



**University of
Zurich^{UZH}**

Design and Implementation of Bazo Blockchain for indoor IoT systems with IEEE 802.15.4

*Elfat Esati
Zurich, Switzerland
Student ID: 15-702-103*

Supervisor: Eryk Schiller
Date of Submission: February 5, 2020

Abstract

Using blockchain to store IoT data adds another layer of security that hackers would have to bypass to get access to the data. IoT devices enable us to send data to the blockchain ledger for inclusion in the tamper-resistant records. Different studies at the University of Zurich have given promising results of the integration of IoT devices with the university's own Bazo blockchain based on proof of stake algorithm. In their studies they use LoRa technologies for long range outdoor data delivery with the TTN server to retrieve data from and send to the blockchain. In comparison, the aim of this study is to incorporate the TelosB devices or indoor Iot devices which use Zigbee protocol over the IEEE 802.15.4 communication standard to submit transactions to the Bazo blockchain. The TelosB devices are programmed to work and send transactions to the blockchain over the air through the radio interface. Each transaction is signed with the ED25519 Elliptic Curve Digital Signature Algorithm (EDSA) over the air. Therefore the transactions will be forwarded to the Bazo client with the help of Zigbee plugin to the Bazo client to eventually reach the Bazo miner.

Contents

Abstract	i
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	1
1.3 Work Outline	2
2 Related Work	3
2.1 Blockchain integration with internet of things	3
2.2 Bazo Blockchain	4
3 Design and Implementation	5
3.1 Hardware Components	5
3.1.1 TelosB / Tmote Sky	6
3.1.2 Raspberry PI	6
3.1.3 Contiki Operating System	6
3.2 Design	7
3.2.1 Indoor IoT Architecture	7
3.2.2 Architectural perspective	8
3.2.3 Rasperry PI Architectural perspective	10
3.2.4 Transaction Overview	10
3.3 Implementation	11
3.3.1 Transaction generation from TelosB	11
3.3.2 Zigbee Plugin Implementation	15

4	Evaluation	17
4.1	Security Aspects	17
4.2	Energy Efficiency	17
4.2.1	TX Performance with TelosB	18
5	Summary and Conclusions	23
	Bibliography	25
	Abbreviations	27
	List of Figures	27
	List of Tables	29
A	Installation Guidelines	33
A.0.1	Mandatory Steps for preparing the environment	33

Chapter 1

Introduction

1.1 Motivation

We know that IoT devices have a major security concerns about the data and are open to vulnerabilities that make them a target for attacks. Another issue with the IoT devices is the scalability, since the number of devices grow, having a centralized system to authenticate, authorize and connect such a high number of nodes will result in bottleneck. On the other side blockchain technology mitigate the security and scalability concerns through its decentralized distributed ledger where any mined record can never be altered. We can see that the IoT and the blockchain basically complement each other to form a high potential technology to close any open gaps. In our university projects we figured out that even the use of TTN the data has to flow to reach its final destination the TTN server. Therefore even trusting the TTN server the data on the move from the IoT devices could be exploited without the need to access the IoT device physically. Since TelosB devices are equipped with radio interface we can take that advantage and transfer data in between.

1.2 Description of Work

The aim of this independent study is the implementation of the TelosB devices to embed the data in the form of transactions in the blockchain such that the sender sends packets and the receiver acknowledges the packets received. The sender generates the transaction with the data then the receiver catches those packets and forwards them to the Zigbee plugin for further processing. In order to ensure data integrity, the transactions will be signed with the ED25519 Edwards-curve Digital Signature Algorithm which will be verified against the accumulated data by the Zigbee plugin on the local machine.

1.3 Work Outline

The rest of the report is enumerated in the following order: The second chapter will give a background introduction to the related and most recent studies in the field of IoT and Blockchain. Then design ideas with implementation strategies will be shown in Chapter three, which is followed with the project evaluation in Chapter four. Finally the comparison and recommendations will be found in Chapter five.

Chapter 2

Related Work

In this section we are going to provide and introduce the most recent projects and research that shed light on the integration of IoT devices with the blockchain. The Bazo blockchain of the University of Zurich provides the basis and the motivation to put more effort on this project. Furthermore, we will also introduce and elaborate the hardware and the Operating Systems employed in this project and the reason why we used them.

2.1 Blockchain integration with internet of things

There are quite a number of attempts that shed light on the integration of IoT devices with the blockchain. For example [8, 7] attempted to find out the security gaps that could be filled with the help of the blockchain to ensure the reliability and availability of the data. Other studies [1] attempted to use Ethereum as their blockchain to send data to the Ethereum from IoT devices. So basically, the ideas of these studies are to combat data manipulation risks. Therefore the question that appears here is that these studies barely take into account the different routes the data travel to reach the destination eventually the blockchain. Besides it seems that they do not look at the advantages of the low powered devices or LoRaWAN devices. Therefore they simply use already known protocols such as Wi-Fi or Ethernet. Different from the current research we will be employing the low powered device TelosB which uses the IEEE 802.15.4/Zigbee protocol.

Not to forget are the most recent implementations of the students of University of Zurich, where [2] designed and implemented a system to measure water quality, [2] used Ethereum blockchain to store the data, the data was coming from sensor with the help of Arduino that was then propagated to The Things Network server (TTN) which then further send to the blockchain. Similar to the mentioned study, [3] implemented a payment system where they integrated the Arduino device with the help of TTN to send data to the Bazo blockchain.

In our study compare to the that of the colleagues we do not use any middle software, like what they use in their studies the TTN plugin. Instead we use the TelosB client device to serve as the gateway so that the data reaches the Bazo blockchain.

2.2 Bazo Blockchain

The starting point of this project is the public permissioned blockchain named Bazo blockchain [3]. Bazo blockchain is developed and maintained by the University of Zurich, it is written in the programming language golang and employs the consensus algorithm "Proof of Stake". The reason why they chose proof of stake is because it does not require much energy consumption and the idea of Bazo was more for experimental purposes. Bazo blockchain similar to the Ethereum uses an account based model, meaning that every user has a distinct key pair (public and private). Therefore the public key serves also as the address and the private key as the wallet which never leaves the user's device in our case the TelosB device. The blockchain has been adapted to fit with IoT transactions. The cryptographic function was also modified to become a more lightweight and suitable to the IoT devices with the incorporation of the ED25519 [4] implementing Elliptic Curve Digital Signature Algorithm, a public key asymmetric cryptography. This public key signature system is also compatible with the TelosB devices programmed in C language. This is useful to us since we are using C programming language in the TelosB devices and Javascript in the Zigbee plugin. Therefore although the programming languages differ but yet we are able to verify the signature to check if the data was altered or not. In this study we also proved that the Ed25519 cryptographic algorithm was producing correct results when giving some random input to sign.

