδίαση και υλοποίηση νέων εγκαταστάσεων IoT αναμένεται να αυξηθεί ακόμα περισσότερο στο ερχόμενα έτη ούτως ώστε να εξυπηρετηθούν πολλαπλές εφαρμογές ποικίλων πεδίων.

Η αξιοπιστία και η μεγάλη ενεργειακή αυτονομία είναι δύο από τους σημαντικότερους παράγοντες που λαμβάνονται υπόψη κατά τη σχεδίαση και υλοποίηση μια εφαρμογής IoT. Τα προβλήματα που δημιουργούνται λόγω αδυναμίας παροχής του δεύτερου από αυτά είναι ανάμεσα στα πιο δύσκολα προς επίλυση.

Έχοντας ως στόχο την αύξηση της αξιοπιστίας και της μακροζωίας (μεγάλης αυτονομίας) των IoT συσκευών, πολλές ερευνητικές ομάδες έχουν προτείνει πρωτόκολλα και αρχιτεκτονικές επικοινωνίας ειδικά σχεδιασμένα για εφαρμογές IoT.

Πρόσφατα, προτάθηκε ένα νέο θεωρητικό πρωτόκολλο επικοινωνίας – το BitSurfing – το οποίο υπόσχεται να απλοποιήσει σημαντικά το σχεδιασμό του υλικού ενός IoT πομποδέκτη, με προοπτική ακόμα και για βελτίωση της ασφάλειας και της ενεργειακής απόδοσης ολόκληρης της συσκευής.

Στην παρούσα εργασία αποδεικνύεται η υλοποιησιμότητα της λογικής στην οποία βασίζεται το BitSurfing, χρησιμοποιώντας απλό υλικό όπως είναι οι συσκευές Raspberry Pi. Βάση της λογικής ΒιτΣυρφινγ που μελετάται, οι κόμβοι εκμεταλλεύονται πακέτα που εκπέμπονται στο δίκτυο τους και τα αξιοποιούν για να επιτύχουν επικοινωνία μεταξύ των κόμβων του δικτύου τους. Κάθε κόμβος διατηρεί ένα καταχωρητή τύπου FIFO, στον οποίο αποθηκεύονται τα ληφθέντα πακέτα. Αν ένας κόμβος θέλει να στείλει μήνυμα, περιμένει ώσπου να εμφανιστεί το μήνυμα στον καταχωρητή του και έπειτα στέλνει ένα παλμό χαμηλής ενέργειας και διάρκειας ενός bit στους γειτονικούς του κόμβους.

Το BitSurfing μειώνει σημαντικά την κατανάλωση ενέργειας σε IoT συσκευές και συνεπώς έχει προοπτική χρήσης ακόμη και σε συσκευές χωρίς μπαταρίες. Σε αυτή την εργασία μελετάται η υλοποίηση της λογικής ΒιτΣυρφινγ σε πραγματικές IoT συσκευές με τα αποτελέσματα να δείχνουν ότι η προτεινόμενη λογική μπορεί πράγματι να χρησιμοποιηθεί για τη μεταφορά δεδομένων μεταξύ των κόμβων ενός δικτύου.

Η μελέτη δείχνει ότι η λογική BitSurfing είναι υλοποιήσιμη ακόμη και με πολύ συνηθισμένο υλικό, ενώ ορίζεται ένα εμπειρικό μοντέλο για τη λειτουργία του BitSurfing.

Με στόχο να καλυφθούν τυχόν προβληματισμοί σχετικά με το μέγεθος του δικτύου στο οποίο μπορεί να χρησιμοποιηθεί η συγκεκριμένη λογική, δημιουργήθηκε ένας προσομοιωτής σε λογική πολυνηματικής υλοποίησης, ώστε να επιτυγχάνεται η κατά το δυνατό ανεξαρτησία μεταξύ των προσομοιωμένων κόμβους BitSurfing. Ο προσομοιωτής δοκιμάστηκε για διάφορα μεγέθη δικτύου και επίπεδα συμφώρησης, με τα αποτελέσματα να συμφωνούν με αυτά της υλοποίησης σε πραγματικές συσκευές και να δείχνουν ότι η λογική επικοινωνίας BitSurfing δύναται να εφαρμοστεί.

Τέλος επισυμένονται ενδιαφέρουσες ερευνητικές κατευθύνσεις αφού πρώτα περιγραφούν δύο συγκεκριμένα παραδείγματα στα οποία μπορεί να χρησιμοποιηθεί η εξεταζόμενη λογική επικοινωνίας.

Acknowledgments

The whole process of studying, implementing, experimenting and writing this thesis was very demanding and time consuming. The guidance of my supervisor, prof. Evangelos Markatos and advisor Dr. Sotiris Ioannidis, was valuable and helped a lot at managing and organizing the whole process to meet my goals. I would like to thank both for that and the study groups they initiated on very interesting research and market topics. My gratitude to Dr. Christos Liaskos and Dr. Ageliki Tsioliaridou for their invaluable guidance and the time they put into me, throughout the period of this thesis fulfilment. Finally, i would like to thank my family and friends for their full support on my choices.

στο επερχόμενο μέλος της οικογένειας

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

IoT applications emerge more and more as technological improvements are introduced and components cost drops [15]. Previously visionary applications, such as Smart Cities and connected Health, are nowadays not only feasible, but also deployed at large scales, providing huge amounts of useful data, used to improve quality of life. The deployment of IoT networks for Tsunami Alert and detection [16] as well as Smart Water Management [17] are only a couple of many remarkable applications indicating the perspective of Internet of Things.

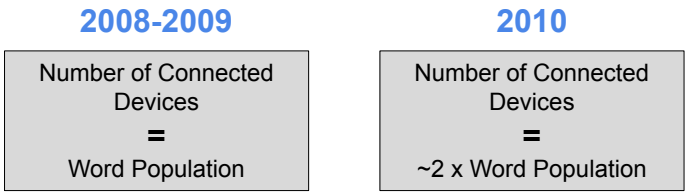

Figure 1.1: Internet of Things Birth [8]

Cisco's 2011 white paper on Internet of Things claims that sometime between 2008 and 2009 was the first time that more devices were connected to the internet than humans. In the same paper, it was stated that by 2020 it is expected around 50 billion devices to be connected. [8] It seems that the forecast overestimated the reality, as another more recent report claims that the number of connected devices will be around 31 billion in 2023. [9]

Most of those IoT connected devices is expected to be short and long range IoT, rather than smart phones, tablets or laptops as shown in Figure 1.2.

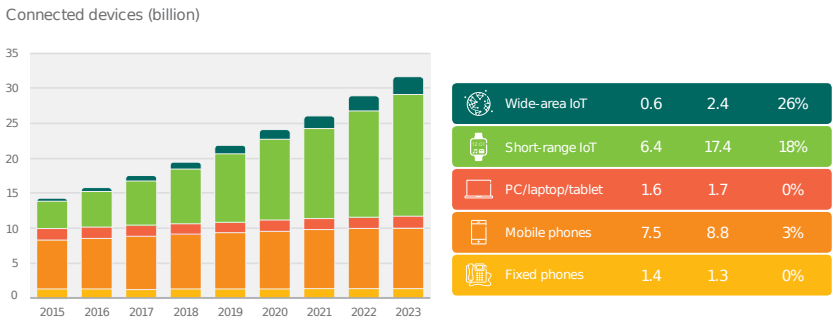

Figure 1.2: Number of connected devices by type (in billion) [9]

According to Ericsson's Mobility report, released in June 2019 [10], two technologies focusing on Low Power Networks, NB-IoT and Cat-M, are expected to account for approximately 45% of cellular IoT connections in 2024.

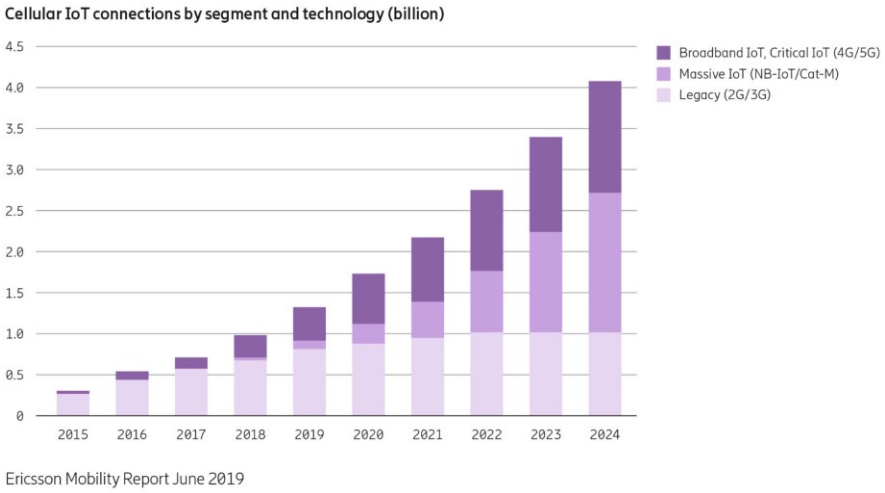

Figure 1.3: Cellular IoT connections (in billion) [10]

Figure 1.3 provides a forecast for Cellular IoT connections by segments and technology. As the bar plot suggests there is expected to be a continuous raise in the number of connected IoT devices in the years to come. The segment defined as *Massive IoT* refers to applications requiring the connection of massive numbers of low-complexity, low-cost, high durability, small data volume and low throughput devices. [10]

Given the high percentage of IoT devices being part of the Massive IoT segment, research on the improvement of any of the segment requirements, would greatly enhance any application supported by Massive IoT deployments.

This thesis studies an IoT communication logic, while also it provides proof-of-concept of the proposed logic, using real IoT hardware devices.

1.2 BitSurfing

IoT communication employs the regular norm: a transmitter creates electromagnetic waves that encode its intended message, and a receiver attempts to convert the waves into useful information. The transmission hardware is one of the most complex and energy-hungry hardware parts for any IoT device, especially if one accounts for the Medium Access Control problem in massive networks [18], [19], [20].

BitSurfing is a novel communication paradigm that breaks this norm, allowing for greatly simplified transmission hardware and miniaturization even at nano-levels [21]. According to it, nodes depend on outsourced generated bit streams to exchange data throughout the network. Each node repeatedly updates a buffer by adding incoming bits, while it waits for a message of interest to appear in its buffer. Once such a message appears, the node sends a simple 1-bit pulse notifying neighbor nodes to view in their buffer and extract the message. The use of simple pulses simplifies the transmission hardware since there is no packet creation. Additionally, the pulse collisions are extremely rare, meaning that Medium Access Control is not necessary even in dense and congested networks [21]. Asynchronous implementation potential is another major advantage of the BitSurfing approach [21].

While the theoretical prospects of BitSurfing have been outlined, its practical realization remained an open question. In this work we provide a proof-of-concept implementation with very common hardware (Raspberry Pi nodes), and outline a practical operation model, challenges and future implementation prospects.

1.3 Challenges

Applications deployed on top of IoT connected devices have certain expectations regarding network characteristics. The network should provide scalability, durability (longevity) and security for the supported application [22]. While scalability is not an issue for most existing communication protocols as it is a major design aspect [23], security and even more durability are two very important characteristics bugging the research community in this field. There are in particular quite a lot of studies concerning security, but in most of them the proposed solutions sacrifice energy efficiency, while all of them require the MAC layer to operate [24]. In other studies, authors present energy efficient communication protocols, but even in such proposals the system cannot operate battery-less.

Bitsurfing adapters main goal is to enable communication at very low energy consumption levels by utilizing an outer bit stream source. In order to achieve this goal the design of the adapter has to be carefully drafted, containing only crucial to the functionality components. The adapter has to be simple enough in order to consume the least amount of energy during its data reception/transmission cycles. To that end the implementation deviates from the traditional OSI model by using

outsourced packet creation and excluding the MAC layer from the adapter's design logic. Another limitation of the adapter made to improve energy efficiency is the lack of any kind of inter-adapter synchronization mechanism.

Considering all the adapter's limitations someone can understand that the only functionalities left on the adapter requiring energy are the incoming data processing and the pulse creation, which is created to inform neighboring nodes of incoming packet. In the proper environment combined with an optimized design of the communication logic, running on ASIC, it is possible to produce a carrier only fed adapter.

This work focuses on the reliability and feasibility of the logic running on top of the adapter, rather than the adapter itself. However, implementation and experimentation on real and of diverse specification hardware devices shows that adapters' buffer may be less similar to one another, as observed in the initially proposed algorithm.

Since there is no clock to use for synchronization among adapters the algorithm needs to be tuned so that any adapter entering the Bitsurfing network, no matter its hardware specifications, will be able to communicate successfully.

1.4 Thesis Contributions

This work offers several contributions to the networking and telecommunications community.

- It provides a thorough analysis of the design, the implementation and the evaluation of BitSurfing's adapter software and hardware components. Considering that Bitsurfing is an energy efficiency centric adapter the major components which enable the adapter's functionalities are : Rx and Tx communication blocks, a buffer to store - potential messages - bits , an (IR) pulse transceiver and a microprocessor running the software logic. The adapter could potentially require no power supply since it can be carrier fed. The proposed software component of the adapter is responsible for handling interrupts and message arrivals as well as sending pulses to other neighbor nodes and thus optimization of this component may greatly impact the adapter as a whole.

- It presents the first BitSurfing proof-of-concept deployment on real hardware IoT devices. The results of BitSurfing evaluation suggest that the proposed logic can be utilized for data transfer among IoT network nodes.

- It provides the adapter's software logic implementation both in a rapid prototyping and flexible programming language − Python − and a highly optimized programming language − C (Kernel) −, which can be compiled even on FPGA devices without much tuning.

- A BitSurfing multi-threaded Simulator is created to support further investigation of the communication logic in various network sizes and congestion levels or even change the functionality of each adapter in order to enhance the tested model.

1.5 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents the necessary background on IoT technologies and refers to related literature. Chapter 3 describes the studied communication logic by analyzing its building blocks, providing also an abstract diagram and an algorithm view of the model. Details of the Testbed Setup and the hardware in use, as well as analysis of the testing scenarios and their results are provided in Chapter 4. Various simulation schemes of BitSurfing are tested in chapter 5, in which the scalability and survivability of the communication logic are evaluated. Finally, potential Research Directions along with conclusions of this study are presented in Chapter 6.

Chapter 2

Background and Related Work

2.1 Background on IoT

Internet of Things refers to a network of "intelligent" objects, which have the ability to communicate with each other and take decisions based on certain criteria. In fact ENISA, the European Union Agency for Cybersecurity, defines IoT as :

> a cyber-physical ecosystem of interconnected sensors and actuators, which enable intelligent decision making.[25]

There are those who claim that IoT is around for quite some time already and other factors such as component cost and new technologies helped it to become a research and commercial trend lately. A couple of famous examples are those of the Internet (ARPAnet) enabled vending machine and the internet toaster. The vending machine remote communication was introduced in 1982 by researchers in Carnegie-Mellon University who wanted to remotely check for cold soda. The second example, the internet controlled toaster was an idea and implementation of John Romkey back in 1990.

Both ideas appeared well before the official IoT term came up. According to Goldman Sachs' authors [15], IoT as a term was originally introduced by Kevin Ashton in 1999. The later stated :

> I could be wrong, but I'm fairly sure the phrase "Internet of Things" started life as the title of a presentation I made at Procter & Gamble (P & G) in 1999. [26]

IoT is one of the main components enabling *Industry 4.0*, one stage of the industrial evolution. [11] IoT is not to be confused with Industry 4.0. The later refers to the automation, data acquisition, self decision making and machine to machine communication in manufacturing. Figure 2.1 represents all the stages of industrial evolution and the most important technologies for each one of them.

Without IoT industry 4.0 and the upcoming industry 5.0 wouldn't be feasible. All the data gathered from hundreds of devices wouldn't exist and Artificial Intelligence wouldn't improve that much in the later years without IoT.

