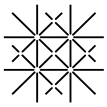

University of Basel

Thesis Title

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Computer Networks Group
https://cn.dmi.unibas.ch/

Examiner: Prof. Dr. Christian Tschudin
Supervisor: Fabrizio Parrillo

Simon Laube
simon.laube@stud.unibas.ch
16-635-021

June 20th 2022

Abstract

In a wireless sensor network with intermittent connectivity, continuous operation requires unsupervised resource management. This project builds a demonstrator for sampling-rate adaption and data thinning that is dependend on memory availability and data drainage progress. The distributed system will consist of constrained devices running on solar power.

Table of Contents

1 Introduction

A network built of small, solar powered devices can be used for a wide variety of different applications. Some possible use-cases could be monitoring agricultural sites, recording weather data for scientific purposes or serve as an independent network for communication. For those devices (network nodes) to be autonomous, it is essential to reduce their energy consumption as much as possible. To transmit data between network nodes, we therefore need wireless technology that has a low power consumption. In this project we use LORA (Long Range) wireless technology which fulfils this requirement. Additionally to having a very small energy consumption, it is also capable of transmitting signals over distances greater than 15 kilometres under optimal conditions. [1] The tradeoff is a smaller data transmission rate.

To manage and propagate data through the network, we use a protocol called 'TinySSB'. It is heavily influenced by the Secure Scuttlebutt Protocol which is an 'event-sharing protocol and architecture for social apps.' [3] TinySSB is optimised for small microcontroller devices by omitting data that is not essential while still guaranteeing authenticity [Later More]. Both protocols use so-called 'append-only-logs' or 'feeds' to append new packets to. Once a network node trusts the head packet of a feed, it can autonomously verify if a new incoming packet is the correct continuation of the last received packet. With this practical feature it is even possible to receive packets that have been propagated via a number of different nodes and still guarantee that no middle man could have altered the data. However the drawback of this method is limited flexibility in changing or deleting old data. Since the devices that are used for this project have only limited storage capability we need to find a way to be able to remove old data while still keeping the secure properties of the append-only-logs.

In this thesis we will have a look at two different approaches of achieving this goal for two different use-cases. We will explore how we can use multiple different feeds and combine them to bigger constructs (feed-trees) using packets that serve as pointers to other feeds. These pointers help us to create a link between different feeds that can be differently interpreted depending on the type of the feed-tree. Those feed constructs use simple feeds and pointers as building blocks but can get quite complex when interacting with each other. We want to hide this complexity from the user and present him with only an abstraction of the feed-tree that can be used like a normal feed with additional functionality.

Furthermore we want to have a look at how we can improve resource management in general. This includes sorting incoming and outgoing packets according to priorities to reduce the number of sent packets. Some feeds have to be given more computing time than others depending on how critical they are for either a specific feed construct or for the system as a whole.
While the focus of this project is not the real-life implementation of the device with solar cells, it should be as well prepared as possible for this scenario. Therefore the system needs to be able to withstand unexpected shutdowns due to power outages and recover later once enough power is available again.

2
Background

Important concepts that are used in this thesis are presented in this section. A Summary of LoRa Technology, a short description of the Pycom devices and an overview of the TinySSB Protocol and its functionality follow.

2.1 Long Range (LoRa) Wireless Technology

In recent years the number of devices connected to the Internet of Things (IoT) has been increasing rapidly. Since a lot of those devices do not need to transfer high amounts of data and rather focus on using as little energy as possible, LoRa and similar technologies have been developed to serve those use-cases. In this section we will give a short overview of how LoRa works and how we can adjust its parameters.

LoRa promises long battery life, far-reaching communication distances (10 to 15 kilometres if devices are in line-of-sight) and high node density (nodes in close proximity can operate at the same time if configured correctly). The downside of having all those practical features is a lower data rate. To achieve this, it uses chirp spread spectrum modulation. Data that has to be sent is converted into *upchirps* and *downchirps*. Chirping up or down refers to a signal that is sent with constantly increasing or decreasing frequency. To adjust the transmission settings optimally to specific environments, different parameters that alter the chirps can be configured individually. One of those parameter is the **bandwidth**. The bandwidth is the difference in frequency of the start and end of one chirp and is typically set to 125, 250 or 500kHz. A higher bandwidth increases the acceptable transmission distance. Another parameter is the **spreading factor**. It determines the angle of one chirp. The higher the spreading factor, the longer it takes to complete one chirp. This can increase the transmission distance but reduces the data rate. Additionally to the physical settings we can set different **coding rates** which determine the forward error correction. A higher coding rate leads to a more reliable reception of packets and a lower data rate. For our project we use a spreading factor of 7, bandwidth of 250kHz and a coding rate of 4/7. These settings have worked well to test the software but need to be reconsidered once the devices get deployed depending on the surrounding conditions.