Chapter 3

Design and Implementation

3.1 Hardware Components

In this section we will be describing the hardware architecture with their components involved in this project. As the main devices in our project we have TelosB and the raspberry Pi. The hardware components used here will serve for demonstrating the flow of data in the form of transactions that eventually reach the Bazo blockchain to be included in the distributed ledger. In order to program the TelosB devices we also needed a memory efficient operating system and in our case we have used the Contiki Operating System. A global picture of these devices and their interaction with each other is depicted in the Fig 3.1

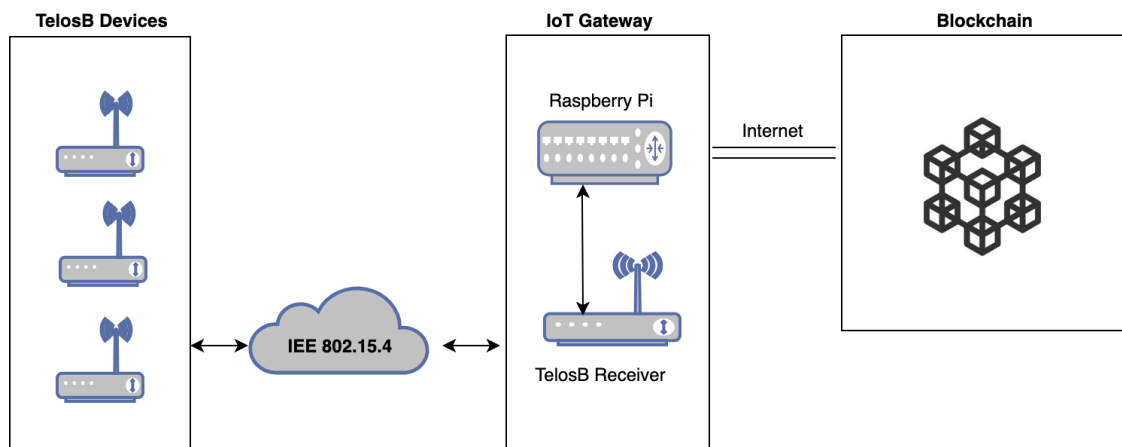


Figure 3.1: Hardware Architecture

3.1.1 TelosB / Tmote Sky

TelosB mote is a wireless sensor node with an open source platform designed for experimentation in the research community. It is a low powered device thanks to its low powered microcontroller MSP430 equipped with 10 kB of RAM, 48 kB of flash, 128 B of data length. We also know that this device is also IEEE 802.15.4 compliant. As by products the TelosB board includes one sensor chip for measuring temperature, humidity and two light sensor chips that capture the photosynthetically active radiation and the total solar radiation. But they are irrelevant to our study. What is important to us is that TelosB is open platform and allows us to program it and serve different purposes. It is also equipped with a USB connector which can be used to program the node and at the same time for powering up the node and sending data to the serial port.

3.1.2 Raspberry PI

Raspberry Pi (RPI) is a little computer device that is equipped with fundamentals for a computer but not necessarily with the latest technology. RPI with its low foot print and energy requirements is meant to promote computer science even to those who cannot afford to buy a real computer. Instead of a hard drive the RPI is equipped with flash memory card where the OS can run. Because of that one can take the advantage and switch between different memory cards and therefore switch between different operating systems and working environment. Here we will not describe in detail the hardware but we will point out the reasons why we chose to use it. RPI is very lightweight and makes the life easier for experimental purposes. In our project RPI serves as the hub where the Bazo blockchain will run. Therefore for the experimental purpose we run the Bazo blockchain locally in the RPI. However RPI is not IEEE 802.15.4 compliant, because of this as you can see in the Fig 3.1 the TelosB Receiver is directly attached to the RPI through the serial port.

3.1.3 Contiki Operating System

The Operating System for wireless sensor networks and TelosB should have a small memory footprint because as we have mentioned earlier TelosB only has 48 KB of flash. With such constraints of this node the general purpose operating system is very difficult to support a large set of functions. Therefore in our study we have used the Contiki Operating System to program the TelosB.

Contiki requires a 10 kB of RAM and 30 kB of ROM for hardware requirements. Contiki has an event driven kernel and to facilitate programming applications uses special threading library called Protothreads. For us to program the TelosB we needed the gcc version for the MSP430 micro controller so that we can compile the program we write. To monitor the serial port traffic we also needed a software which comes as a by product with contiki. Contiki also offer the opportunity to test and run the applications on emulated hardware device in a simulation software named Cooja. In this project it has been very useful. It was not always necessary to test the code in the hardware itself.

3.2 Design

In this section we will be explaining the design decisions taken. Although we do not have a requirement specification document we are still going to show in detail what we are accomplishing with this project. First we will explain from different viewpoints how IoT devices specifically TelosB communicate with each other and we will also elaborate how transactions eventually reach the Bazo blockchain for mining.

3.2.1 Indoor IoT Architecture

In the Figure 3.2 we can have a detailed look of the software components used for the indoor environment of TelosB devices. The communication takes place with two protocols, the Zigbee protocol IEEE 802.15.4 for communication between TelosB devices and the regular internet protocol for reaching the blockchain but for experimental purposes we have the blockchain run locally in the RPI. At the beginning we have just raw data that together with other information will form a transaction which will be mined in the blockchain. For sending one transaction to be mined in the blockchain there are three necessary transmissions to be sent. First the public key is sent to establish the connection with the blockchain and create a wallet in the blockchain in case this is the first time this node sends transactions. Secondly the data itself will be sent with other header information which will be described later. The last step is sending the signature which will be used to verify the integrity of the transaction. In this project we are using two