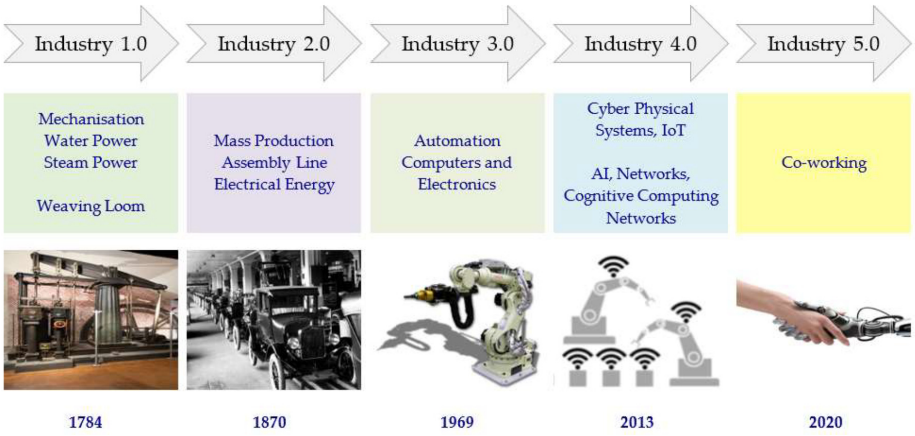

Figure 2.1: Milestones of Industrial Evolution [11]

2.2 IoT Applications

There are numerous segments disrupted by IoT. In fact we can safely say that there is no segment which is not directly or indirectly supported by IoT.

A rather interesting chart, released on January 2018, presents the top ten IoT segments based on 1600 projects globally (Fig. 2.2).

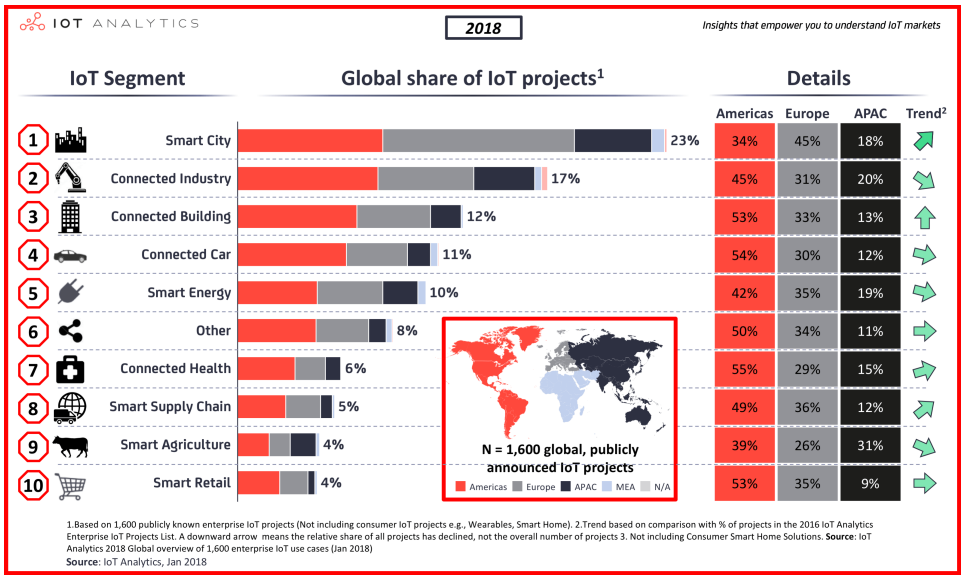

Figure 2.2: Most in demand IoT Applications [11], [12]

Data suggest that most projects are focused on **Smart Cities** IoT solutions, while the interest in this segment still increases. One of the reasons Smart City applications gather that much attention is the number of people directly influenced and the wideness of possible applications aiming to the same goal, improve

the Quality of Life. Among others, some Smart City applications are Traffic control, street lighting, environmental monitoring and public safety. IoT can greatly enhance City Utilities monitoring.

Connected Industry is the runner up in the race of applied IoT projects. Indoors and Outdoors manufacturing can be fully automated, while the infrastructure can be monitored and controlled in real time *fully remotely*. PLCs in a production line can be reprogrammed on demand any time from any place. Quality controls can be scheduled automatically requiring neither the specialist to be there in person nor the production line to stop. Wearable sensors can help employees to look after their personal safety by informing them for potential hazards. A combination of sensors and actuators can be utilized to prevent the destruction of company assets from a number of factors (e.g. high levels of heat or humidity).

Following, the **building sector** has the highest trend among the segments in this chart. Energy costs can be cut down by installing building automation systems (BAS) or other IoT enabled solutions. Heating, ventilation, and air conditioning (HVAC) are power hungry appliances which can be remotely controlled or even better be fully self or system controlled thanks to IoT. This kind of building automation increases energy efficiency and as a result cuts down energy cost. The personal financial interest is likely a main reason why connected building IoT solutions are trending.

In the **automotive and energy industries**, IoT has also made a dynamic introduction. New vehicles contain a network of interconnected sensors and actuators monitoring many useful data, from tire's pressure all the way up to rain sensing system, and taking decisions based on them. Other sensor groups, such as parking or obstacle detection or even road detection sensors, provide a better driving experience. There is also another group of sensors related to cabin comfort. For companies possessing large fleets, there are fleet management solutions available, which provide the ability to remotely check vehicles and run diagnostics. As for the energy sector, the concept of Smart Grids is very popular these days. The grid, through IoT devices, becomes safer. IoT along with artificial intelligence are presented as a promising solution for the prevention of power outages and more importantly wide area black outs. Those technologies can also enhance power flow control over the grid in order to decrease power loses.

Connected Health couldn't be excluded from the segments which can take advantage of IoT. Potential applications vary from Telehealth to Robotic and nano-robotic surgeries. Telehealth refers to the distribution of health services via online systems. Remote patient to doctor contact and care, advice and monitoring are some examples of telehealth. On the other hand Robotic and nano-robotic surgeries can utilize IoT devices or nano-devices respectively to accurately and precisely conduct in vivo surgeries.

Regarding **Smart Agriculture**, IoT can help farmers to better manage their fields. IoT can continuously provide data about the environmental conditions along with other more specific measurements, such as pH and soil humidity. Moreover, gathered data can be uploaded on a platform where the user along with the help of

agronomists will analyze the data and take decisions leading to less product waste and higher production rates.

A very interesting domain which can greatly benefit from IoT, is that of **Supply chain and logistics**. A combination of IoT and Blockchain technologies can provide security, transparency and accessibility. There are such solutions even for tanker and container ships and what they promise aside from those already mentioned are cost reduction and better tracking systems.

As we can see there are numerous applications based on IoT. Take into consideration that this is only a short list of possible applications which are greatly enhanced by this new technology and its advantages. One thing is for sure, the possible applications will only get wider.

2.3 IoT Building Components

IoT has itself a number of important building blocks and although there is a big variety of possible applications, diverse to each other, the backbone of each and every IoT application remains more or less the same.

The architecture is simple yet it is capable of handling information from hundreds of sources quite efficiently.

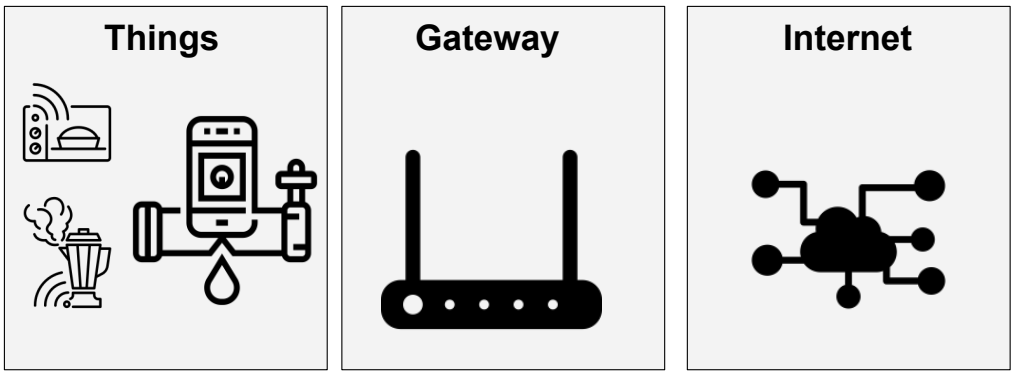

Figure 2.3: IoT Building Components Architecture [13]

As Figure 2.3 represents, the most important components upon which an IoT application operates are:

- **Sensors** : Components of this type were already used many years before IoT was introduced as a term. Sensors allow devices to gather information from their environment. Each sensor is specifically designed to convert measurements of a specific form to another, e.g. environment humidity to voltage, most of the times measurements are converted to an electric pulse being

voltage or current. Accuracy and precision are two extremely important characteristics of any sensor, since devices will later make decisions based on sensed data. Thus the cost of a sensor may widely vary depending the standard error provided by the manufacturer. Some typical sensor examples are: light, temperature, humidity, pressure, accelerometer, touch, proximity and gas sensors.

- **Actuators** : Those devices operate in the opposite way of a sensor. Instead of monitoring the environment and forwarding data to the controller, actuators take action based on signals arriving from the controller. *From a pure technical side of view, actuators are transducers, converting energy from one form to another, just like sensors, but they are differentiated because they don't feed in data to the controller like sensors.*

- **Controller** : This component represent the brain of an IoT device. It is most likely the most important component, as data would be useless if we couldn't process them somehow. A controller is responsible for data collection from sensors, taking decisions based on certain criteria (environmental, provided by the user or critical cases) and passing commands to its actuators in order to achieve the application goals.

- **Gateways** : Just on the edge of the backbone network, is the place where gateways are installed. They are responsible for receiving and sending data from and to sensors and actuators respectively. Those devices are crucial to IoT as a whole. Depending the application, gateways may operate just as a bridge connecting the cloud to sensors/actuators, or they can be utilized as local data senders and controllers. A gateway may be used to encrypt data and messages before they are forwarded to the cloud platform for further analysis and decrypt them before they are forwarded to the controller and the actuator. Gateways can be used for local data analytics, sanitization and compression. Such operations are extremely helpful when trying to reduce the size of data streaming to the cloud for a specific application from thousands or millions of devices, making the data more manageable. Gateways are also unique in terms of communication protocols support. They have to support both IoT low power specific communication protocols in order to receive and send messages to IoT devices, as well as they need to support other, more energy hungry, protocols to communication with the backbone of the network.

- **Platform** : In the core of every IoT application stands the supportive platform. It is the entity which has access to extensive resources, in terms of power, computing and storage, while being also responsible for the front end (most of the times) to the end user. IoT platforms collect data from all over their network and processes them. There are though platforms of many variations mostly based on the specific application they support. So,

there are monitoring and logging centric platform which doesn't provide any data analytics capabilities, while others support many functionalities such as : IoT devices remote control, big data analytics and end user to "things" interaction. We could argue that a good IoT platform is a hybrid solution capable of delivering services upon request. Such a hybrid platform could easily adapt on more than one IoT applications.

- **Connectivity** : Sensor, Actuators, Gateways and Platforms require a communication mechanism in order to exchange data from the edge of the network to the core and vise versa. There is a wide variety of communication protocols strongly dependant on the IoT application specifications, the network requirements and the environmental and resource constrains. Utilization of more than one protocols is a common practice towards ensuring interoperability among network layers. Apart from the more established protocols, like Wi-Fi, Bluetooth, GSM, UMTS and LTE, there is a sufficient number of protocols specifically designed for highly constrained − in terms of power − IoT applications (LPNs), such as : Nb-IoT, Sigfox, LoRa and NWave.

2.4 IoT Connectivity Technologies

We could discriminate three main categories of connectivity technologies for IoT. Those of **short**, **medium**, and **long range** protocols, each of which best supports different applications.

Figure 2.4 provides a very good representation of a number of the most well known IoT communication protocols. Each protocol is placed on the 2d-axis based on its peak data rate in kbps and its maximum range in kilometers, given the data rate. A quick observation of the figure is enough to see that there are many protocols competing in each of the three categories. Each of these protocols has a strong characteristic − e.g. power consumption, bit rate, range, network size (nodes), security, frequency − which makes it the best choice for certain applications.

Several of those protocols are briefly described after Table 2.1, which provides a comprehensive overview of those IoT protocols.

2.4.1 Short Range Protocols

- **BLE** : Bluetooth Low Energy is a protocol member of the Bluetooth family. The main characteristic defining BLE is the very low energy consumption, compared to the classic Bluetooth. It is designed to support IoT applications demanding high autonomy, such as health monitoring wearable devices, intra-automotive communications and home automation

- **NFC** : Near Field Communication refers to a set of communication protocols, designed to support near field applications. It provides support for

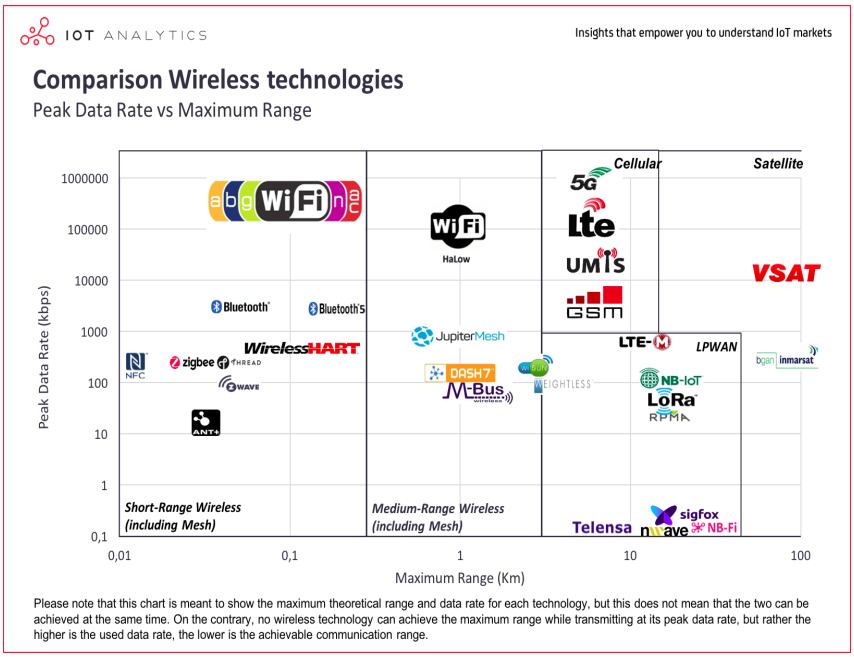

Figure 2.4: IoT Communication Protocols in Categories. [14]

Table 2.1: IoT Communication Protocols basic characteristics [1],[2],[3],[4]

Protocol	Frequency(MHz)	Range(m)	BitRate(Mbps)	Consumption(mW)	Security
BLE	2400	1-100	1	10–500	128-bit AES
NFC	13.56	≤0.2	0.424	2	N/A
ZigBee	2400(868 in Europe)	10-20	0.25	1	128-bit SEK CCM
Wi-Fi(HaLow)	2400/5000	700	1	309.8	AES
Dash7	2400(868 in Europe)	1000	0.166	N/A	AES-CCM
Weightless	470-790	2000	0.1	N/A	128 bit AES
NB-IoT	700-900	10000	0.25	716	3GPP(128-256bit)
LoRa	1000	15000	0.05	297	AES
Lte-M	850	10000	1	N/A	EEAx

point-to-point communications among enabled devices. Each NFC device may function in one of three possible modes, as card emulator, reader/writer device and peer-to-peer. Based on the application the device may require a power supply or it may be RF energy fed. POS-to-Credit card communication is a well known NFC consumer application. NFC keycards and close range file sharing among smartphones are two other applications. Although NFC is widely used in our days, research has shown that there are considerable vulnerabilities of the protocol family [27].

- **ZigBee** : The protocol was proposed as an energy concerned and low cost customizable alternative solution to Wi-Fi and Bluetooth for PANs and LANs. The supported range can reach up to 100 meters based on the frequency configuration and data rates up to 250kbps. It is an established solution for industrial control systems, home and building automation.

2.4.2 Medium Range Protocols

- **Wi-Fi(HaLow)** : Or Wi-Fi 802.11ah as the IEEE standard specifies is a 802.11 communication protocol, specifically designed for IoT applications. The protocol operates at 900 MHz frequency and provides high data rates even at wide range requiring also lower power consumption. Moreover, because of the security offered by the protocol, along with the previously mentioned characteristics, HaLow is a very good candidate for security concerned IoT applications with demand of high data rates, such as security cameras.

- **DASH7** : DASH7 is a protocol initially designed for military logistics. It has a simple, small protocol stack which allows devices to operate on the same battery for several years, thanks to the low power consumption. With data rates up to 167kbps, a functional range of 2km and the 128-bit AES shared key encryption, DASH7 evolved in 2011 from a military protocol to a more commercial supportive protocol. Smart City Logistics and Smart Energy are two examples of IoT applications well suited for DASH7.