In Fig. 2.1 we can see ten upchirps (preamble) and two downchirps (start frame delimiter)

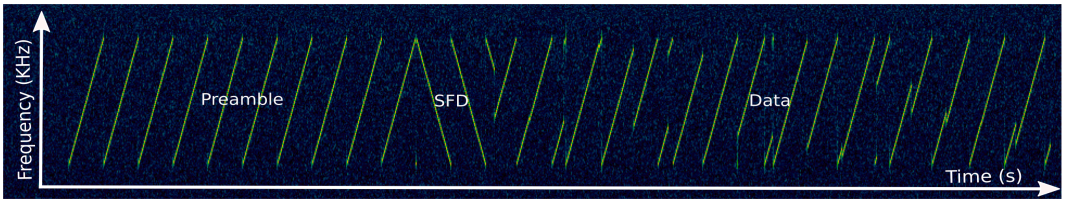

Figure 2.1: A visualized lora packet transmission. Source: [2]

followed by modulated chirps that contain the actual data.
(Sources for Section 2.1 are [1] and [2])

2.2 Overview of TinySSB

2.2.1 Pure25519 Signature and Hash functions

Two very important concepts in TinySSB that are used at various different points in the system are signature and hash functions. [COMPLETE THIS]

2.2.2 Signing and Verifying Packets

TinySSB (Tiny Secure Scuttlebutt) is a protocol that is heavily influenced by SSB (Secure Scuttlebutt). An important concept of both protocols is the use of 'append-only-logs' or 'feeds'. In TinySSB a feed consists of a 128 byte feed header block containing some general information about the feed and any number of 128 bytes log-entries appended after it. Every node has one ID feed to be identifiable and an arbitrary amount of child and continuation feeds. [ADD REF]

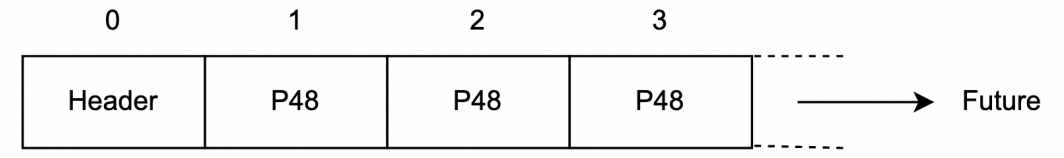

Figure 2.2: Feed with the header block and three appended log entries. (P48 describes the standard packet type with 48 bytes payload)

To create a feed, a key-pair has to be calculated first. This is done using ed25519 elliptic curve cryptography. [ADD SOURCE]A key-pair consists of a public key - which also serves as the name and identification of the feed (feed ID) - as well as a secret key. These two keys have the following essential property for the TinySSB feed scheme. If a message is cryptographically signed with the secret key, the signature can (only) be verified with the corresponding public key. Every packet that is created and appended to a feed has to be signed by the feed owner with his secret key. This ensures that other nodes in the network - which know the feed ID - are able to verify that a message was indeed signed by the feed owner.
In Fig. 2.3 we see two packets with the signature appended at the end. The first 8 bytes are not used and reserved for encryption purposes that will be implemented at a later stage.

The next 7 bytes are used for demultiplexing. [ADD REF] The next byte specifies the type of the packet and the remaining 48 bytes are interpreted differently depending on the packet type. [CHECK THIS!]Using the first 64 Bytes, the signature and the public key of the feed, any node in the network can verify that the signature was created with the secret key corresponding to the public key and therefore trust the packet.