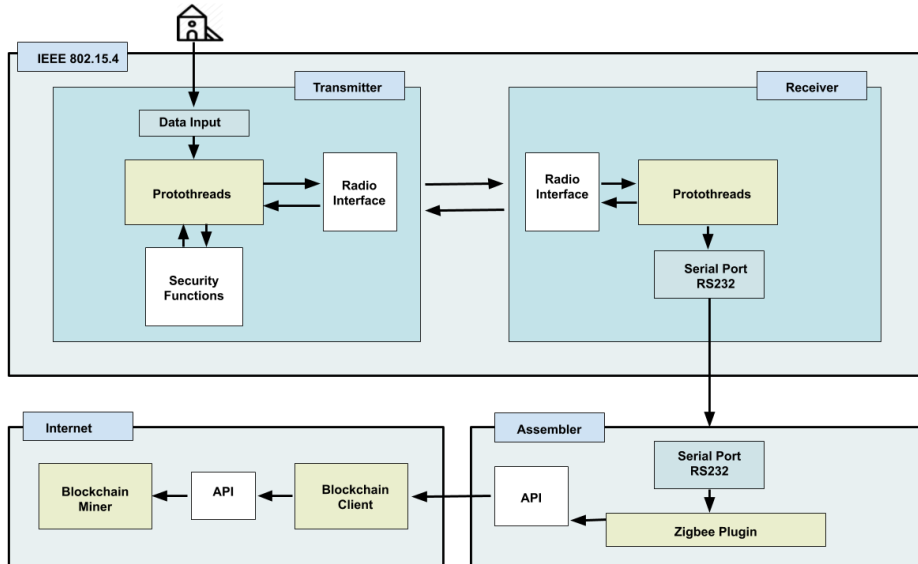


Figure 3.2: Hardware Architecture

TelosB devices that together using the IEEE 802.15.4 medium send transactions to the Bazo blockchain. On the other hand we have the Raspberry Pi that servers as a hub to do the rest of the job. In the RPI we have three components. First we have the Bazo client and the Bazo miner and we also have the Zigbee plugin that awaits data from the TelosB to do the verification and further send the data to the Bazo client for processing.

3.2.2 Architectural perspective

Now we will elaborate on what is needed in the TelosB devices to get the raw data transformed into transactions and sent to the blockchain without losing data integrity. The TelosB devices are equipped with radio antenna however the RPI does not come with radio antenna and is not IEEE 802.15.4 compliant. Anyways we can still take the advantage of the radio antenna to establish communication between the TelosB devices. Therefore the idea is to send transactions from one TelosB to another, transmitter and receiver respectively. The transmitter communicates with the receiver over the air and it doesn't have to be physically connected to receive power as it comes with two AA batteries. After that to reach the blockchain located in the RPI we need to send this data from the TelosB receiver to the RPI. Therefore TelosB receiver is attached to the RPI through the USB plug to the jack of the serial port of the RPI. The data delivery from the TelosB receiver to the RPI is achieved with the help of the RS232 protocol by transferring data bit by bit in hexadecimal numbering system with a baud rate of 115200.

Transmission model for TelosB communication

The Unicast transmission model is employed where data travels from TelosB transmitter directly to TelosB receiver. The sender knows the rime address of the receiver which is hardcoded. In TelosB devices the addresses are RIME addresses which is the address format used in the contiki RIME stack. Rime stack in contiki is an alternative network stack for use when the overhead of the IPv4 or IPv6 stacks is prohibitive. The rime addresses bind together once the receiver is alive, the sender starts sending transactions and the receiver will get the transaction and acknowledge back if it has received. To make sure that the receiver successfully receives the packets implicit acknowledgements and retransmissions are used. This is achieved with the help of callback functions which basically notify the sender. The transmitter job is to basically send the transaction multiple times until it has received an acknowledgment in the form of sequence number that matches with the packet ID attribute of the sender. The medium access control in Contiki by default is the CSMA Contikimac, therefore through our study we also use it and no changes are made.

Security Engines

Data that is generated from the indoor environment devices it is clustered to form a transaction. The transaction includes the header and the data itself which can be seen in the Fig 3.4. Blockchain offers data immutability but this applies only when data is already in the blockchain. In this step the data is not yet inside the blockchain. Therefore to ensure trust, transparency, and data reliability there is still the need to handle the data manipulations risks that could come while data is being transmitted to the blockchain. For these reasons different algorithms and security engines are used to overcome these obstacles. In our case we are using the Public-key cryptography, hosted in the TelosB transmitter and the Bazo blockchain miner respectively. The Bazo blockchain is normally

aware of the sender unless the public key has not yet been added to the blockchain which means the device is not yet registered with the blockchain.

Once there is data in the TelosB transmitter then it will be accumulated in a buffer. When the max data length has been reached the data is then hashed. The maximum packet length in the TelosB is 128 bytes. However this is not entirely data but includes also the header information. In the head we have the source key and the destination key. Both of these keys are hashed. When we have the entire data compact then we use the SHA3-256 algorithm to produce the transaction ID with a length of 256 bit. At this point we employ the ED25519 algorithm to sign the transaction. The ED25519 is a public-key digital signature algorithm based on Edwards-curve Digital Signature Algorithm (EDDSA) which is designed to be faster and more robust compared to the previous schemes without trading off the security aspects. At the TelosB receiver we do not check for data integrity therefore when data arrives the Zigbee Plugin in the RPI it then verifies the transaction ID produced by the Zigbee Plugin with the same algorithm and security functions.

Protothreads

In order to run more efficiently on such small memory systems, Contiki employs protothreads without stack. Protothreads are a type of thread designed to be very lightweight because it doesn't have a separate stack, one stack for all the threads. Although event driven programming could keep the memory down but it adds overhead with the state machine programming style which can be cumbersome to maintain and debug the code. The aim of the protothreads is to implement sequential flow of control without state machines. An example of protothread can be seen in the Listing 3.1

```
static int
protothread1(struct pt *pt)
{
    /* A protothread function must begin with PT_BEGIN() which takes a
       pointer to a struct pt. */
    PT_BEGIN(pt);

    /* We loop forever here. */
    while(1) {
        /* Wait until the other protothread has set its flag. */
        PT_WAIT_UNTIL(pt, protothread2_flag != 0);
        printf("Protothread 1 running\n");

        /* We then reset the other protothread's flag, and set our own
           flag so that the other protothread can run. */
        protothread2_flag = 0;
        protothread1_flag = 1;

        /* And we loop. */
    }
}
```

```

/* All protothread functions must end with PT_END() which takes a
   pointer to a struct pt. */
PT_END(pt);
}

```

Listing 3.1: Protothread function with the name "protothread1"

3.4

3.2.3 Raspberry PI Architectural perspective

In the raspberry Pi we have the core components of the blockchain therefore it serves as a hub for this project. First we have the Zigbee plugin which acts as an intermediary and is responsible for aggregating transactions from the TelosB node to the Blockchain client. The Zigbee plugin adapted from [3] listens to the serial port for the incoming data. The data comes in in hexes and then the plugin converts and offset them to proper format so that it catches the payload. The public key is the first data to be sent from the TelosB node, which serves as the starting point to establish an account in the blockchain and the transactions follow. Regarding the Bazo blockchain we used the adaptation from [3]. In order to save the incoming IoT data to the blockchain we used the same structure of the transaction. From the cryptographic viewpoint we employed the public key cryptography Ed25519.