- **Weightless** : Weightless is a LPWAN set of standards to enable communication between a base station and thousands of devices around it. Weightless originally had 3 variant but Weightless-P a technology offering bi-directional and narrow band communication in licensed and unlicensed ISM frequencies. The protocol provides adaptive data rates and has the benefit of being an open standard. Scalability depends highly on the way the developer implemented the application.

2.4.3 Long Range Protocols

- **Lte-M** : Long Term Evolution - Machine Type Communication is a low power wide area cellular technology designed by 3GPP. The technology focuses on providing high data rates, mobility and power conservation on its

enabled devices. The downside of those advantages is the increased requirements in bandwidth and the cost of implementation. Because of the years of experience in Lte deployments, the infrastructure does already exist and the protocol is quite secure and thus it is capable of supporting sensitive applications, being inter-vehicle communication.

- **LoRa** : LoRa, which stand for Long Range was not introduced as a full stack communication protocol, rather as a physical layer component. LoRaWAN on the other hand is a data link-layer standard that enables bi-directional communication. The technology is highly customizable and thus it is a very good choice for various IoT applications. It operates on the unlicensed spectrum and it doesn't require any cellular infrastructure, a parameter reducing the overall deployment cost.

- **NB-IoT** : Narrow Band IoT is another (like Lte-M) low power radio technology developed and backed by 3GPP. NB-IoT is mostly suitable for highly penetrating indoor applications. It does support communication at long ranges (10-15km) at rates of 250kbps. Just like Lte-M, the protocol relies on Lte and thus security is one of its advantages. NB-IoT and Lte-M are two of the main competitors on the market supporting cellular IoT applications.

2.5 IoT Challenges

Just like many new concepts, IoT comes with a number of challenges which need to be addressed, at least in some extend. Considering that IoT promises to connect every device to the internet, someone may be thrilled and anxious at the same time. Back in 2011 IPv4 address space was exhausted and at the same moment the demand for such addresses raised exponentially with millions of devices requesting to connect to the internet. With that mentioned the reader takes just a small taste of the challenges arising on the road to delivering IoT applications to the world. Several of those challenges are analyzed below.

- **IPv6 large scale deployment** : Besides IPv6 exists since December 1998, it is far from being considered a mature solution. As already mentioned, IPv4 address space has been exhausted and thus authorities are more or less forced to switch to IPv6, but it may be years before IPv4 is completely abandoned. Right now both Internet Protocols are utilized, a situation adding extra complexity in terms of monitoring and deployment of enabled networks. Given the rate new deployments with hundreds or even thousands of IoT devices show up, each of which requires a unique IP address to communicate, it is easy to understand the difficulty of keeping track of such a network. Apart from scalability concerns, IPv6 is widely used only for several years and thus it hasn't been thoroughly tested for potential security flaws. Last but not least, operators will have to find a way to bind existing legacy systems and devices to the new protocol. [28]

- **Energy Requirements** : Sensor and Actuators as well as other IoT devices require energy to operate. Supplying the required amount of energy for short periods of time is not an issue, especially if devices are already plugged-in somehow to the power supply (cases of indoors IoT devices). The real challenge is to provide longevity to millions of outdoors IoT devices. Consider for example a national wide deployment of IoT devices to monitor earthquake signal for years. It is not feasible to track down each and every device to change its batteries every now and then. Devices need to take advantage of their surrounding environment and harvest energy on demand. Ideally, outdoor IoT devices would be able to function batteryless. On the other hand, the harvested energy will be likely not enough to power devices running legacy, power hungry protocols. To that end, the protocol stack needs to be redesigned with protocols especially implemented for IoT applications, which tend to use minimum amount of resources. [28]

- **Interoperability through Standards and Regulations** : Although IoT is a new technological concept, it has already raised the interest of many industries. The possible application domains are numerous and thus manufacturers and vendors are already competing on a race to bring IoT enabled devices and applications to the market. There are IoT protocols which are developed as a result of joint research agreements or special interest groups with a number of well known companies joining forces, but there are also others introduced and backed only by one manufacturer. As a result there is already a large variety of IoT protocols and architectures released in the commercial market for various application domains. It is therefore very difficult to secure interoperability among IoT networks, devices and the Internet. Such a state, in which IoT devices of different manufacturers won't be able to interconnect will eventually lead to a billion device communication problem. [29]

  International Authorities need to standardize IoT in terms of architecture and protocol stack. Setting also minimum requirements for IoT devices along with a device ranking would be beneficial for consumers. The release of corresponding regulations would be the next step towards management of the IoT industry. Vendors may be able to release their own standards, however they will always have to follow the regulations released by Authorities.

  A step to the correct direction has been made by the International Organization for Standardization in 2018, when they released ISO/IEC 30141:2018 standard [30], a Reference Architecture for IoT. Since international authorities are showing interest in the technology, documentation will be expanded and general rules will be set for interested parties to follow.

- **Scalability** : As mentioned before, IoT is a technological concept which seems like a trend but there are many who claim that it will enable the internet of Everything (IoE) and will be crucial towards developing Industry 4.0

and the upcoming Industry 5.0 as stated in section 2.1. In the introductory chapter is also stated that 31 billion devices are expected to be connected by 2023 (section 1.1), many of which are going to be mobile. Thus the existing telecommunications infrastructure may not be capable of supporting the growing system. There are two levels of scalability to consider, those of network and data scalability.

In terms of network scalability, it is important to design and deploy new networks not considering the present requirements, but the needs of the network in the upcoming future. IPv6 support is a must for new networks. Required security varies based on the application, however designers shouldn't trade off security of some devices to keep the simplicity of others. Moreover, energy requirements of network devices is a top priority since as the number of devices raises from hundreds to thousands or even billions, power consumption adds up to intractable numbers.

As of the data scalability side of view, there are continuous (probably streaming data) flowing in the network. The system won't be able to handle beyond a certain amount of data. Thus data sanitization and maybe a couple of preprocessing stages shall take place on the edge of the network, that can be gateways or even IoT devices themselves. Securing the confidentiality and integrity of the transferred data to the core database as well as assuring the availability of the data are three factors crucial in priority. [1]

- **Security and Privacy** : Two of the most important factors when it comes to IoT is Security and Privacy offered by the design of the underlying network.

  In a continuously increasing network of heterogeneous devices and protocols, it is very difficult to ensure that the whole systems remains secure. The attacking surface of the network widens each time a new technology is added to the existing network. Taking also into account that IoT is mostly built upon wireless communications, adds one more concern to the system designer.

  Moreover, providing privacy to IoT users and the data they access through IoT applications is of great importance, especially in cases of sensitive data such as those related to health and user well being. [31]

  Given the fact that IoT is not a mature technology, researchers are more interested in creating energy efficient technologies than providing security and privacy to the system. Considering that IoT applications are vulnerable to DoS attacks supports this claim. Thus, IoT related technologies designed with a security and privacy awareness are most likely to be preferred by the end user. [1],[29]

  In 2018 the European Union published a revised edition of the General Data Protection Regulation (GDPR) which was enforced throughout the Union. That is a factor preserving higher security and privacy standards for any new

technology developed in the EU. Confidentiality, Integrity and Availability (CIA) of data generated by IoT devices can be preserved at least at a certain level, but such a choice usually comes at a cost, in terms of underlying communication technologies and actual cost of components. The development of lightweight security protocols and highly optimized IoT ASICs, in order to minimize energy consumption while maximizing security awareness is very important.

In this work, we try to tackle 2 major challenges, energy consumption and development complexity, while the proposed technology provides also a level of privacy to the transferred data.

2.6 Related Work

Communication protocols with an energy efficiency awareness have been studied earlier in other works focusing on IoT applications. Although there are numerous studies suggesting improvements in the routing algorithmic and node architecture sides, there are very few approaches challenging the classic packet creation/reception pattern.

2.6.1 Low Power IoT Communication Patterns

An alternative to the classic communication pattern in the category of low power IoT protocols utilizes the **time dimension** to achieve communication among devices.

Zhu and Sivakumar proposed a *Communication through Silence (CtS)* scheme as an alternative communication approach, which by design requires far less energy than conventional protocols [32]. As stated in their paper, their protocol introduces a time dimension as a parameter and relies on inter-node synchronization for successful data transfer. More specifically, as shown in Fig. 2.5 if a node wants to send a packet over the network, it emits a pulse notifying the receiving node of incoming data. The receiving node has a *High Resolution Clock* which has to be synchronized with that of the sending node, based on which a counter is incremented. The sending node has to emit a second pulse to indicate end of message with a high level of precision. Although the protocol conserves energy compared to other IoT protocols, it also has important drawbacks, among which are: collisions among messages and the need for very high resolution Clocks. In addition, a potential lost pulse indicates directly to a lost or differentiated packet, thus the protocol is very sensitive to environmental changes. Furthermore, due to the time dependence the protocol cannot improve much more, since a reduction of the sending/receiving time will most probably result in a different message delivery to the receiving node.

The Authors in [32] use a very simple idea to derive their model energy consumption. They consider only the energy required per transmitted bit, since data

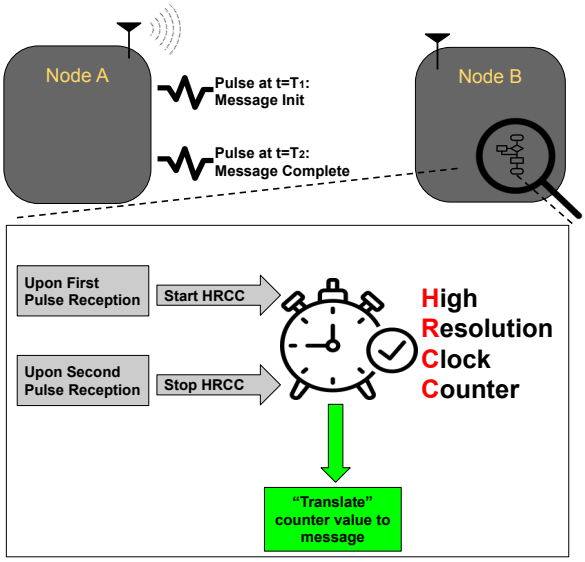

Figure 2.5: Communication Through Silence Logic

transmission is among the most power hungry operations. Thus, since their CtS model requires one bit in the beginning of the packet and another in the end, they claim that their model requires only $Energy = 2 * e_b$ for each packet, where e_b is the energy required per transmitted bit. Their proposal improves the classic packet transmission energy requirements, which requires $Energy = (packetsize) * e_b$. Following the same logic BitSurfing adapters conserve half the transmission energy of CtS adapters since they only need $Energy = 1 * e_b$ in order to send a packet. This is a very good improvement considering also the fact that BitSurfing requires no synchronization among adapters in contrast to CtS scheme.

Authors in [33] indicate CtS weaknesses and propose a *variable-base tacit communication (VarBaTaC)* in contrast, in order to address CtS delay issues. This pattern utilizes the CtS logic coupled with variable-base radix information coding to achieve lower communication delay. The authors also propose three MAC layer variations to be used along with VarBaTaC. The usage of medium access layer adds extra energy overhead to the system and complicates the adapter architecture.

Very similar approaches relying on clock synchronization are presented in other studies. In [34] the author proposes a *redundant radix based number (RBN) encoding and silent periods* to achieve energy conservation, while also proposes enhancements on the MAC layer. A timing channel-based MAC protocol, which encodes data in the time dimension using pairs of bits is studied in [35] to reduce energy needs in nanonetworks .

Another approach deviating from the typical pattern is this of **Optical wireless communications (OWC)** [36]. Nodes take advantage of the light speed and the photon energy in order to achieve data transmission among devices. OWC is

most likely the future of high speed wired and wireless communications. In OWC the source embeds data in light beams by altering the electromagnetic wave. The transmission energy is calculated by $E = h * f$, where f is the frequency and h is Planck's constant. Consequently, the higher the frequency the higher the potential energy to be transmitted across the network or p2p. The transmission energy might even be used to power all nano-devices of an Optical network (e.g. usage of solar cells) [37].

Outsourcing device power hungry functionalities is yet another way towards achieving energy conservation in IoT. Research teams have detected and analyzed the benefits of outsourcing mostly cryptography related functionalities in their works.

Authors in [38] propose an outsourced privacy preserving and verifiable decryption scheme which can be applied directly on IoT applications and enhance system efficiency. Similar studies are conducted in [39] and [40].The first comprising the outsourcing of decentralized multi-authority attribute based signature (ABS). As described, a signing cloud server is utilized to perform the energy hungry signature creation on behalf of other devices. The later paper suggests the outsourcing of computations regarding Public Key Cryptography without sacrificing device privacy and security.

Moreover, researchers in [41] proposed and analyzed a privacy preserving semi outsourced scheme for secure data transmission, named SOPP. The scheme presumes one way authentication from IoT devices and the public (untrusted) cloud, while the energy consuming encryption/decryption hash is derived in the data center side for all IoT devices. Experiments suggest that SOPP is a good choice on small to medium scale deployments, where untrusted public cloud solutions are preferred due to cost benefits.

The need of outsourcing IoT functionalities from limited resource devices to more powerful is indicated multiple times in [42]. Authors presented a number of IoT systems limitation and proposed solutions to overcome the problems. Sensor function virtualization (SFV) is such a solution which distributes computational requirements by utilizing modular functional blocks in any network device.

An interesting case in which multiple technologies were utilized to achieve the application goal is that of Cricket localization system [43]. Researchers took advantage of both ultrasound and RF capabilities to develop a highly accurate and precise localization system. The scheme requires wall and ceiling device deployment which transmit RF beacons and ultrasonic pulses subsequently. Once mobile devices receive the RF beacon start a counter which is stopped upon ultrasonic pulse arrival. After running the algorithm computations a wall-to-mobile device distance is calculated.

As we can see, disruptive ideas such as outsourcing IoT device functionalities, utilizing optical wireless communications and using multiple technologies to communicate have already been studied. Nevertheless, combining all those ideas to take advantage of all their benefits hasn't been studied, to the best of the author knowledge, except BitSurfing.

BitSurfing was initially introduced in [21] as a new nano-networks communication paradigm without any dependence on node clocks. Authors analyzed potential benefits of employing an outer bit generator and supported their thesis by presenting simulation results. Results suggested that BitSurfing has low energy demand (could potentially run on battery-less nodes) and almost perfect packet delivery rates regardless the network congestion levels. However, in the present work, BitSurfing is considered as a communication solution for IoT applications and it is tested on real IoT hardware devices. A proof-of-concept is provided, showing the feasibility of the logic on real systems, based on a tested model.

2.6.2 Energy Harvesting Sources

Considering that the goal is to create a battery-less adapter capable of communicating by utilizing the proposed communication logic, it is useful to show the energy harvesting capabilities for such a device.

The idea of energy harvesting in order to reduce a device self-consuming requirements is around for quite some time. Karl A. Faymon from NASA [44] researched Mars airplane powering options in which was microwave beam energy harvesting back in 1990. His study showed that utilization of microwave beams was an efficient solution for the Mars airplane powering system. The downside of the microwave beam powering solution is the demand for Line-of-Site between the transmitter and the receiver (airplane).

Authors in [7] provide some very interesting insight in their study on wireless power transfer systems. By comparing several EM based technologies they show that wireless energy transfer is a feasible option.

Similar results are provided in [6], where an RF energy survey is conducted. The authors comment on several other wireless energy transfer studies and compact the results in a presentable manner. They take into consideration the Friis equation,

$$P_r = P_t \frac{G_t G_r \lambda^2}{(4\pi d)^2}$$

, to theoretically calculate the transmitted power.

Authors also mention that using an isotropic RF transmitter with source power of *4 Watts* at 902-928MHz band and at a distance of 15 meters, we can harvest efficiently 5.5 μW. Just for an indication of the sizes, consider that a simple arduino uno needs 29 mWatts to operate. There are companies though promising to harvest power up to mWatts [45].

Table 2.2 provides insightful data on several energy harvesting technologies. Microwave energy beams and solar energy seem to be the most efficient solution for longer distance applications.

As Cao and Li indicated in their study [46] solar energy harvesting has the highest power density but is only available during daytime. RF energy is available all day long but has lower efficiency. Especially RF signals of medium frequencies (0.3-3MHz) are capable of transferring energy to long distances (up to few

Table 2.2: Energy Harvesting Technologies [5],[6],[7]

Technology	Efficiency	Range	Coverage	Remarks
Wired	90-95%	As desired	limited to wired deployment	Static deployment, Cost
Microwave beam	30-80%	>2Km	Narrow beam	Potential hazards,Line of sight required
Magnetic Resonance	45-90%	1-2m	Omni directional	Limited range
Reflected Solar Energy	>90%	>1Km	Narrow or wide beam	Daytime only, Line of sight required
RF Energy	70%	12-14 m	Omni directional	Limited range
LASER beam	10-18%	1Km	Narrow beam	Potential hazards,Line of sight required

kilometers) without requiring any line-of-sight.