3.2.4 Transaction Overview

Once the communication link is established the first thing to be sent to the blockchain is the Public Key and then the rest of the data in the form of transaction. In the Fig 3.3 we have depicted what kind of data we send when we deliver the Public Key. The maximum packet size TelosB can support is 128 bytes, this comes from the IEEE 802.15.4 is 128 protocol. Therefore in order to send transaction to the blockchain the first thing blockchain expects to receive is the Public Key. Public Key also serves as the address of the node in the blockchain. In the message structure of this very first packet we have the size of the packet size, the address of the TelosB device which is the rime address of 8 bytes, sequence number of 1 byte represents the sequential transmission of the packet if in case the packet was lost it will attempt to send again and increase the sequence number, then we have Public Key of 32 bytes and lastly the trailing sequence of 2 bytes.

In the Figure 3.4 we have the message structure of a real transaction. We assume that the public key is already sent to the blockchain so that a wallet about this node has been instantiated. TelosB allows us for a maximum packet size of 128 bytes therefore it is up to us to decide how to structure the message and compose it again in the TTN plugin. We have several important parameters without them it would be impossible to issue a transaction in the blockchain. Therefore this is just the representation of the transaction and it is up to us to decide how we want to send the data securely.

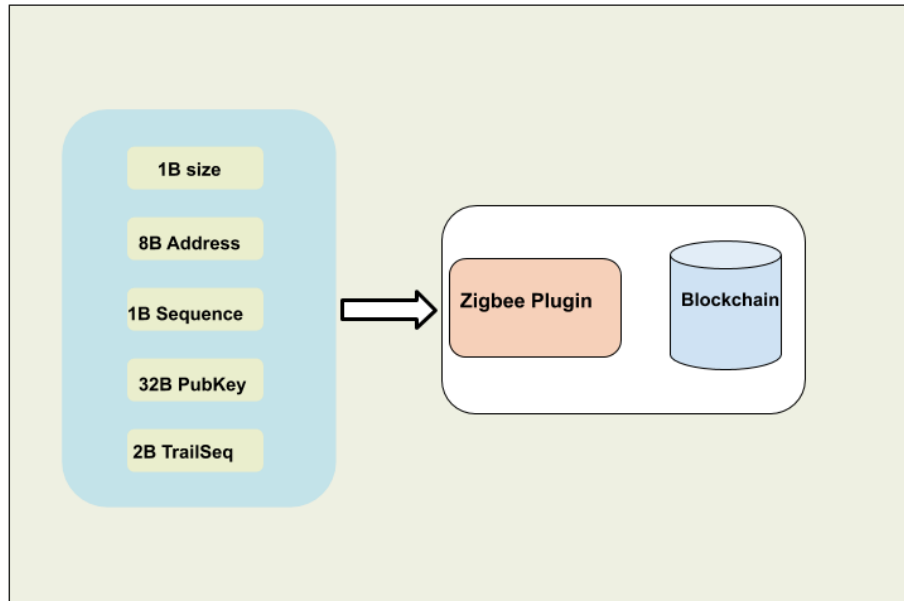


Figure 3.3: Public Key message structure.

3.3 Implementation

We have already talked about the design decisions, the network protocols involved, the transaction structure and the network communication decisions. Therefore in this section we will show what it takes to implement the above mentioned design decisions.

3.3.1 Transaction generation from TelosB

Until now we know that the structure of the packets, the very first is the public key followed by the data or transactions. We have seen in the Fig 3.3 the structure of the message. First we send the public key, then we send the transaction and lastly the signature. Now we will have a look at the implementation of the public key. The public key is sent to basically instantiate an account or address in the Bazo blockchain. With public key we have additional data or header that we need to confirm the public key is intact and nothing can be altered during the transmission. The structure of the message is as follows:

- **Size** The length of the entire packet
- **Node ID** The address of the sender node or TelosB

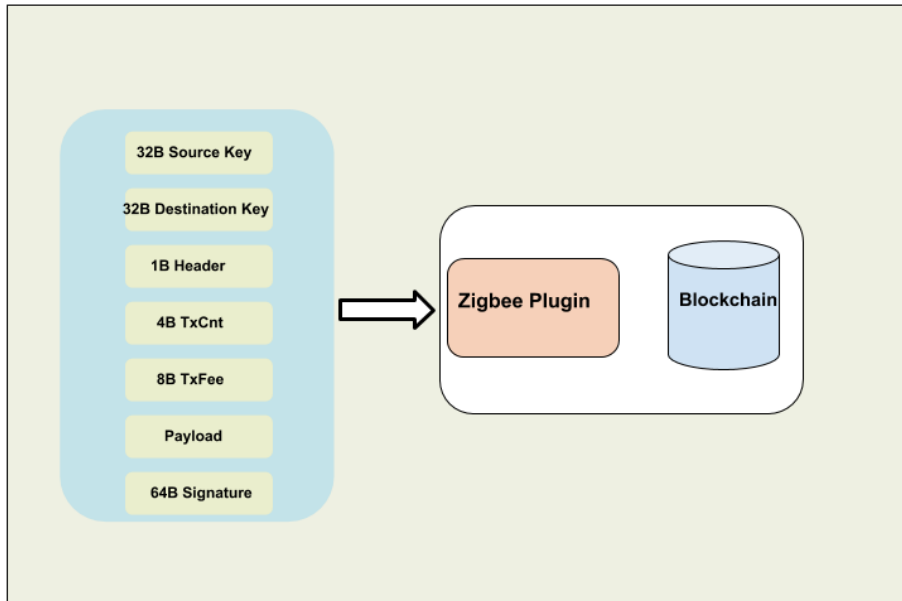


Figure 3.4: Transaction message structure.

- **Sequence** The transmission sequence
- **Public Key** Public key itself
- **Trailing Sequence**

As we can see from the implementation the callback function `rescv_unicast` is used to receive the packets and at the same time transmit them to the serial port with the `printf` function. The packets are sent from the sender using reliable unicast which employs a list to keep track of the packets received. With this model of communication we are using acknowledgements and retransmissions to ensure that the packet is sent successfully. The packets from the client are sent to the Zigbee Plugin which is located in the RPi. The packets are sent in hexadecimal through the serial port. To write data to the serial port from TelosB, messages are printed by calling the `printf` command using a familiar syntax of the C language.

Listing 3.2 showing the implementation of TelosB receiver

```
// TelosB client sending to serial port
```

```

static void
recv_runicast(struct runicast_conn *c, const rimeaddr_t *from, uint8_t seqno)
{
    struct history_entry *e = NULL;
    char *buffer;
    uint8_t pkt_size;

    for(e = list_head(history_table); e != NULL; e = e->next) {
        if(rimeaddr_cmp(&e->addr, from)) {
            break;
        }
    }
    if(e == NULL) {
        e = memb_alloc(&history_mem);
        if(e == NULL) {
            e = list_chop(history_table);
        }
        rimeaddr_copy(&e->addr, from);
        e->seq = seqno;
        list_push(history_table, e);
    } else {
        if(e->seq == seqno) {
            printf("runicast message received from %d.%d, seqno %d (DUPLICATE)\n",
                from->u8[0], from->u8[1], seqno);

            printf("Received from %d.%d: '%s'\n",
                from->u8[0], from->u8[1], (char *)packetbuf_dataptr());

            return;
        }

        e->seq = seqno;
    }

    buffer = (char *)packetbuf_dataptr();
    pkt_size = (uint8_t) packetbuf_datalen();

    // SIZE 1 B
    printf("%02x", pkt_size);

    // LONG NODE ID 8 B
    for(inc = 0; inc < 8; inc++)
        printf("%02x", from->u8[inc]);

    // SEQ 1 B
    printf("%02x", seqno);

```

```

// BUFFER VARIABLE SIZE
for(inc = 0; inc < pkt_size; inc++)
    printf("%02x", buffer[inc]);

// TRAILING SEQUENCE 2 B
printf("\n");

}

```

Listing 3.2: Sending public key to the serial port

Now we will explain the most important variables that are needed to reach the blockchain and be mined. In the first two arrays we have the destination address and the source address, which is basically nothing else but the public keys of two nodes each with 32 bytes in size. After that we have the transaction counter **TxCnt** which is basically used to differentiate a transaction with exact same data in case it is sent again. Because if a transaction is mined in a block it won't allow to have duplicates inside. After that we have the **TxFee** which is the amount rewarded to the blockchain for mining the transaction in a block. And finally we have the data in the variable named **message** which is of 20 bytes.

```

static uint8_t dstpk[crypto_sign_PUBKEYBYTES]={0x67, 0x27, 0xf5, 0x1d,
    0xb4, 0x14, 0x46, 0xe0, 0x40, 0x26, 0x1f, 0x5b, 0x08, 0x8d, 0x32, 0x94,
    0x4a, 0xe0, 0xb2, 0xe1, 0xa9, 0xf4, 0x7b, 0xec, 0x97, 0x3e, 0xcf, 0x75,
    0x96, 0xec, 0xd6, 0x9a}; //32

static u8 pk[crypto_sign_PUBKEYBYTES] = { 0x3d, 0x40, 0x17, 0xc3, 0xe8,
    0x43, 0x89, 0x5a, 0x92, 0xb7, 0x0a, 0xa7, 0x4d, 0x1b, 0x7e, 0xbc, 0x9c,
    0x98, 0x2c, 0xcf, 0x2e, 0xc4, 0x96, 0x8c, 0xc0, 0xcd, 0x55, 0xf1, 0x2a,
    0xf4, 0x66, 0x0c };

static uint8_t TxCnt[] = { 0, 0, 0, 0 };
static uint8_t TxFee[] = { 0, 0, 0, 0, 0, 0, 0, 0, 1 };
static uint8_t Hdr[] = { 0 };
static uint8_t message[] = { 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
    0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20 };
static uint8_t toSign[97];

```

Listing 3.3: Transaction with header and data

In the Listing 3.4 below we can see the use of hash function SHA3-256 where the hash of destination address is generated. In this case the function **sha3_Update** takes three arguments, the public key and its size with the context. The function is called two times once for the destination public key and once for the source public key. After that the function **memcpy** is called to assemble the variables into one single packet which will be sent.

```

printf("Generating SHA3-256 Signature of the DSTPK...\n");

sha3_Init256(&context);
sha3_Update(&context, dstpk, sizeof(dstpk));
hash = sha3_Finalize(&context);

memcpy(toSign, hash, 32 * sizeof(uint8_t));

sha3_Init256(&context);
sha3_Update(&context, pk, sizeof(pk));
hash = sha3_Finalize(&context);
memcpy(&toSign[32], hash, 32 * sizeof(uint8_t));
memcpy(&toSign[64], TxCnt, 4 * sizeof(uint8_t));
memcpy(&toSign[68], TxFee, 8 * sizeof(uint8_t));
memcpy(&toSign[76], Hdr, 1 * sizeof(uint8_t));
memcpy(&toSign[77], message, 20 * sizeof(uint8_t));

```

Listing 3.4: Assembling transaction into a buffer to be sent

3.3.2 Zigbee Plugin Implementation

Zigbee Plugin stands in the RPI that waits for incoming data from the serial port. The implementation is adapted from another project at the University of Zürich. It first extracts the transaction and the rest follows the same as in the transaction generation. Since we never know what has happened to the data while on transfer, once some data is received it will be loaded inside a buffer and accumulated until the signature is received. Once the signature is received the data verification takes place. First a transaction ID is created from the accumulated data then it will be verified with the signature received in the last packet. Besides the idea is to not flood the blockchain miner with invalid transactions. The function for verifying the transaction in the Listening 3.5 shows the similar way as how it was done in the TelosB transmitter. The same security functions are applied but in now in Javascript.

```

function verifyData(){
  if (ed25519.Verify(data, new Uint8Array(sign), new Uint8Array(pubK))) {
    console.log('Signature valid');
  } else {
    console.log('Signature NOT valid');
  }
}

```

Listing 3.5: Verifying transaction with the provided signature

Chapter 4

Evaluation

In this section we will talk about the accomplished work and evaluate the reasons we chose different design decisions and security algorithms. In addition, we measure the energy efficiency of the TelosB devices. So, for in each design decisions there is a trade off on why we decide to use different algorithms over the others.