New and enhanced energy harvesting techniques continuously appear in the research community. An example is [47], in which researchers harvest energy from water flow over Graphene, the 2D material. Authors claim that their Graphene based solution is able to harvest up to approximately $175W/m^2$. With such densities we could be able to continuously power IoT devices.

Chapter 3

BitSurfing Adapter Architecture

3.1 BitSurfing Model Logic

BitSurfing is based on a simple yet powerful concept, thoroughly described in [21]. No packet creation process takes place on any of the communicating nodes. Rather, an outer bit-stream source is employed to create and broadcast packets to the network.

Every BitSurfing enabled node contains Rx and Tx blocks, used to receive and transmit data respectively. The **Rx block** is constantly enabled, receiving bit-streams, which are stored into a FIFO buffer. As long as the packet interarrival time ΔT is greater than the processing time PT required by each node, all network nodes have identical buffers.

A **Codebook**, of predefined $\mu bits$ long words, is utilized for the intra-network communication. Any byte sequence can be translated, through a HashMap, to a Codebook's word sequence. The Codebook is created to discriminate valid and invalid messages, according to a selected prefix. E.g., in the studied network of [21] the prefix has a size of 4 bits.

That said, once a node has message to send over the network, it uses the HashMap to convert the message to its corresponding Codebook's word sequence. Then, it searches the updating buffer for the first hashed word to appear. Once the later happens, the **Tx block** is activated to send a low power pulse (1-bit), in order to inform neighbor nodes of the incoming message as shown in upper part of Fig. 3.1.

Upon signal detection, the receiving node drags the latest valid word observed or searches its buffer for the next valid word (i.e. Codebook word) to appear. The choice of the most appropriate word depends on node parameters as well as BitSurfing Network configuration. Once found, the word is translated back to its corresponding original bit sequence, using the same HashMap as the one used by the sending node.

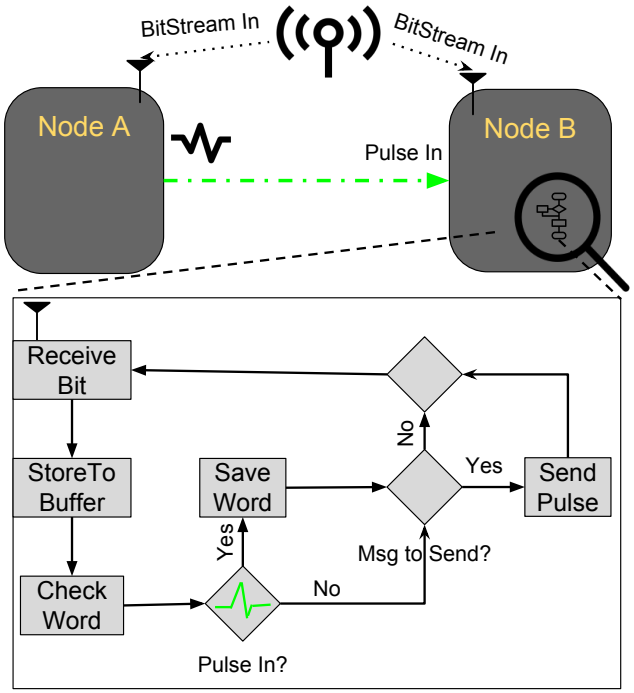

Figure 3.1: Illustration of the proposed Communication Logic

The lower part of Fig. 3.1 illustrates the workflow logic running on each BitSurfing enabled node. The proposed logic may slightly vary among network deployments, but the underlying hardware will remain mostly the same.

3.2 Adapter's Hardware Components

A BitSurfing node requires at least the following components to function. A microcontroller or a microprocessor, a RF receiver, a RF transmitter or UltraSound transceiver or InfraRed transceiver, or simple light sensor along with a led and a small sized buffer.

Microcontroller : Due to the simplicity of the whole system logic, there is no need to use a powerful and power hungry multi-core microprocessor along with any peripheral components. Each incoming bit can be efficiently processed by a microcontroller, which contains any needed components (read-write and read-only memories).

RF Receiver : An RF receiver is required in order to feed in data to nodes. The RF unit may also have a unique design for energy harvesting through RF signals. Regarding the operating frequencies two cases shall be distinguished. The first assuming that nodes will use "junk signals" to communicate, i.e. signals sent among other devices in frequency bands which are widely used (e.g. 900-1200MHz, 2.4GHz and 5GHz). In the second case the deployment shall also contain

a BitStream Source, set in a specific frequency in order for the nodes to listen.

A ROM : In case a microcontroller is utilized in node design, there is no need for a separate Read Only Memory, since it has an integrated one. In other case, a ROM is required to store the executed binaries and other important application predefined and generated data, such as the network map, the Codebook and any log files.

An Optical Transceiver : Given the selected type of deployment, the inter-node communication among network devices may be accomplished with a combination of RF and Optical chips. In that case, the optical transceiver is responsible for sending light pulses to neighbor nodes (a led can also be used to send pulses), while it is also responsible for sensing light pulses from other BitSurfing nodes.

Other components may also be installed on the adapter depending on the requirements of each application. For example, if an application is created to monitor temperature then the adapter will also carry a temperature sensor. A Battery is not included in the most important components list since nodes might operate batteryless, by harvesting energy from their environment.

3.3 Codebook Selection and HashMap Creation

As mentioned in the beginning of this chapter, the adapter utilizes a HashMap to convert messages into Codebook words, which are later used for intra-network communications.

A word of size $\mu bits$ long needs $\frac{2^{\mu}}{bitRate}$ time to appear in buffer, using as source a random bit generator. Thus, choosing a small to average value for word size μ is preferred, since the experiments are not simulated, rather they are conducted on real hardware systems.

For the selection of the word size and the optimal Prefix size, we can consult Figs. 4 and 5 of [21]. The selection may vary based on the application. Besides, if the supported application uses ASCII [48] characters, then there are 256 possible characters, while the Codebook size may be far bigger. At this point, in order to fully utilize the Codebook, we need to configure the mapping between Codebook valid words and characters.

Considering that ASCII characters are used for the application − i.e. there are 256 characters − and the Codebook may have a size 2,3 or even 10 times the number of ASCII characters, there are too many Codebook words unassociated.

One way to fully utilize the Codebook is to associate each ASCII character with more than one Codebook words. This way, every character will have multiple chances of appearing in the buffer in less time.

Another idea − given that English language messages are sent − would be to use combinations of frequent two character words in the mapping process with the Codebook words. By doing that, there is a chance to send more data using just a single pulse.

3.4 Timing Constrains

For this communication protocol to operate efficiently, the total required processing time (PT) has to be less than the packet interarrival time (ΔT) for all nodes in the network.

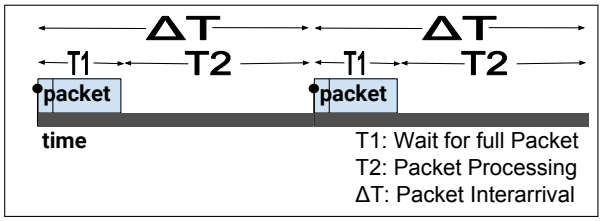

Figure 3.2: BitSurfing phase timings

Figure 3.2 demonstrates this requirement visually. *T1* corresponds to the time required by any network node to fully receive a packet, given that the packet is already in the node's network adapter queue. In addition, *T2* corresponds to the time required by each node to process in packet data and send or receive pulses depending the case, as shown in the lower part of Fig. 3.2. *T1* and *T2* combined provide the total required processing time PT as shown in equation (3.1).

$$PT = T1 + T2 \tag{3.1}$$

PT is strongly bounded to node hardware and software specific parameters and thus it can not be treated as a constant value. On the contrary, ΔT is not related to any hardware specific parameters, rather it depends on the size of each packet (differs among applications), the supported link data rate and the actual distance from the source, among other parameters. ΔT is more or less constant for a specific application, given that no network or spatial changes take place.

During $T1$ interval, no packet processing operation takes place in any node. Any such operation is proceeded during $T2$ time interval. Packet processing depends highly on the underlying hardware of each node and therefore is the most important timing which needs to be adjusted in order for all network nodes' buffers to match.

Chapter 4

BitSurfing : Proof of Concept on Real IoT Devices

4.1 Implementing The Logic on Real Hardware Devices

Based on the above theoretical description and the study conducted in [21], BitSurfing is capable of supporting IoT applications with minimal energy requirements. What's left to be confirmed before optimizing the design of the adapter by creating a ASIC or an FPGA design is to actually prove that the logic is functional on real and low cost IoT devices. To that end, a − Proof-Of-Concept − case study which utilizes the technology is described in this chapter.

In the first section of the chapter, the specific logic implementation, hardware setup and algorithmic logic are extensively analyzed, while in the second part of this chapter the evaluation of the studied case is provided. The results of each conducted experiment are analyzed and improvements considered during experimentation are applied and tested on the logic.

4.1.1 TestBed Setup

For the needs of the analysis, a file transfer scenario is considered. More specifically the objective is to successfully transfer an English text file between two different IoT devices, always based on the BitSurfing logic. The file has a size of 100 bytes and is always transferred from a platform (IoT device) of higher clocked processor to a platform of lower clocked processor.

In order to further strengthen the supported paradigm, we used two pairs of IoT devices, three of which are of different vendor and specifications.

Figure 4.1 illustrates the topology of the testbed, while Fig. 4.2 shows the actual hardware in use. A separate device (laptop) operates as the network's bitStream source, running a live stream application for the data (bits) generation which are transmitted through the Local Area Network. Packets follow the route

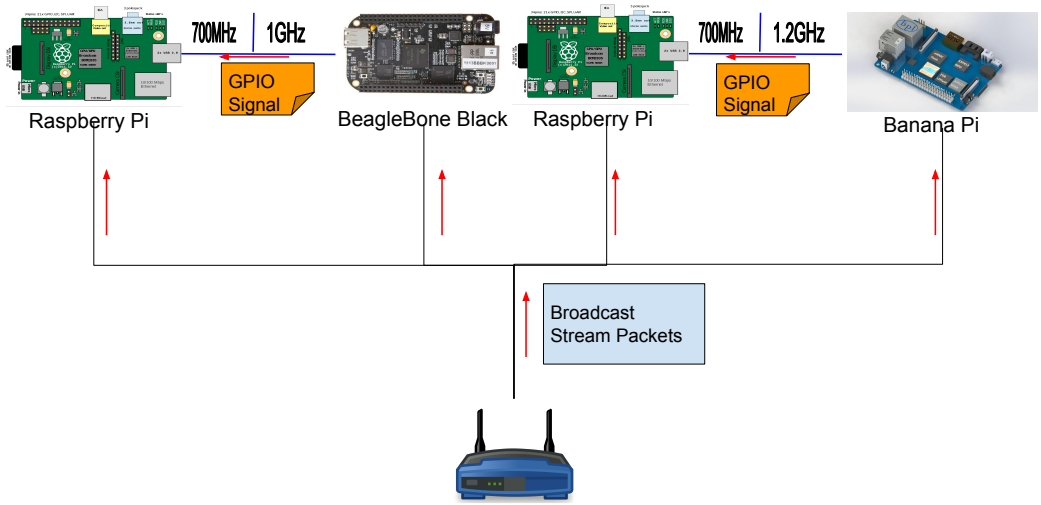

Figure 4.1: Detailed Topology

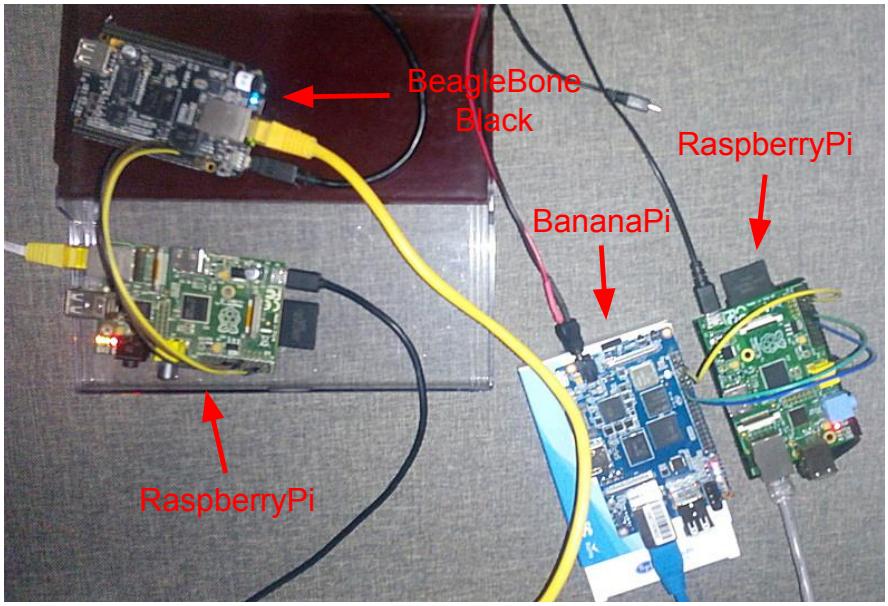

Figure 4.2: Testing equipment

$laptop \rightarrow Router \rightarrow BitSurfing\ nodes$, in a broadcast manner. The first part of the connection, between the bitStream source (laptop) and the Router is wireless − using a 54Mbps 802.11g wireless link −, whereas connections between the Router and all BitSurfing enabled devices are wired, using 100 Mbps Ethernet links.

So to get everything straight, instead of the theoretical wireless BitStream signals of Fig. 3.1 we use a combination of wireless and wired connections through a Router to feed in the same data network wide. Subsequently, instead of the simple and low energy light pulse we use simple GPIO pulses between devices. Other than that there isn't much difference between the theoretical and the actual models.

In addition, the temporary FIFO buffer is configured to has a size of 32bits for all nodes of the network. The size of 32bits is preferred due to word size, which is 16bits. Since the buffer is 32bits, it is capable of fitting up to 2 whole words in it. The size could be configured to fit many more words but such an arrangement might cause technical issues to the implemented logic.

BitSurfing devices are connected in pairs through their GPIO interfaces. To ensure reliability among platforms, ground pins of paired devices are also connected, apart from pins which operate as input and output respectively.

4.1.2 Hardware Specification

The studied network consists of two Raspberry Pis [49], a BeagleBone Black [50] and a Banana Pi [51]. Table 4.1 provides information regarding each platform specifications, as well as how access to GPIO pins is achieved. Several different approaches were considered, among them: widely used libraries, access through sysfs interface and direct register access. Tomaz Solc in [52] and Joonas Pihlajamaa in [53] provide results after comparing several GPIO access approaches. In both cases, experiments led to the conclusion that Direct Register Access (DRA) has the highest response speed among all tested scenarios. GPIO interrupt-response time can hugely impact the $T2$ timing and subsequently the required total processing time, PT. Based on these results and considerations DRA is selected as the method to access GPIO pins in this study.

4.1.3 The selected Codebook and Mapping Logic

As explained in section 3.3 there are many Codebooks, as well as Mapping techniques to choose from.

In this case study the selected Codebook, uses "$0b1110$" as prefix which results in a Codebook containing 2031 valid 16bit long words. Selection of this specific prefix and Codebook is based on the intend to maximize word expected cover time as calculated in Fig. 7 of [21]. Taking also into consideration that any message is a sequence of characters and every extended ASCII [48], [54] character has a value in the range $[0, 255]$, it is clear that only 256 out of 2031 Codebook words are required to map all the characters. In order to fully utilize the Codebook, a HashMap is also

Table 4.1: Testing Hardware Specification

Platform	Raspberry Pi Model B, Rev2	Beagle Bone Black Rev.A5C	Banana Pi M3
CPU	1core-700Mhz	1core-1Ghz	8core-1.2Ghz
Boot	microSD (class10)	eMMC	microSD (class10)
GPIO Access	Direct Register Access	Direct Register Access	Direct Register Access
Architecture	armv6l	armv7l	armv7l
OS	Raspbian	Arch Linux	Arch Linux

created for the rest 1775 words in the Codebook, matched with 2byte sequences. Since there are 65536 possible 2byte words and only 1775 available Codebook words to match, a script is created to derive frequent 2byte words of English text based on data collected from [55], [56], [57] and Linux "*/usr/share/dict/words*" wordlist.

After the HashMap Table is created, any message can be mapped to a valid Codebook word sequence. Message compression is also possible if the message contains any of the 1775 2byte sequences. In such cases the overall transmitted message size decreases without any loss of information.

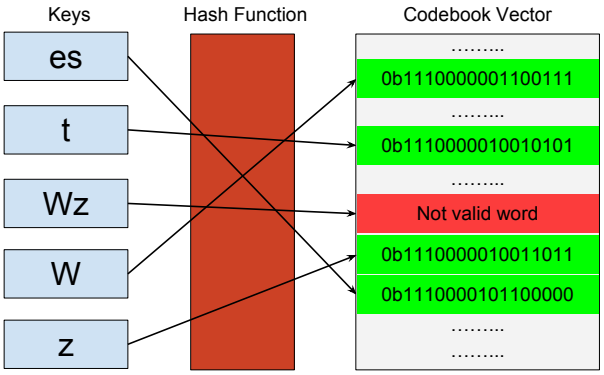

Figure 4.3: Keys - Codebook's word mapping

As illustrated in Fig. 4.3 the word "es" has a corresponding valid word in the Codebook and thus it is successfully mapped. The same stand for characters "t". On the other hand, the word "Wz" is not associated with a valid word and it has to be broken down further to its consisting characters "W" and "z".

Since the valid word size is configured to 16bits, the packet format is adjusted as shown in Fig. 4.5. A 4 bits prefix is utilized to distinguish valid from invalid words. Following, the Sender and the Receiver IDs both requiring 5 bits each for their unique identifiers. Finally, a 2 bits long block containing the actual

application payload. This scheme has a downside though. The combination of $Prefix + Sender\_ID + Receiver\_ID + Payload$ may contain the Prefix more than once and thus there will be a number of adapters with IDs which cannot send valid messages to a number of adapters with a certain ID. The scheme issue is presented in Fig. 4.4. As someone may observe there are nodes which can send valid messages only to 1 or even no other nodes, so these nodes need to be configured as gateway nodes. There are also other nodes (13 of them) which can send and receive valid messages from all network nodes. The designer can overcome this issue by carefully assigning the id for each node.

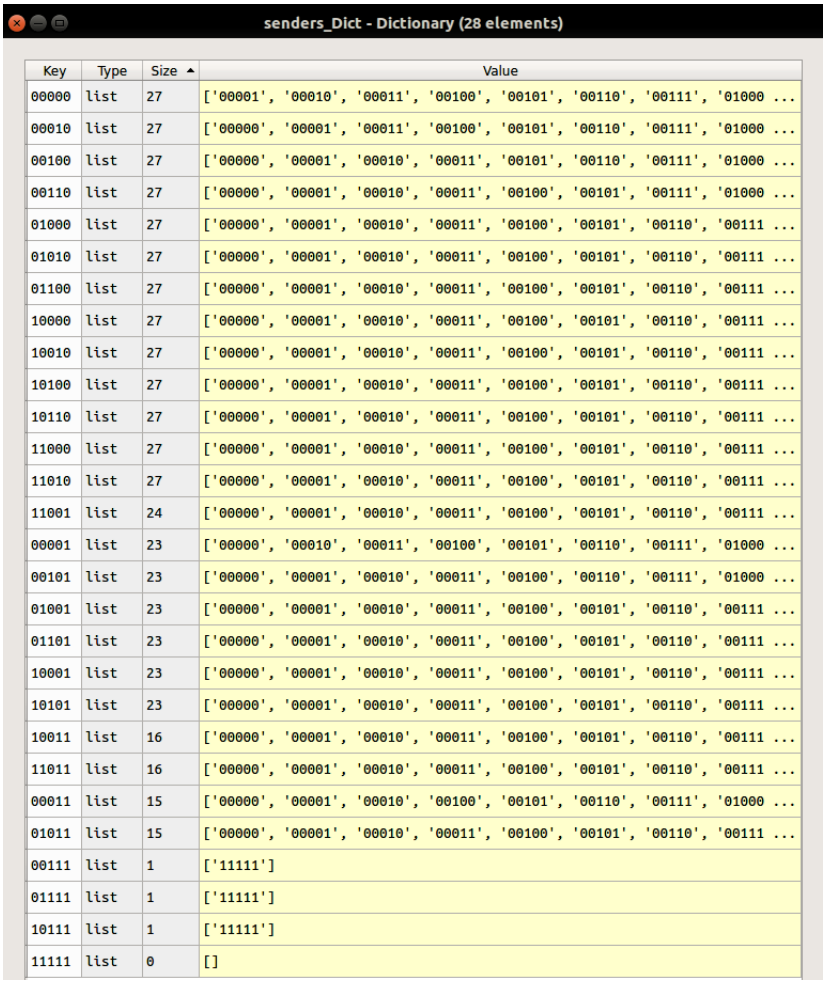

Figure 4.4: 4bit Prefix Valid Unique Ids Pairs

An alternative scheme can also be used − this one was indeed used in this study −, for point to point communication between BitSurfing nodes. In this case the nodes don't look in their buffers for the valid Prefix and their ID, rather they do accept as valid words any sequence starting with the predefined Prefix which

has to be unique in the 16bit long message, as shown in Fig. 4.5 format (b).

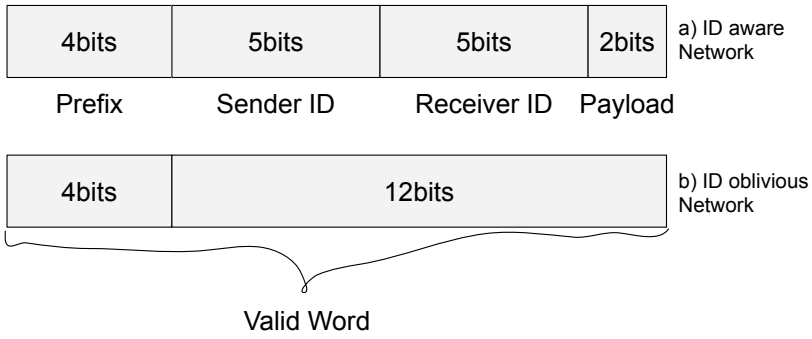

Figure 4.5: BitSurfing Packet Format

4.1.4 Description of the underlying Algorithm

In the "heart" of each BitSurfing node lies the adapter's logic. This logic is provided in the process denoted as Algorithm 1, which shows the code execution flow of a BitSurfing enabled node.

Algorithm 1 Node executed PseudoCode

Inputs: Codebook, HashWords, [Data to be sent]

```
1: Initialize GPIO, open UDP socket, create HashMap, [load Data to be sent]
2:
3: while terminate==false do
4:     receiveNextPacket();
5:     for byte in pakcet do
6:         for bit in byte do
7:             if validMessageFound then
8:                 store Message to Valid Message Buffer;
9:                 if validMsg == TerminationMsg then
10:                    terminate = true;
11:            if GpioEventDetected then
12:                store latest valid message;
13:            if message to Send in Buffer then
14:                sendGPIOPulse();
15:                getNextMessageToSend;
16: Use HashMap to convert Codebook messages to original Messages
```

Mandatory **input parameters** are: the *Codebook* containing all valid − based on the selected prefix − predefined words and a *HashWords file* which contains all of the words or characters to which Codebook words correspond to. The size of

the Hashwords file should be at least the same to that of the Codebook, otherwise predefined words of the later are not associated to any real word and thus they are ignored by the protocol every time they appear in buffer. An optional input parameter is the *Data file* to be sent over the network. If no Data file is provided the node will operate only as receiver.

During the **initialization** stage, the platform's GPIO pins are configured as inputs and outputs, a UDP socket is created to be able to receive the bitStream later, while HashMap is created between words in the Codebook and the Hash-Words file. As a final step of this phase, Data files, if any, are loaded and translated in Codebook words using the HashMap created previously.

As soon as the initialization phase is complete, the actual communication begins, inside the **loop**. In the beginning of each iteration the node halts until the next packet is fully received − **T1** timing −. If an iteration takes more than time ΔT to complete, then there will be a packet fully received in the queue and ready to be processed. In such cases the node does not need to halt, rather it starts the actual packet processing phase immediately.

Once the packet is available the node starts to process it in a per bit basis. For each bit added to the FIFO buffer the last $\mu bits$ long word is checked for validity, i.e. if the word is a valid Codebook's word. If so, the word is stored in a separate FIFO Valid Message buffer for latter usage.

Afterwords, the node checks if an incoming pulse is received and if that is true, it stores the latest valid message in a dynamic array. A final flag is evaluated before the nodes starts processing the next bit. This flag evaluates to true only if the node has outgoing message and the next word of the message appears in the node's buffer. If both conditions are true, then a 1-bit low energy pulse is sent to inform other nodes of incoming message, while the node gets the next word of the message which needs to be sent.

The loop terminates once a special predefined termination message is sent or a certain amount of time have passed. During the final phase of the algorithm, the HashMap is once more utilized to convert words stored in the dynamic array to the actual message.

$$\frac{PT}{\Delta T} \leq 1 \tag{4.1}$$

Lines 5 to 15 are those mostly affecting inter-node successful communication. For convenience, this time interval is the one previously denoted as *T2*. As long as inequality 4.1 holds, no inconsistencies emerge, since processing time is less than the time required for the next packet to be fully received by the node.

Apparently, for the network to be able to operate efficiently at higher data rates, based on diverse nodes' underlying hardware, inequality 4.1 needs to be ported in system's logic. As long as PT is less than ΔT, no matter how much a node's processing speed deviates from another, the difference is reseted back to zero. Although, if inequality 4.1 does not hold, deviation among nodes will increase after each iteration.

Taking into consideration inequality 4.1, we come to understand that highly optimized code is mandatory in order to achieve very fast Processing Time on every node. To that end, a *user space C* code implementation was drafted which was further enhanced and finally implemented in *Kernel Space C* code for even better results. The kernel space implementation allowed us to execute the logic at maximum priority and access to resources. The downside is the lack of user space libraries, even basic functionalities such as float division, but the benefits are much more important in the studied case and thus it does worth the extra effort required for the Kernel implementation.

The actual code is more or less identical on all three platforms of Table 4.1, but there are though some parts which need to be especially coded for each IoT device. Such a case is the way we achieve GPIO Direct Register Access. Each IoT device has a different architecture and peripheral register address mapping. Specific code snippets for each one of the three IoT platforms, showing how the GPIO registers are accessed is provided in Appendix A.

Listing 4.1 shows the GPIO initialization phase for Raspberry Pi nodes.

```
int init_gpio(void){
  map_base = (volatile unsigned long*)ioremap(GPIO_BASE,
    GPIO_BLOCK_SIZE);

  if(((void*)map_base) == NULL){
    printk( KERN_ERR "ioremap failed.\n Are you root?\n");
    return(-1);
  }

  if(!gpio_is_valid(INPIN) || !gpio_is_valid(OUTPIN) || gpio_request
    (INPIN, "INPIN") || gpio_request(OUTPIN, "OUTPIN")){
    printk(KERN_CRIT "GPIO initialization failed.\n"); // /sys/class
    /gpio/export
    return(-1);
  }

  INP_GPIO(INPIN);
  INP_GPIO(OUTPIN); /* must use INP_GPIO before we can use OUT_GPIO
    */
  OUT_GPIO(OUTPIN);

  return(0);
}
```

Listing 4.1: RPi GPIO Initialization

The initialization could be also achieved using the *SYSFS* export functionality for the GPIO pins without any loss in performance since this procedure takes place only once in the beginning of the experiment. DRA is extremely important during the experiment for pulse sending and sensing between nodes towards speed and logic enhancement.

```
/* Send GPIO Pulse */
void raiseFlag(void){
```

```
  static size_t raiseCounter = 0;

  if(raiseCounter%2==0)
    GPIO_SET = 1<<OUTPIN; /* set output to HIGH */
  else
    GPIO_CLR = 1<<OUTPIN; /* set output to LOW */
  raiseCounter++;
}

/* Sense GPIO Pulse */
bool gpioEventDetected_v2(void){
  static unsigned long previous_state = 0;
  unsigned long cur_state;
  if((cur_state=GET_GPIO(INPIN))!=previous_state){
    previous_state = cur_state;
    return(true);
  }
  return(false);
}
```

Listing 4.2: RPi GPIO Send and sense Pulse

Listing 4.2 provides two functions *raiseFlag() & gpioEventDetected_v2()* which are responsible for sending and sensing GPIO pulses respectively for the Raspberry Pi IoT device. The functions are mostly the same for the other two IoT devices as well.

Another functionality which required optimization is the one presented in Listing 4.3. The function *searchValidMsgInBuffer()* is specifically implemented for 16bits long words and 4bits long Prefix scenarios. Knowing those two details is enough to create a fast Mask-test for words acceptance or rejection.

```
unsigned short searchValidMsgInBuffer(unsigned int buff){
  static bool foundPrefix = false;
  static short bitsAfterPrefix = 0;
  unsigned char buffTail = buff;
  buffTail<<=4; // keep only last 4 bits
  buffTail>>=4; // reset last 4 bits to their initial positions

  if(PREFIX==buffTail){
    foundPrefix=true;
    bitsAfterPrefix=0;
    return(0);
  }
  else if(foundPrefix){
    bitsAfterPrefix++;
    if(bitsAfterPrefix==12){
      foundPrefix=false;
      return((unsigned short)buff);
    }
  }
  return(0);
}
```

Listing 4.3: Valid Word Inspection

A few more implementation elements worth mentioning are the custom division, the data collection and the hashing functions.

A long long int division function couldn't be avoided since the functionality is necessary for many important calculations during the experiments.

Apart from logging various vectors of useful data during the experiments a special *rolling average function* is coded to dynamically calculate the average value of certain measurements and the total number of samples. The benefit of this function is the minimum requirements in memory space, since the only data required to calculate the rolling average are the previous value and the number of samples.

Regarding the hash function utilized for the Hashmap creation, it is a simple modulo operation between a hashed key (word) and the total number of words in the Codebook.

4.2 Evaluation

In this Section we validate at a proof-of-concept level the functionality of the outlined BitSurfing process, on real IoT devices.

To verify that the system qualifies the criteria of section 4.1, time intervals and message delivery error rates are measured on various platform combinations, described previously. There are no assumptions considered for the experiments. The model is evaluated in a controlled environment.

4.2.1 Experiments and Results

In this case study, the goal is to transfer a 100 byte long file between nodes. Two distinct cases were considered. One of file transfer from BeagleBone Black (BBB) to RaspberryPi (RPi) and another file transfer from BananaPi (BPi) to a second RPi device, as shown in Figs. 4.1 and 4.2. The node connectivity and the network wide deployment are those described in the previous section.

The measured quantities are the Message Error Rate (MER) and data processing Time ($T2$) as well as the packet interarrival time (ΔT) for various bit rates. The later of the measurements is conducted in order to be able to verify that its value is close to − if not exactly the same − the one which is set on the bitStream source side.

Since a lot of experiments take place, it is important to ensure all system-wide parameters remain unchanged among tests. To that end, a live stream mp3 audio file is selected to be broadcasted over the network during all tests. Another approach could be to use a random packet generator as bitStream source, but there would be no assurance of equal network parameters among tests.

The live stream Packet interarrival time, ΔT, was measured in the bitStream source using Wireshark network analyzer. The average value of ΔT is $41.8msec$. This value applies network wide and has to be, on average, the same on all network nodes.

Apart from Wireshark on the bitStream source side, ΔT is also verified on bitStream nodes side. To achieve this, nodes were configured to operate as UDP clients reading packets from a specific port to which the source was sending data. After sufficient number of samples were collected, it was possible to create a histogram to verify Wireshark results on client side also. Histogram of Fig. 4.6 shows that packet interarrival time does indeed fluctuate around 41.8$msec$.