4.1 Security Aspects

The main approach in this project ensures data integrity and mitigates manipulations risks while data is being transmitted. First the data is sent as it is without encryption and after that it is signed and the signature is sent. The same is done on the Zigbee plugin, the data is accumulated and then again signed and the signature is compared. Using the same algorithms in both ends we could verify that data was not altered. This was achieved with the use of SHA3-256 and the ED25519. Although the ED25519 seem to be a bit slow but yet it offers the possibility to run in different programming languages as in our case the Zigbee plugin is implemented in Javascript while the TelosB devices are programmed in C. Besides ED25519 is a concise implementation, very robust and compatible with different programming languages. But due to limited instruction set of 16 bit of the MSP430 CPU in the TelosB mote it can take more time than expected on contemporary computers. The MSP430 has a very simple but very limited instruction set. The register operations take up 1 processor cycle. However accessing the memory takes 3 cycles per memory operation.

4.2 Energy Efficiency

The approach used in this project is to send data from the TelosB data generator to the TelosB receiver which forwards the data to the Zigbee plugin to eventually reach the Bazo blockchain. Therefore once the data reaches the Zigbee plugin safe and sound then the aggregation to the Bazo blockchain is easily achieved with high accuracy. When the data

is generated in the TelosB sender it is transmitted to the TelosB receiver through the IEEE 802.15.4. In this project, performance is evaluated against the power consumption of the TelosB devices, the number of packet re transmissions and acknowledgements and the traffic volume in bytes.

The TelosB device sends traffic with bandwidth of 250 kbps with a packet size of 128 bytes. TelosB employs the CC2420 Transceiver chip compliant with IEEE 802.15.4. The default transmission power (TX) of CC2420 is equal to 0 dBm which is equivalent to 31 power amplifier (PA) level, therefore this is also used as a default in this project.

The distance between TelosB is set to direct line of sight and non direct line of sight, in office and outside office respectively. A packet is retransmitted up to a configurable retry number, if its corresponding ACK does not arrive on time. The evaluation of the employed reliable unicast scheme is shown in Fig 4.1 and Fig 4.2 which compare the transmission performance based on the number of packets sent including retransmissions, and the traffic volume measured in bytes, respectively. In the Fig 4.1 (x axis) presents the number of packets needed to be sent and signed and the traffic exchanged in the network including retransmissions in y axis. The traffic is comprised of three different groups for direct line of sight and non line of sight: (a) Data (i.e payload) which increases gradually in sending data chunks of 20B per TX, (b) ACK (18), number of acknowledgments in return to successful transmission and (c) Signature (64). The packet and traffic overhead is depicted in the y axis. There is a slightly higher number of ACKs compare to data packets in both environment line of sight and non line of sight, whereas the signature is stable and not much overhead is noticed. Furthermore, the performance between the line of sight and non line of sight is also slightly differentiated.

4.2.1 TX Performance with TelosB

A similar setup is used to measure the energy consumption of the TelosB sender in terms of the number of chunks transmitted. The TelosB sender needs to do some computation so that it doesn't only send the data but also signs the data. We assume the public key is already registered in the Bazo blockchain. Once the sender wakes up, it then starts sending data packets towards the TelosB receiver once there are enough data collected in form of chunks of 20B. For reliability we have also added small waiting time in between the transmissions which is equal to 5 clock seconds of the Contiki Rime stack. The procedure requires computing of the signatures. The measurements in mWh collected are depicted in the Fig 4.3. The ED25519 is computationally expensive and therefore takes the highest span. Due to this we can see that the highest the number of samples or packet transmissions the highest the energy effectiveness. This happens because packet transmission does not require much time and eventually we recommend to send multiple data chunks and then sign. It would be inefficient if we had to send small amounts of data multiple times, which means the data has to be signed multiple times and that is costly.

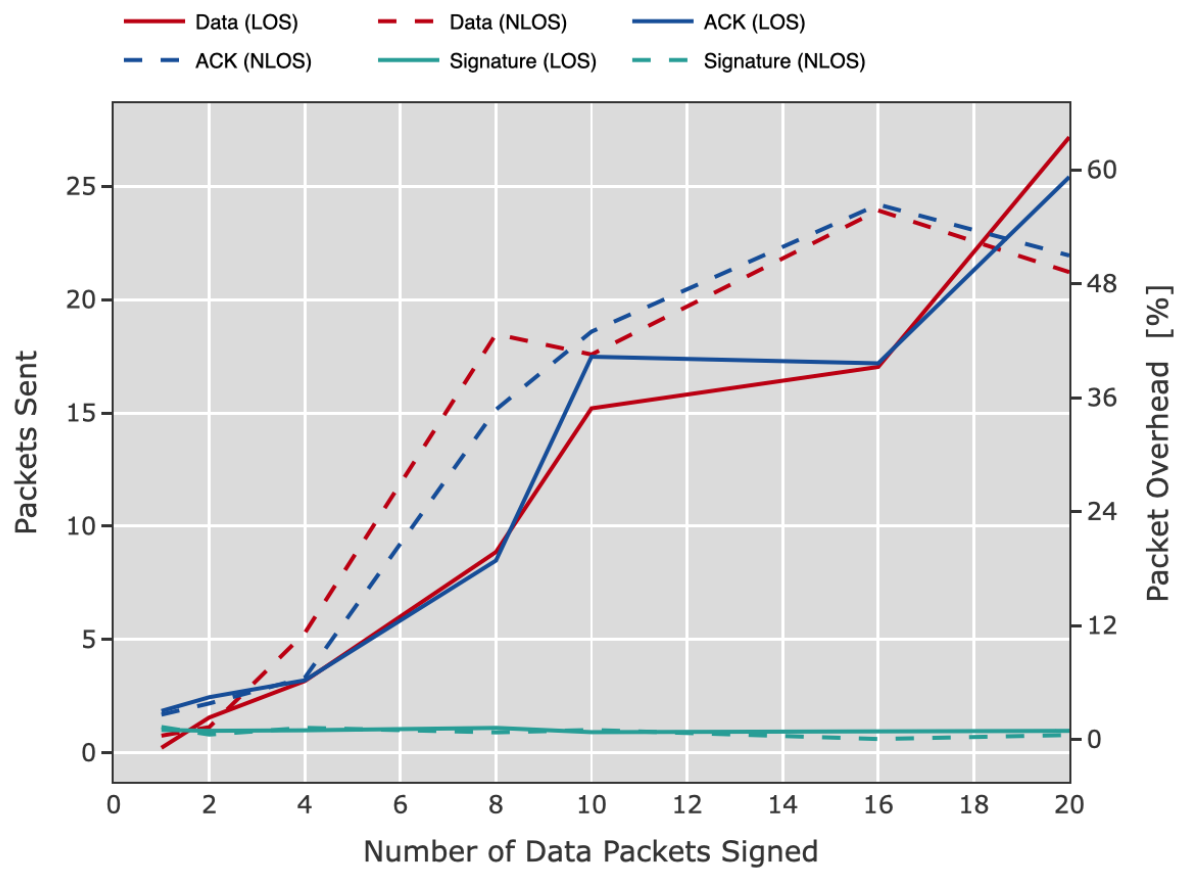


Figure 4.1: Packet Volume Exchanged in The Network.

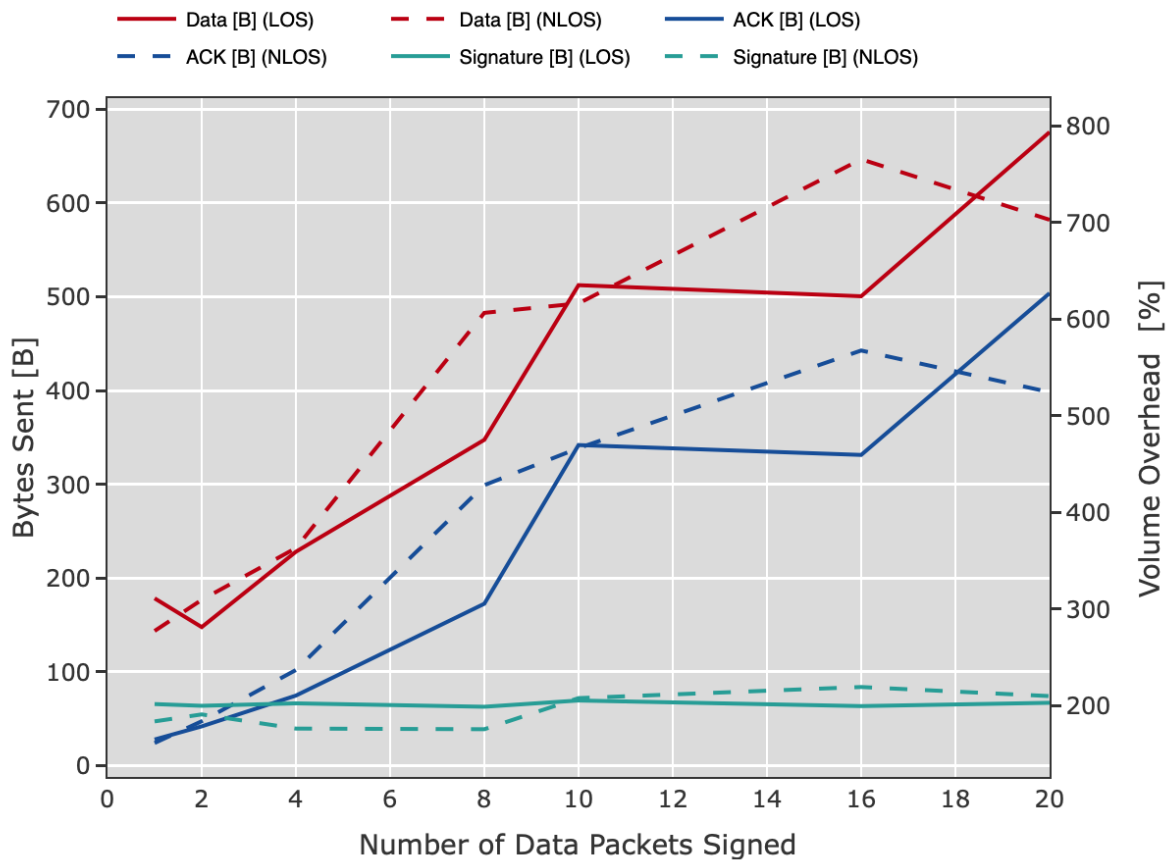


Figure 4.2: The Total Traffic Volume Exchanged in The Network.