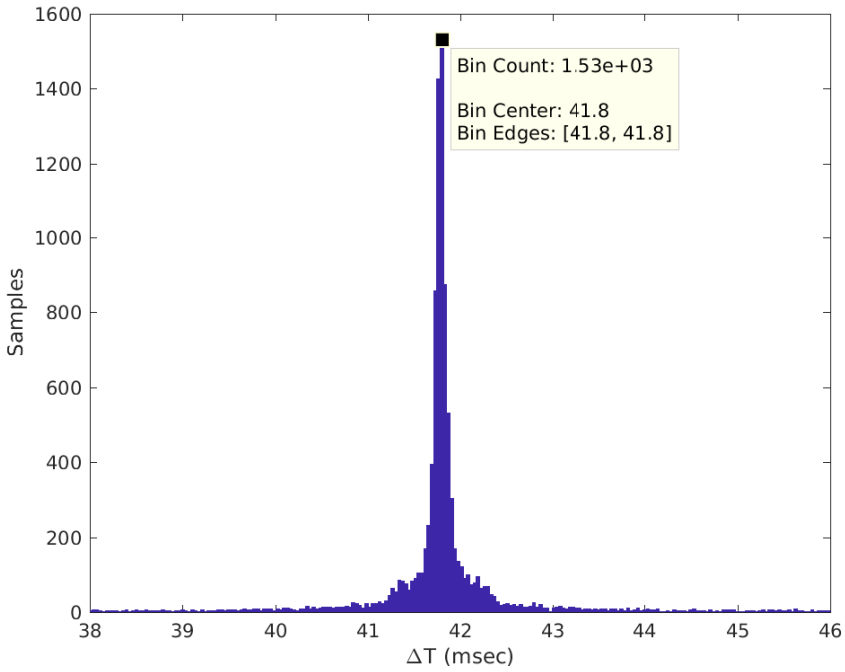

Figure 4.6: ΔT Interarrival Time as measured on client side

In addition, packet interarrival time is measured for several different data rates, during actual testing period. The observation of Fig. 4.7 indicates that data rate is irrelevant to ΔT, as expected.

Figure 4.8 illustrates the Message Delivery Error Rate for a range of different data rates, for both tested platform pairs. As someone may observe, the examined communication logic can be successfully employed in applications with limited energy supply capabilities and low bitStream generator supported bit rates. In particular, there is no error for bit rates below 191bps and thus the whole 100byte file is successfully transferred to the destination intact. Although, as data rate increases the error rate does also increase.

Considering a maximum threshold of 50% in file transfer error rate, it is clear that data rate should be limited up to 1.2kbps. Above that data rate threshold the deviation in processing speed between the tested platforms starts to play a major role. A temporary workaround could be the usage of similar characteristics

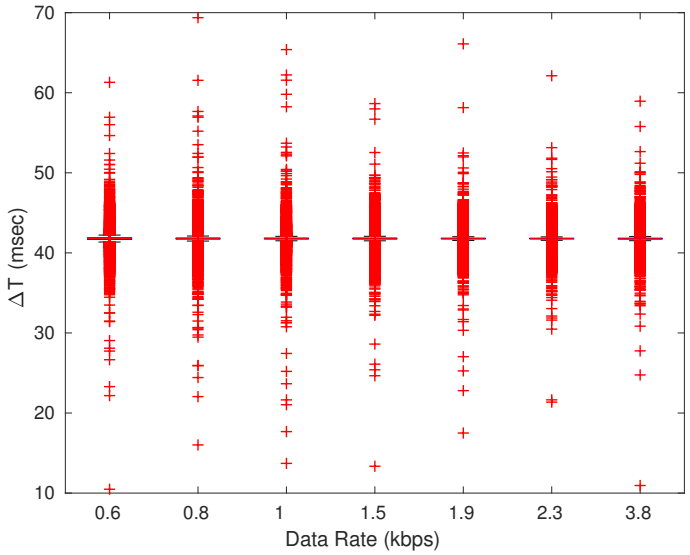

Figure 4.7: ΔT Interarrival Time for various data rates

nodes in the network, i.e. clock difference between nodes of 100MHz instead of 300MHz in the case of BBB (1GHz) and RPi (0.7GHz) and 500MHz in the case of BPi (1.2GHz) and RPi. This is also visible in Fig. 4.8, in which Error Rate increases slightly slower for BBB-RPi pair compared to BPi-RPi.

Furthermore, the fact that both studied platform pairs provide very similar results − mean error rate difference of 0.73% −, support the idea that BitSurfing can operate regardless the underlying hardware of each platform-node.

The T2 Processing time is a value also measured during tests and its illustrated average value for different data rates provides a logical explanation for the Error rate threshold. As shown in Fig. 4.9 the average T2 processing time increases almost linearly, in most parts, to the data rate. For values up to 1kbps − which happens to be very close to the data rate for which 50% threshold is met in Fig. 4.8 − T2 barely increases, but once data rates increases above this threshold T2 increases almost linearly. That is an interesting observation which associates Error rate and T2 processing time, also confirming inequality 4.1.

4.2.2 Applied Improvements and Results

As pointed out in subsection 4.2.1, the proposed communication logic can be indeed used as a low energy consumption solution, without any need to take into account nodes' underlying hardware specifications.

The simplicity of the proposed logic, which is almost hardware agnostic, is an advantage which makes it an easily ported communication solution for IoT devices. Besides the demonstrated proof-of-concept regarding the functionality of

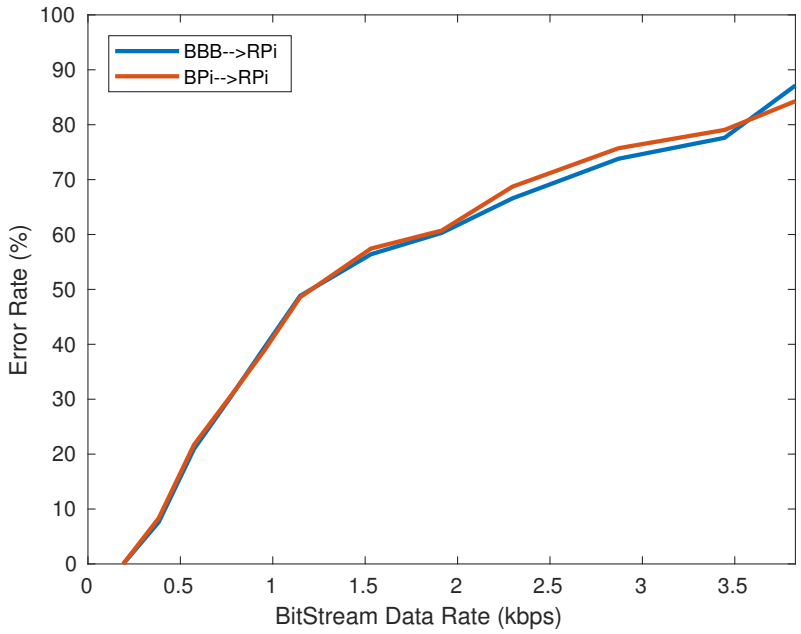

Figure 4.8: File Transfer Error Rate (%)

BitSurfing communication logic on real hardware, we also propose an extension which can decrease the experimental system's Error Rate at higher data rates.

To that end, every node connected to the network has to complete a training phase before starting its actual communication. During the training phase, which is decided to last for 100 packet arrival cycles, the node adjusts a delay parameter according to its needs. More specifically, each node measures the time required for packets to arrive and after the training phase completion it extracts the average interarrival time ΔT.

In the deployment phase, nodes keep on measuring timings, but they also introduce delays in their communication process. This way nodes are able to dynamically adopt changing application or network (change in topology) characteristics. Nodes intra-measurements include : *T1*, *T2* and $T2_{byte}$ times, where $T2_{byte}$ is the time required to process each byte of a packet. As a result, the following equation holds :

$$T2 = \#(Bytes\_In\_the\_Packet) * T2_{byte} \tag{4.2}$$

According to these measurements, a delay is added, forcing the node to wait a certain amount of time after each byte is processed, based on $T2_{byte}$, always with respect to the inequality 4.1. So, ideally an order of μsec delay is added to $T2_{byte}$ leading to modification of equation 4.2 to the following :

$$T2 = \#(Bytes\_In\_the\_Packet) * (T2_{byte} + \mu delay(byte)), \tag{4.3}$$

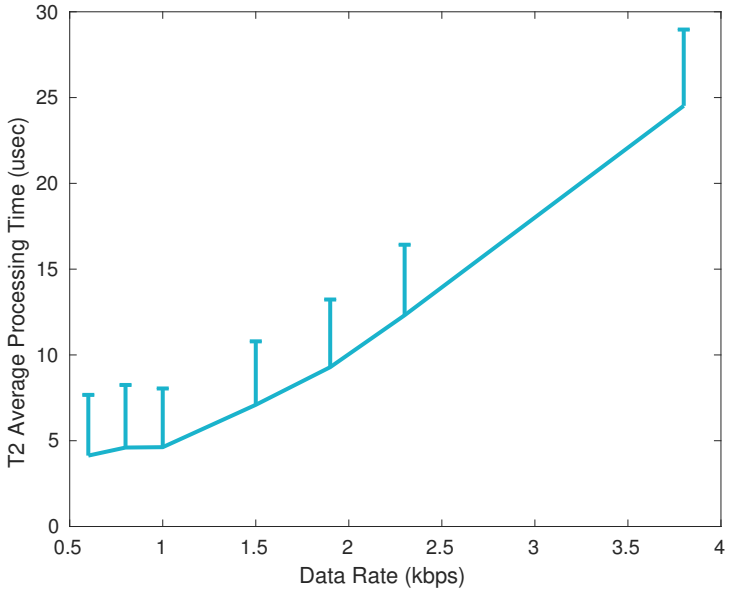

Figure 4.9: T2 Average Processing Time for various data rates

$\mu delay(byte)$ is not a constant value, rather a delay adjusted depending $T2_{byte}$ and ΔT measured during the training phase. Although this technique may seem to slow down the whole network, it does improve the overall results even for much higher data rates.

Figure 4.10 shows Message Delivery Error Rate for various data rates. It is clear that there is a huge improvement compared to results in Fig. 4.8. The 50% Error Rate threshold can be achieved for data rates below 55kbps. The improved model raises the threshold more than 45 times higher regardless the underlying hardware used.

Furthermore, similar to the observations made in Fig. 4.9 regarding the relation of file transfer error rate and T2 processing time, Fig. 4.11 strengthens the idea that File Transfer Error Rate depends highly on T2 processing time. A cubic distribution is provided along with the resulted line. As shown the average T2 timings follow a cubic distribution. It is also interesting the fact that File Transfer Error Rate increases above the 50% threshold around 55kbps, which happens to be the threshold after which values of distribution in Fig. 4.11 increase once again.

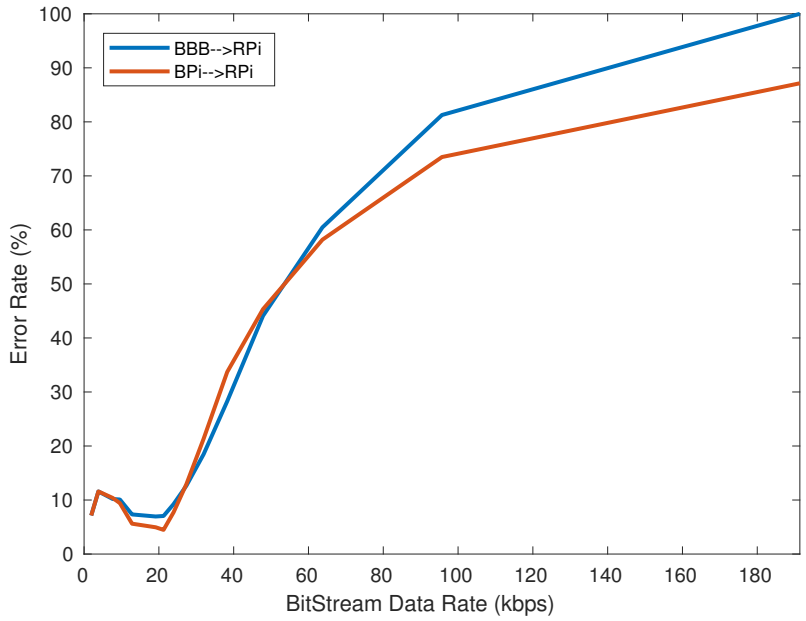

Figure 4.10: File Transfer Error Rate (%) using node delay parameter

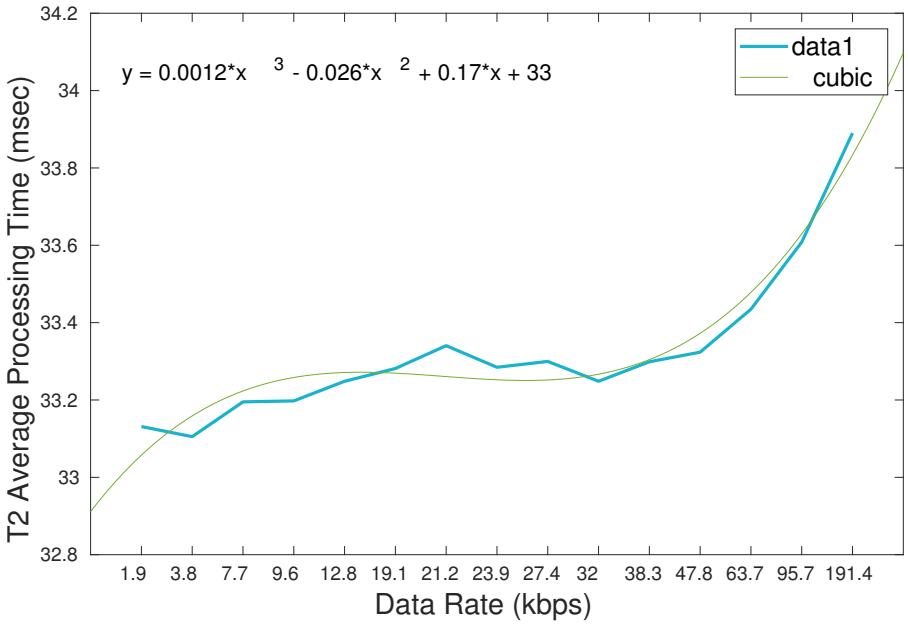

Figure 4.11: T2 Average Processing Time for various data rates using node delay parameter

Chapter 5

Simulation of BitSurfing Logic

5.1 Simulation Setup

By fulfilling the real world proof of concept experimentation part successfully, a question arising regarding the studied communication logic is whether the logic is applicable on a larger network on the one hand and how does the logic cope with heavy network congestion levels on the other hand. The objective of this chapter is to examine the scalability and survivability ability of BitSurfing communication logic.

For the needs of this part of the study, a BitSurfing Simulator is created. The Simulator is implemented in Python using the threading module, in order to port the idea of independent nodes in the simulated environment. Each node runs on a separate thread, while all simulated nodes retrieve new data from a queue fed by a BitStream Source running on the main (parent) thread.

Initially we did allow all threads to be totally free of sync considering that real world adapters are also totally free of sync (as shown in chapter 4), but in the case of data processing under the same unit (multi-core processor) the idea of free sync doesn't work very well.

Although during the experiments the logic abstraction seemed to work properly, under the hood there were serious threading issues. To be more specific observation showed that a number of threads (BitSurfing adapters) were processing data arriving to other threads-adapters much later, causing the whole simulation to fail. Once more there was a need to port the rule of equation 4.1 in the simulation logic. In order to do that a $\mu sleep()$ delay was utilized in the main thread, in order to ensure that all threads-adapters will be processing the same data more or less.

```
1 import resources as res
2
3 #%% Init Simulation
4 sim=res.Simulation(networkDimensions=[4,5], prefix="1110")
5 sim.setupNetworkGraph()
6 sim.visualize_network()
7
```

```
8  sim.associate_Graph_to_Nodes()
9  sim.startSimulation(
10         bits_from_codebook=False,
11         num_of_msgs_to_send=50,
12         total_Bits_to_send=1000000
13         )
```

Listing 5.1: High Level Simulation View

Listing 5.1 provides an overview of the highest level of the simulation environment.

As shown in the code, the user can tune directly variables, such as the network dimensionality, the Codebook Prefix, whether a specific file will be used for streaming data or not, the total number of messages to be randomly created and send over the network, as well as the duration of the simulation in terms of events.

In the first line of the code, the important custom module containing the Node, Packet, BitSurfing Source and other classes of the simulation logic is imported.

Following, in lines 4, 5 and 8 the simulation is initialized, the network graph is created and the several network and node level characteristics are configured. Line 6 is purely optional since it does only provide a graphical visualization of the created network. An example network visualization of a 4x5 topology is provided in Figure 5.1. Graph creation and visualization processes were implemented based on *networkx* Python package.

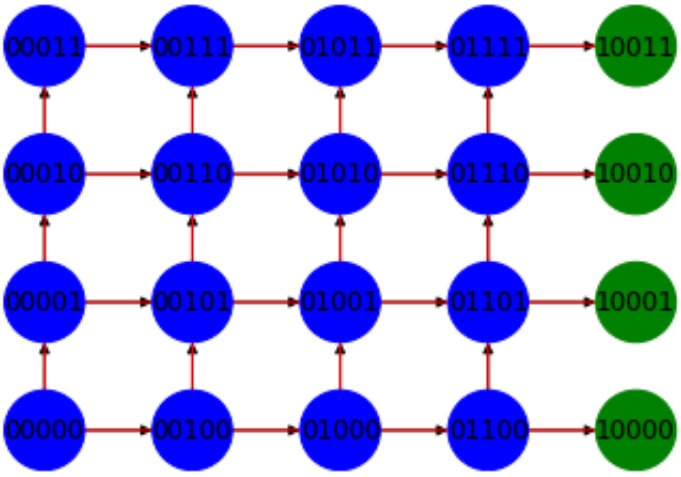

Figure 5.1: Simulated Network Visualization for a 4x5 topology example case.

As someone can see identification names are provided to all network nodes. Gateway nodes are also explicitly defined during setup process, along with the connectivity among nodes which is also predefined. The core logic is to reconfigure the network in some extend in order to reduce any node level overhead which would be required for neighborhood exploration.

So, the network is created from a bottom up and left to right logic, and thus each node will always randomly choose a right or upper neighbor to forward a message in order to prevent infinite messaging loops, while the gateways will always be positioned at the right end of the network.

The graph connectivity logic can be easily tuned to fit any other specific network requirements.

In line 9 the simulation is started by providing specific simulation parameters, such as: a codebook, the number of messages to be created for transmission over the network as well as the total number of events the simulation will last, at most. This last parameter is the one determining the simulator as an event based one, regardless its multi-threading logic.

Going a level deeper, inside *resources* module someone can see the Bit Stream Source, BitSurfing Adapter and Packet classes. A Packet class was created to help better keep track of ongoing events, even though there is no packet actually transferred between nodes. As shown in Listing 5.2 the packet has many properties most of which are utilized for the simulation monitoring process. Those properties help the user to examine the path of nodes each packet followed during the simulation, as well as the number of events required for the packet to reach its destination and the elapsed time in seconds for this event to happen.

```
1  class Packet(object):
2      def __init__(self,sid,payload):
3          self.senderId=sid
4          self.path = list()
5          self.payload=payload
6          self.marked_to_send = False
7          self.creationTime=time.time()
8          self.creationBit = 0
9          self.arrivalBit = int
10         self.BitDifference = int
11         self.arrivalTime=float
12         self.TimeDifference = float
13         self.arrived_to_gateway=False
14     def log_packet(self):
15         logging.info("########### New Packet ###########")
16         logging.info(f"
17             Packet Path={self.path}
18             \nBitDifference={self.BitDifference}
19             \nTimeDifference={self.TimeDifference}
20             ")
```

Listing 5.2: High Level Simulation View

5.2 Simulation Results

Having explained the abstract simulator logic we can continue to the objective of this part of the study, i.e. conducting the actual simulations and retrieving the results.

For the needs of this study 2 different sets of simulations were implemented. The *first* set consisted of 10 distinct simulations in which the total number of messages created and sent over the network is constant and the network size varied. The purpose of this set of simulations is to understand how BitSurfing logic cope with multiple hopping messages inside a network.

The simulator is configured to use a Codebook containing only valid words, the Prefix is set to the value "1110", the constant number of messages sent over the network is set to 10 and the total number of events which will at most take place before each simulation comes to an end is 1 million. The network was also configured to have a number of nodes from 2 up to 32, different in each simulation.

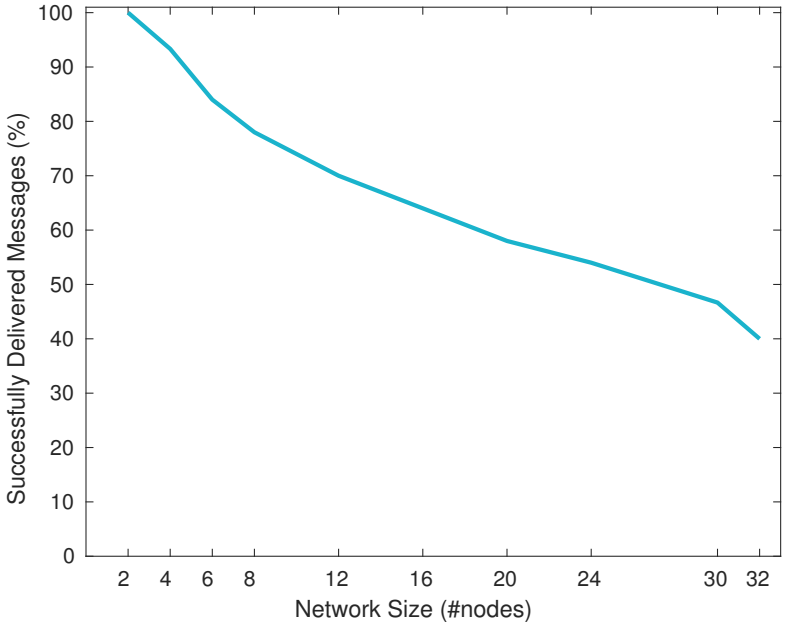

Figure 5.2: Successful Message Delivery Rate for Variant Network Sizes

Figure 5.2 shows the ability of the network running BitSurfing communication logic to successfully transfer messages network wide. As someone may observe the message delivery rate is almost perfect for smaller network sizes, while as the network size grows the rate decreases. Those results suggest that BitSurfing can be employed in small as well as bigger communication networks in some extend.

Another insightful representation is provided in Figure 5.3. This Figure shows the average number of kbits required for a message to reach a gateway node. The values are collected for various network sizes. Data suggest that a message requires on average more bits to be delivered as the network size increases. This observation seems logical since a random packet needs to follow a longer path in the case of a larger network and thus hop between more nodes.

For the *second* set of simulations the goal is to identify the ability of BitSurfing

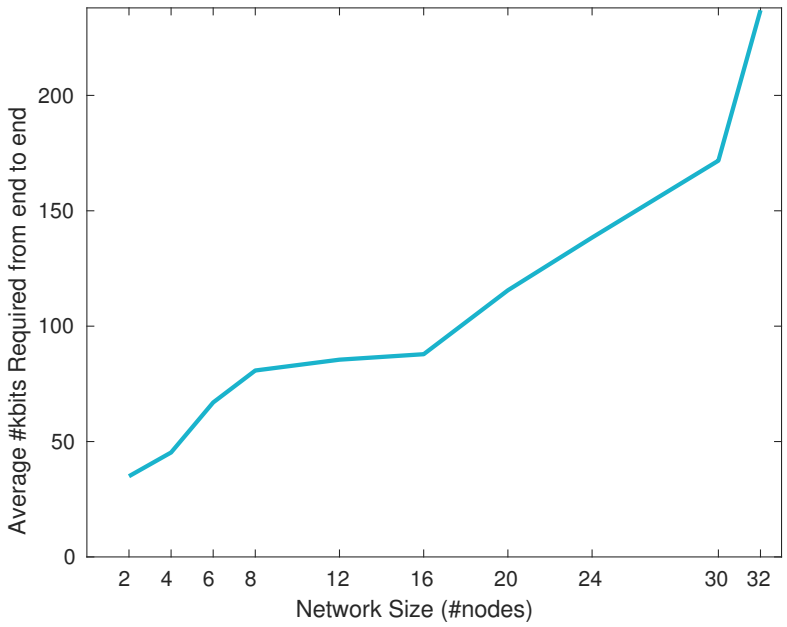

Figure 5.3: Average number of kbits (events) required for each message transmission for various network sizes

logic to support multiple messages over the same network. To that end, the network size (number of network nodes) remained constant while the total number of messages issued for creation and transmission over the network varied.

In this case study the simulator is configured to use a Codebook containing only valid words all starting with the prefix "1110", the network has a size of 32 nodes in a 4x8 topology − the largest possible network for a Prefix of 4bits length −, while the total number of events before each simulation completion is set to 1 million. The total number of messages which are created changes in every simulation and has a value in the interval [1,100] using a step of 10 messages each time.

The fulfilment of all the simulations in the set supplies enough data to create the plot of Figure 5.4. This Figure provides a representation of the message delivery rate with respect to the total number of messages sent over the network. Results do not provide a clear view of whether BitSurfing is capable of handling high network congestion. Although for low congestion the logic performs well, the performance reduces for higher congestion level but somewhat stabilizes for even higher congestion.

An important fact worth mentioning is the system upon which all simulations took place. The system is a Lenovo ThinkPad T420 equipped with a 2-core Intel 2520m CPU and 8GB of RAM.

The results shown in Figure 5.4 might seem abnormal, but we might be able to

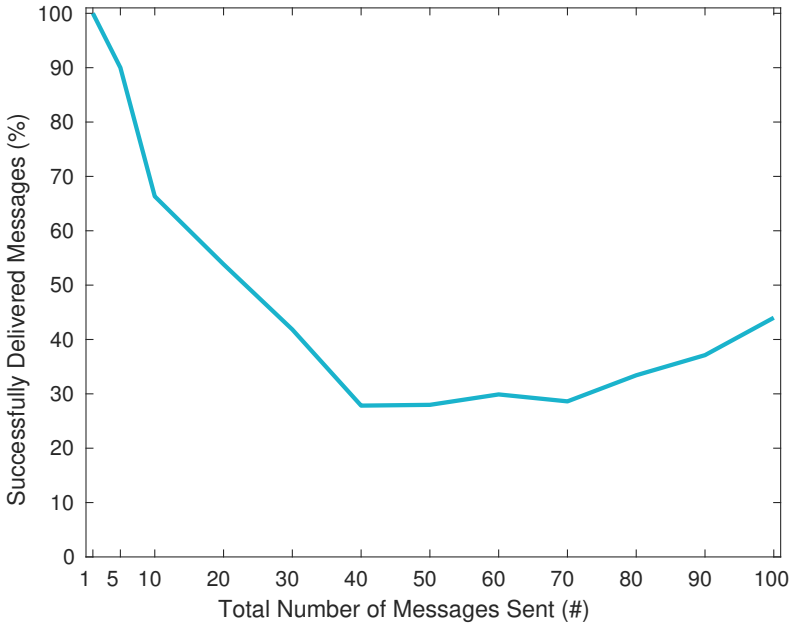

Figure 5.4: Successful Message Delivery Rate for Variant Network Congestion Levels

explain why this is happening if we consider the system on top of which all simulation are taking place. Running the simulations on top of a 2-core system doesn't provide true independence among nodes. Those pseudo-independent threads increase in number as the network size increases, causing the logic to deviate from its pseudo-parallel execution.

Nevertheless, specific simulation setups and associated results were presented in this chapter, suggesting that BitSurfing can be a feasible communication solution worth further investigation, ideally on top of a system equipped with newer generation hardware capable of providing more computing resources.

Chapter 6

Conclusions, Applications and Research Directions

6.1 Conclusion

This work provides a proof-of-concept level implementation of BitSurfing, which is a novel communication pattern for IoT devices. According to it, nodes wait for the opportunistic production of their intended messages by other sources, reducing inter-IoT node communication to the exchange of simple short pulses. The present study compliments past theoretical studies and proves that BitSurfing is feasible in practice, using common Raspberry Pi hardware. A simulator was created and utilized to examine the logic scalability and survivability. Results collected from simulations do also suggest that BitSurfing is a viable communication alternative solution for low energy applications. A practical operational model was introduced, and future directions that can impact the IoT research were outlined.

6.2 BitSurfing Applications

Although BitSurfing is a new idea which surely requires further investigation before commercial field deployment, there is already a domain of possible applications which could benefit from this technology.

- **Band-Jamming Communications** : By listening to band jamming the first idea popping to someones mind might be the deliberate or not signal alteration or loss in certain frequency bands.

  Although that is indeed a problem buzzing the communications research community, jamming or more appropriately put, the result which is the created noise, can be utilized as a BitStream Source in the case BitSurfing Adapters are installed in the area. This way energy free BitSurfing deployment is possible. Tests and measurements need to take place before the

actual deployment in order to ensure that all adapters in the network receive the same signal.

Such an application could be IoT enabled shoes, which monitor pressure. Tiny BitSurfing adapters, RF powered, collect data and using the analyzed logic send the data to a gateway node (which might have a battery) for storage or further processing.

- **Controlled Environment Communications** : The idea of controlled environment communication has two main differences with the Band-Jamming Idea. The environment or more specifically the frequency band has to be silent (noise free) and the BitStream Source is deployed and controlled based on the application requirements.

  In that sense, if the deployment uses a Codebook with a number of words (bit sequences) repeated, the BitStream Source can be optimized to only produce those words instead of random bit sequences. This way, the expected elapsed time until a valid word appears further reduces.

  Forest Fire Detection

  Such an environment could be reproduced in forests, where RF signals are much more limited or almost absent. Low cost batteryless BitSurfing adapters can be placed in several spots of the forest and be programmed to monitor the environmental temperature. An isotropic antenna can be placed somewhere in the centre of the forest, which will operate as the Controlled BitStream Source. BitSurfing adapters will also be equipped with an ultrasonic transceiver for intra-network communications.

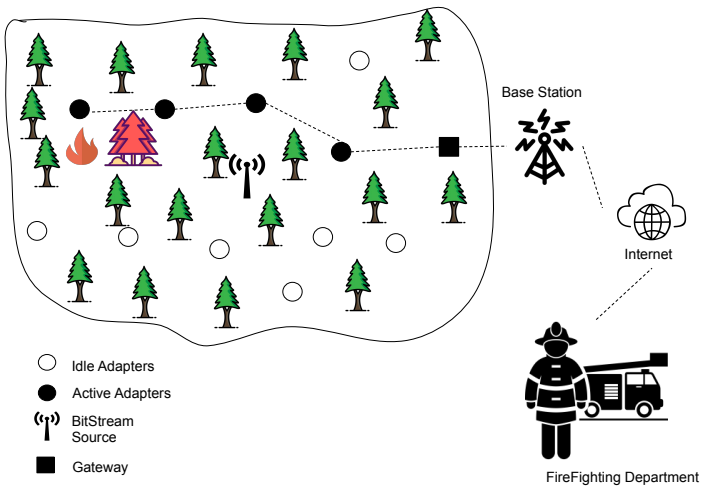

Figure 6.1: The Forest Fire Detection Model

The decision to place the BitStream Source in the center of the forest is based on the logic of minimizing propagation delay network wide. The Source can

be programmed to transmit specific bit sequences, in order to reduce the expected time until a valid word appears. Another characteristic to consider is the operating frequency, which shall not be too high (not above 11GHz) since there will not be necessarily any line-of-sight between the BitStream Source and the adapters.

Figure 6.1 shows the Forest Fire detection application logic. The BitStream Source is placed in the center of the forest and feeds all surrounding adapters both with RF energy and data.

The BitSurfing adapters listen to the incoming stream and if one of them senses a temperature above the acceptable limits it waits for the specific emergency word to appear in buffer and sends an ultrasonic pulse to the closest neighbor. The adapters keep on transmitting the information from node to node until the gateway is reached.

Once the gateway is reached, another communication protocol is utilized to alert the fire Department of the ongoing situation.

Agricultural Monitoring

Another potential application of the BitSurfing communication logic is in the Agricultural domain. Similarly to the forest fire detection system, farmers are able to utilize a controlled BitSurfing deployment to monitor their farming field.

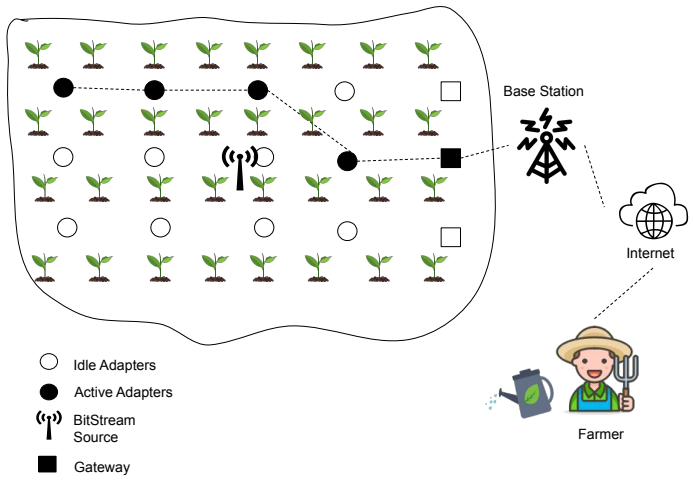

Figure 6.2: Agricultural Monitoring Model

BitSurfing adapters can be installed in different point along the field and be programmed to continuously listen on a BitStream Source which will be located in the middle of the field. Bit sequences of 2 bits will carry the actual information (e.g. low humidity levels, low pH levels).