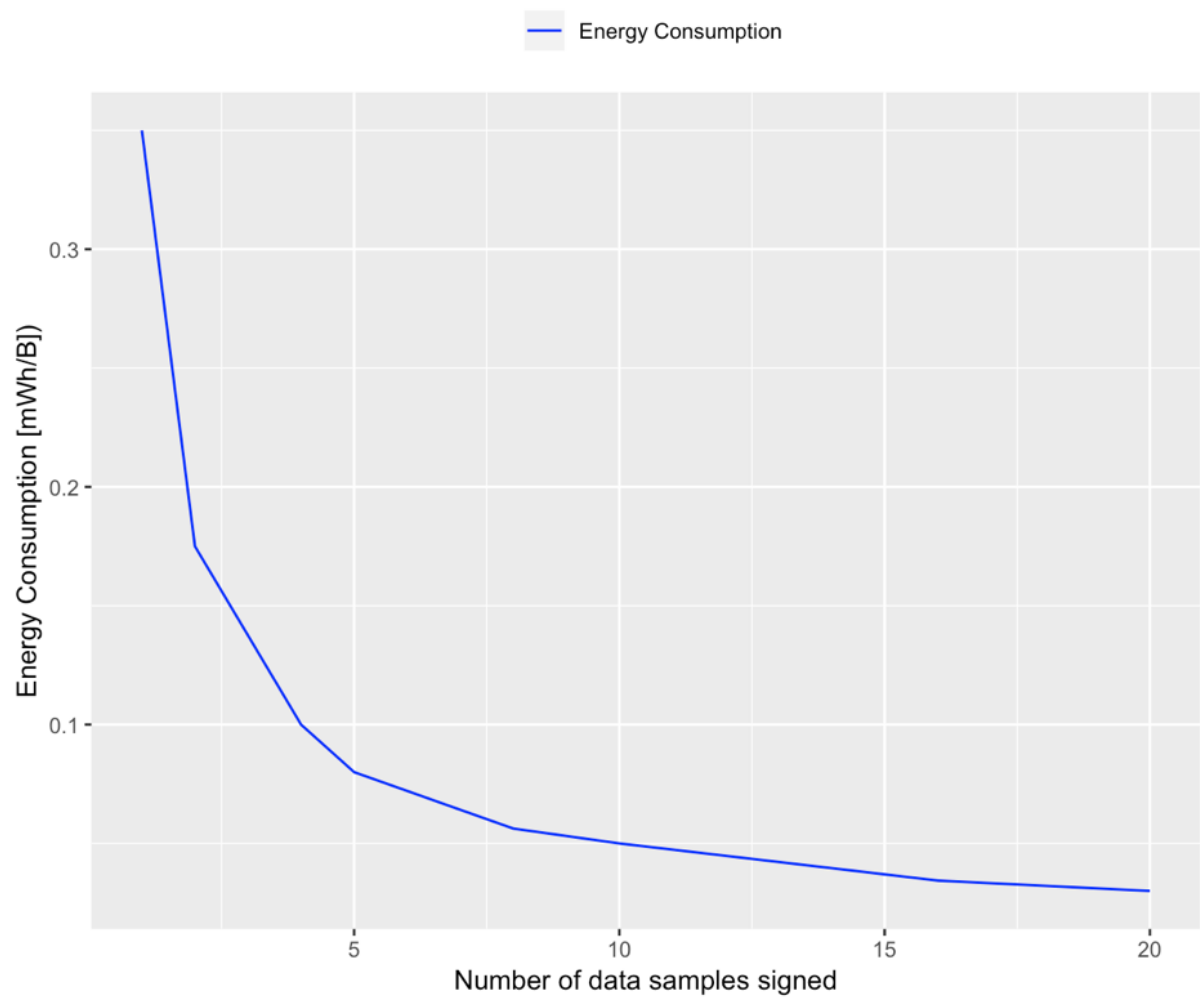


Figure 4.3: Energy Performance.

Chapter 5

Summary and Conclusions

The potential of using blockchain for IoT devices is highly considered recently. The main idea behind this project is to materialize the use of indoor IoT systems specifically TelosB to work and communicate with the Bazo Blockchain so that data can be mined in the distributed ledger. In addition, in this project we carefully considered the data manipulation risks that could come while data is on transmission from the sender to the receiver. Therefore different security engines and strategies have taken place to close this gap. Eventually the whole system can be considered safe. In order to achieve this implementation we used the TelosB devices and RPI to accomplish the goal. For experimental purposes the blockchain is run locally in the RPI, this is because we did not want to create more overhead but accomplish the idea. The blockchain can be located in different places and be accessed with a standard internet protocol. The RPI used in this project should not limit the scope, therefore it should not stop from using different computers or Operating Systems.

This project is also another proof of IoT devices to work with heterogeneous environments and programming languages. We also conclude that TelosB devices work best when first all data is sent and then signed. The SHA3-256 always gives a fixed length transaction ID, therefore it doesn't matter how much data is sent since the ED25519 will always consume fixed time interval. To encapsulate the design ideas in this project prove that indoor IoT devices together with the blockchain form a highly secure system with heterogeneous devices.

Bibliography

- [1] S. Huh, S. Cho and S. Kim, "Managing IoT devices using blockchain platform," 2017 19th International Conference on Advanced Communication Technology (ICACT), Bongpyeong, 2017, pp. 464-467. URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=789013&isnumber=7890033>, accessed December 2019.
- [2] Sanjiv Subodhnarayan Jha, Design and Implementation of an Integrated Water Quality Monitoring System and Blockchains, January 25th, 2018.
- [3] Ile Civelov, Design and Implementation of an IoT-based Hierarchical Payment System with BAZO Blockchain, Apr 11th, 2019
- [4] Edwards-Curve Digital Signature Algorithm (EdDSA), <https://tools.ietf.org/html/rfc8032>, accessed December 2019
- [5] NaCl: Networking and Cryptography library <https://nacl.cr.yp.to/sign.html>, accessed December 2019
- [6] Documentation: Processes and events, <https://github.com/contiki-ng/contiki-ng/wiki/Documentation:-Processes-and-events>, accessed November 2019.
- [7] Minhaj Ahmad Khan, Khaled Salah, IoT security: Review, blockchain solutions, and open challenges, Future Generation Computer Systems, Volume 82, 2018, Pages 395-411, ISSN 0167-739X, <https://doi.org/10.1016/j.future.2017.11.022>. (<http://www.sciencedirect.com/science/article/pii/S0167739X17315765>)
- [8] A. Dorri, S. S. Kanhere, R. Jurdak and P. Gauravaram, "Blockchain for IoT security and privacy: The case study of a smart home," 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, 2017, pp. 618-623. doi: 10.1109/PERCOMW.2017.7917634 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=7917634&isnumber=7917470>
- [9] M. Samaniego and R. Deters, "Blockchain as a Service for IoT," 2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Chengdu, 2016, pp. 433-436. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.102 URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=7917130&isnumber=7917027>

- [10] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. 2017. Towards an Optimized Blockchain for IoT. In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation (IoTDI â17). Association for Computing Machinery, New York, NY, USA, 173â178. DOI:<https://doi.org/10.1145/3054977.3055003>
- [11] How protothreads really work <http://dunkels.com/adam/pt/expansion.html>, accessed December 2019
- [12] Contiki Input and Output <https://github.com/contiki-os/contiki/wiki/Input-and-output>

Abbreviations

Ack	Acknowledgement
API	Application Programming Interface
ECDSA	Elliptic Curve Digital Signature Algorithm Ed25519
tx	Transaction
txCnt	Transaction Counter
txFee	Transaction Fee

List of Figures

3.1	Hardware Architecture	5
3.2	Hardware Architecture	7
3.3	Public Key message structure.	11
3.4	Transaction message structure.	12
4.1	Packet Volume Exchanged in The Network.	19
4.2	The Total Traffic Volume Exchanged in The Network.	20
4.3	Energy Performance.	21

List of Tables

Appendix A

Installation Guidelines

A.0.1 Mandatory Steps for preparing the environment

Contiki OS comes with many by products that are necessary to compile, run and test the applications, however the downside of it is that it is pretty tough to compile an application without a proper IDE for example it would be difficult to debug at times. To compile an example code we use the following command :

```
https://github.com/eesati/Master-Thesis.git
```

To program the TelosB device we used the intuitive command:

```
make TARGET=sky example-name.upload.
```

To observe the outcome of the program executed on the telosb device, we use the serial dump traffic monitoring in the `/contiki-2.7/tools/sky` directory with the following command:

```
./serialdump-linux -b115200 /dev/ttyUSB0
```

Once the TelosB devices are programmed and ready we can then start the Zigbee plugin. To start one simply has to go to the directory where the files are located and simply use this command:

```
node Zigbee.js
```

To start the Bazo miner from this directory `/home/ttn/go/src/github.com/bazo-blockchain/bazo-miner/` :

```
bazo-miner start âdatabase StoreA.db âaddress localhost:8000 âbootstrap localhost:8000  
-2âwallet WalletA.txt âcommitment CommitmentA.txt âmultisig WalletA.txt ârootwallet  
WalletA.txt ârootcommitment CommitmentA.txt
```

To start the Bazo Client from `/home/ttn/go/src/github.com/bazo-blockchain/bazo-client/` :

```
bazo-client rest
```