Whenever an adapter has data to send to the gateway, it raises a flag (led blink or IR light) to inform neighboring nodes of the incoming data. The process repeats from node to node until the gateway is reached.

As someone may observe in both BitSurfing applications the deployment is more or less static, in the sense that the engineer knows how the nodes will be distributed in the forest or farming field respectively and any potential addition will require new adjustments to the network.

6.3 Research Directions

The successful proof-of-concept implementation analyzed in the present work opens various research directions regarding BitSurfing communication logic and its capabilities.

Energy Efficiency. BitSurfing design indicates a low-energy consumption scheme. Monitoring real hardware, in order to create an energy profile of the proposed logic, can be highly valuable. Furthermore, a comparison of the proposed communication logic against other established protocols, can provide sufficient information on how the logic performs in terms of energy consumption needs.

Security. The utilization of a Codebook along with a corresponding HashWords dictionary for the conversion and the actual transmission of messages between BitSurfing nodes can be thought as a security measurement. According to Kerckhoffs's principle: "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge". Thinking of the Codebook and the HashWords dictionary as the system's key, it is safe to state that the logic uses by default a symmetric key cryptography. Security of IoT devices is a huge research topic and the exploration of possible and practical security schemes which may be applied on BitSurfing nodes could be of great interest.

Hardware Optimization. For the proof-of-concept study we used generic IoT (RasspberryPi) devices. Now that both simulations [21] and real hardware implementation supports BitSurfing functionality, a hardware specific optimization can take place. It is worth mentioning that for the experiments of this study the logic was implemented as a C Loadable Kernel Module − almost optimal implementation given the hardware −. Porting the logic to an FPGA can be an action towards optimal hardware implementation (a step before ASIC).

Smart Logic. The simplicity of the proposed logic is a huge advantage over other protocols. It can, however, be enhanced by porting in node level artificial intelligence logic, while maintaining its effectively simple logic. As shown in section 4.2, completing a training stage before the actual communication can be beneficial, resulting in higher supported data rates. A *Reinforcement Learning* model can be derived, in order to provide nodes the ability to learn through observation and dynamically adopt to changing network characteristics (E.g. mobility, SINR, data rate).

Network congestion. As shown in [21] the packet delivery rate is high even

for heavy network congestion. Conducting a survivability analysis on a relatively large network of real BitSurfing enabled hardware devices can verify simulation results. It would also be interesting to test the logic in a completely wireless environment. This would require all devices, BitSurfing adapters and the BitStream Source to have a wireless communication interface. A combination of Raspberry Pi Zero W devices and a handful of IR modules could be enough for the task, along with a laptop or another device to operate as the BitStream Source.

A cluster Based Simulator The simulator presented in chapter 5 is based on multi-threading logic to achieve independence among nodes. A very good alternative would be to port the adapter's logic as a containerized application in a Kubernetes pod and populate-replicate the number of truly independent BitSurfing nodes in a Kubernetes cluster.

Appendix A

GPIO Direct Register Access

```
/* Valid only for Raspberry Pi with BCM2835 ARM Microprocessor */
#define GPIO_BASE (0x20000000 + 0x200000)
#define GPIO_BLOCK_SIZE 4096
#define INP_GPIO(g) *(map_base+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(map_base+((g)/10)) |=  (1<<(((g)%10)*3))
#define GET_GPIO(g) (*(map_base+13)&(1<<g)) // 0 if LOW, (1<<g) if
    HIGH
#define GPIO_SET *(map_base+7)  // sets bits which are 1 ignores
    bits which are 0
#define GPIO_CLR *(map_base+10) // clears bits which are 1 ignores
    bits which are 0

#define INPIN 17 // board pin 11
#define OUTPIN 27 // board pin 13
/* ground is board pin 6 */
/* ------------------------------------------------- */
```

Listing A.1: RPi GPIO Register Access

```
/* Valid only for BeagleBoneBlack with AM335x ARM Cortex-A8
    Microprocessor */
#define GPIO2_BASE_ADDR 0x481AC000
#define GPIO2BANK_SIZEINBYTES 4096
#define OE_ADDR 0x134
#define GPIO_DATAOUT 0x13C
#define GPIO_DATAIN 0x138
#define OUTPIN 2 // board pin P8.7  66
#define OUTPINSYSFS 66 // board pin P8.7  66
#define INPIN 5 // board pin P8.9   69
#define INPINSYSFS 69 // board pin P8.9   69
/* ground is board pin P8.1 */
#define GET_GPIO(g) (map_base[GPIO_DATAIN/4]&(1<<g)) // 0 if LOW,
    (1<<g) if HIGH
/* -------------------------------------------------*/
```

Listing A.2: BBB GPIO Register Access

```
/* Valid only for BananaPi - M3 AND ONLY for Port C (PC) registers
    with Allwinner A83T ARM Microprocessor */
#define CCU_MODULE_ADDRESS 0x01c20000 /* need this for mmap (since
    we have to use a value multiple of page size, i.e. 4096) */
#define GPIO_BLOCK_SIZE 4096
#define GPIO_BASE_BP_RELATIVE_ADDR 0x800
#define PC_CFG0_REG_RELATIVE_ADDR 0x48 /* Port C GPIO address! The
    code below is valid only for this port (PC) of BPi - M3 */
#define PC_DATA_REG_RELATIVE_ADDR 0x10
#define OUTPIN 4 /* PC4 or board pin 11 or gpio 68 or bcm 17 */
#define INPIN 7 /* PC7 or board pin 13 or gpio 71 or bcm 27 */
#define TURN_OFF_PC_PUD_0_15(pin) map_base[0x1c/4] &= (0xFFFFFFFF ^
    (1 << 2*pin)) /* 0x1c = 0x64 - 0x48 which is the relative
    distance from map_base address -# default value is 0x5140 = 0
    b101000101000000 */
#define SET_PC_INP_GPIO(pin) *(map_base) &= (0xFFFFFFFF ^ (7 << 4*
    pin)) /* default value 0x77777777 = 0b0 111 011101110 111
    0111011101110111 */
#define SET_PC_OUT_GPIO(pin) *(map_base) |= (1 << 4*pin) /* default
    value 0x77777777 = 0b0 111 011101110 111 0111011101110111 */
#define READ_PC_PIN_VALUE(pin) (map_base[PC_DATA_REG_RELATIVE_ADDR
    /4]&(1<<pin)) /* 0x10 = 0x58 - 0x48 -# returns 0 if LOW, (1<<pin)
     if HIGH */

#define INPINSYSFS 71 /* board pin 13 --> 71 */
#define OUTPINSYSFS 68 /* board pin 11 --> 68 */
/* ground is board pin 6 */
/* ------------------------------------------- */
```

Listing A.3: BPi GPIO Register Access

Bibliography

[1] G. ΧΑΛΚΙΟΠΟΥΛΟΣ, "*διαλειτουργικά πρωτόκολλα επικοινωνίας στο* IoT," 2019.

[2] F. Samie, L. Bauer, and J. Henkel, "IoT technologies for embedded computing: A survey," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, p. 8, ACM, 2016.

[3] J. Finnegan and S. Brown, "A comparative survey of lpwa networking," *arXiv preprint arXiv:1802.04222*, 2018.

[4] X. Shi, X. An, Q. Zhao, H. Liu, L. Xia, X. Sun, and Y. Guo, "State-of-the-art internet of things in protected agriculture," *Sensors*, vol. 19, no. 8, p. 1833, 2019.

[5] J. A. Khan, H. K. Qureshi, and A. Iqbal, "Energy management in wireless sensor networks: A survey," *Computers & Electrical Engineering*, vol. 41, pp. 159–176, 2015.

[6] X. Lu, P. Wang, D. Niyato, D. I. Kim, and Z. Han, "Wireless networks with rf energy harvesting: A contemporary survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 757–789, 2014.

[7] S. D. Barman, A. W. Reza, N. Kumar, M. E. Karim, and A. B. Munir, "Wireless powering by magnetic resonant coupling: Recent trends in wireless power transfer system and its applications," *Renewable and Sustainable energy reviews*, vol. 51, pp. 1525–1552, 2015.

[8] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, no. 2011, pp. 1–11, 2011.

[9] N. Heuveldop *et al.*, "Ericsson mobility report," *Ericsson, Stockholm*, 2017.

[10] "Ericsson mobility report." `https://www.ericsson.com/assets/local/mobility-report/documents/2019/ericsson-mobility-report-june-2019.pdf`, June 2019. Accessed: 2019-6-25.

[11] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, "Internet of nano-things, things and everything: Future growth trends," *Future Internet*, vol. 10, no. 8, 2018.

[12] P. Scully, "The top 10 IoT segments in 2018 – based on 1,600 real IoT projects." `https://iot-analytics.com/top-10-iot-segments-2018-real-iot-projects/`. Accessed: 2019-07-18.

[13] "IoT asics." `http://www.open-silicon.com/solutions/iot-asics/`. Accessed: 2019-07-24.

[14] "IoT connectivity market." `https://iot-analytics.com/iot-segments/iot-connectivity/`. Accessed: 2019-08-01.

[15] S. Jankowski, J. Covello, H. Bellini, J. Ritchie, and D. Costa, "The internet of things: Making sense of the next mega-trend," *Goldman Sachs*, 2014.

[16] S. R. Nimbargi, S. Hadawale, and G. Ghodke, "Tsunami alert & detection system using IoT: A survey," in *Big Data, IoT and Data Science, 2017 International Conference on*, pp. 182–184, IEEE, 2017.

[17] C. Kamienski, J.-P. Soininen, M. Taumberger, S. Fernandes, A. Toscano, T. S. Cinotti, R. F. Maia, and A. T. Neto, "Swamp: an IoT-based smart water management platform for precision irrigation in agriculture," in *2018 Global Internet of Things Summit (GIoTS)*, pp. 1–6, IEEE, 2018.

[18] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities," *IEEE Wireless Communications*, vol. 20, no. 6, pp. 91–98, 2013.

[19] H. Hejazi, H. Rajab, T. Cinkler, and L. Lengyel, "Survey of platforms for massive IoT," in *Future IoT Technologies (Future IoT), 2018 IEEE International Conference on*, pp. 1–8, IEEE, 2018.

[20] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad hoc networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[21] A. Tsioliaridou, C. Liaskos, and S. Ioannidis, "Bitsurfing: Wireless communications with outsourced symbol generation," in *IEEE CAMAD*, 2018.

[22] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[23] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad hoc networks*, vol. 3, no. 3, pp. 325–349, 2005.

[24] M. Bhalla, N. Pandey, and B. Kumar, "Security protocols for wireless sensor networks," in *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pp. 1005–1009, IEEE, 2015.

[25] "Baseline security recommendations for IoT." `https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot`, Nov. 2017. Accessed: 2019-7-4.

[26] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[27] E. Haselsteiner and K. Breitfuß, "Security in near field communication (nfc)," in *Workshop on RFID security*, pp. 12–14, sn, 2006.

[28] M. Miraz, M. Ali, P. Excell, and R. Picking, "Internet of nano-things, things and everything: future growth trends," *Future Internet*, vol. 10, no. 8, p. 68, 2018.

[29] A. M. Ortiz, D. Hussein, S. Park, S. N. Han, and N. Crespi, "The cluster between internet of things and social networks: Review and research challenges," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 206–215, 2014.

[30] ISO Central Secretary, "Internet of things (IoT) – reference architecture," Standard ISO/IEC 30141:2018, International Organization for Standardization, Geneva, CH, 2018.

[31] M. LAVANYA, "Survey of internet of things (IoT) & its challenges," in *Journal of analysis and computation, 2019 NCRCISCS Conference on*, JAC, 2019.

[32] Y. Zhu and R. Sivakumar, "Challenges: communication through silence in wireless sensor networks," in *Proceedings of the 11th annual international conference on Mobile computing and networking*, pp. 140–147, ACM, 2005.

[33] Y. P. Chen, D. Wang, and J. Zhang, "Variable-base tacit communication: a new energy efficient communication scheme for sensor networks," in *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, p. 27, ACM, 2006.

[34] K. Sinha, "A new energy efficient mac protocol based on redundant radix for wireless networks," *arXiv preprint arXiv:1606.04935*, 2016.

[35] S. D'Oro, L. Galluccio, G. Morabito, and S. Palazzo, "A timing channel-based mac protocol for energy-efficient nanonetworks," *Nano Communication Networks*, vol. 6, no. 2, pp. 39–50, 2015.

[36] S. Wu, H. Wang, and C.-H. Youn, "Visible light communications for 5g wireless networking systems: from fixed to mobile communications," *Ieee Network*, vol. 28, no. 6, pp. 41–45, 2014.

[37] Z. Wang, D. Tsonev, S. Videv, and H. Haas, "On the design of a solar-panel receiver for optical wireless communications with simultaneous energy harvesting," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 8, pp. 1612–1623, 2015.

[38] K. Fan, H. Xu, L. Gao, H. Li, and Y. Yang, "Efficient and privacy preserving access control scheme for fog-enabled IoT," *Future Generation Computer Systems*, vol. 99, pp. 134–142, 2019.

[39] J. Sun, Y. Su, J. Qin, J. Hu, and J. Ma, "Outsourced decentralized multi-authority attribute based signature and its application in IoT," *IEEE Transactions on Cloud Computing*, 2019.

[40] M. Khodjaeva, M. Obaidat, and D. Salane, "Mitigating threats and vulnerabilities of rfid in IoT through outsourcing computations for public key cryptography," in *Security, Privacy and Trust in the IoT Environment*, pp. 39–60, Springer, 2019.

[41] X. Zhang, C. Liu, S. Poslad, and K. K. Chai, "A provable semi-outsourcing privacy preserving scheme for data transmission from IoT devices," *IEEE Access*, vol. 7, pp. 87169–87177, 2019.

[42] F. Van den Abeele, J. Hoebeke, G. K. Teklemariam, I. Moerman, and P. Demeester, "Sensor function virtualization to support distributed intelligence in the internet of things," *Wireless Personal Communications*, vol. 81, no. 4, pp. 1415–1436, 2015.

[43] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 32–43, ACM, 2000.

[44] K. A. Faymon, "Microwave beam powered mars airplane," in *Proceedings of the 25th Intersociety Energy Conversion Engineering Conference*, vol. 1, pp. 28–33, Aug 1990.

[45] "Rf energy harvesting solutions." `https://e-peas.com/types/energy-harvesting/rf/`. Accessed: 2019-9-22.

[46] S. Cao and J. Li, "A survey on ambient energy sources and harvesting methods for structural health monitoring applications," *Advances in Mechanical Engineering*, vol. 9, no. 4, p. 1687814017696210, 2017.

[47] P. Dhiman, F. Yavari, X. Mi, H. Gullapalli, Y. Shi, P. M. Ajayan, and N. Koratkar, "Harvesting energy from water flow over graphene," *Nano letters*, vol. 11, no. 8, pp. 3123–3127, 2011.

[48] "ASCII format for network interchange." RFC 20, Oct. 1969.

[49] "Raspberry pi." https://www.raspberrypi.org. Accessed: 2018-10-27.

[50] "Beaglebone black." https://beagleboard.org/black. Accessed: 2018-10-27.

[51] "Bpi-m3." http://www.banana-pi.org/m3.html. Accessed: 2018-10-27.

[52] T. Solc, "Measuring interrupt response times." https://www.tablix.org/\%7Eavian/blog/archives/2016/04/measuring_interrupt_response_times/. Accessed: 2018-11-14.

[53] J. Pihlajamaa, "Benchmarking raspberry pi gpio speed." http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/. Accessed: 2018-11-14.

[54] "Telnet extended ASCII option." RFC 698, July 1975.

[55] "Wordlist." https://weakpass.com/wordlist. Accessed: 2018-11-29.

[56] J. Bilkey, "Oxford english dictionary." https://sites.google.com/a/vhhscougars.org/johnsearch/searchindex/oxford-english-dictionary. Accessed: 2018-11-28.

[57] "first20hours/google-10000-english." https://github.com/first20hours/google-10000-english. Accessed: 2018-11-29.