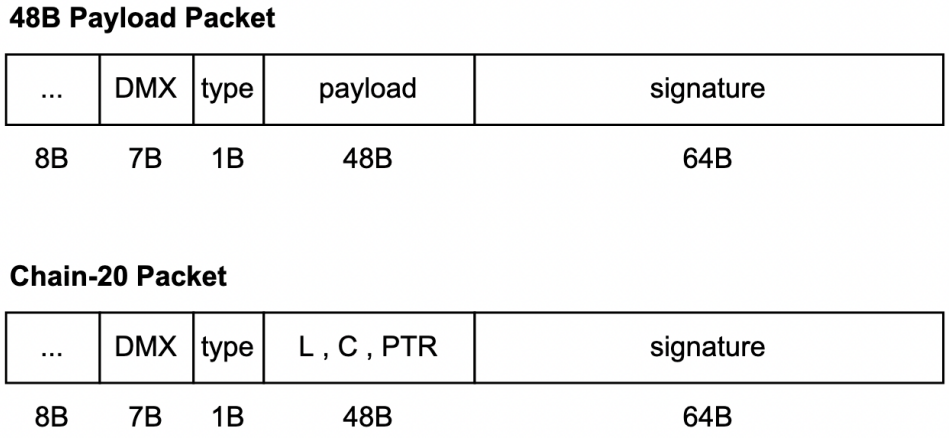

Figure 2.3: Layout of two different packet types in TinySSB.

2.2.3 Sending Data through the Network

To propagate data through a TinySSB network, a receiving node has to replicate the feed of the sender. This means that if a feed gets transferred through a series of nodes, in the end every one of those nodes has a copy of the same feed stored on their device. (some replicas may be at a different state and have already more packets appended than others) As explained in Section 2.2.2, while replicating the feed, every node has to verify the signature of every packet starting at packet one up until the last packet in the feed. Suppose one node does not verify a malicious packet or intentionally appends a wrong packet to the feed, nodes that will receive those altered packets will not be able to verify them with the feed ID and therefore drop them immediately.

2.2.4 Trust Anchors

In Section 2.2.3 we explored, how after successful verification nodes can append packets to feeds. Once a node has a replica of another feed, this is a straight forward process. However this does not yet explain how we can start a replication feed. If a node does not know the feed ID of a feed it wants to replicate, it cannot verify and trust its packets. To overcome this problem, we have to introduce **trust anchors**. Before a node can receive any packets

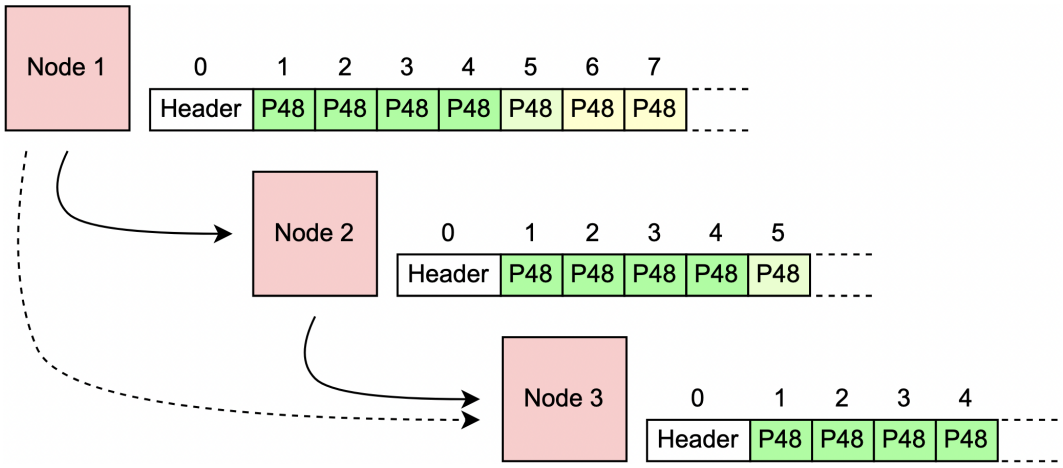

Figure 2.4: The same feed forwarded over a series of three nodes. Node 1 is the feed owner and has already appended new packets that are not yet forwarded. Node 2 and 3 create their own replicas and try to catch up with Node 1. If Node 1 and 3 are geographically close enough, packets may even reach Node 3 directly from Node 1.

from other nodes in the network, it has to be manually[1] initialized with trust anchors. A trust anchor refers to the feed ID of an identification feed of a node. If a trust anchor is installed, the node has the ability to verifying packets of the corresponding feed and start replicating it. While the first packet of a feed is directly dependent on the trust anchor to verify the signature, all the following packets can only be verified if a hash value of the previous packet is available. Since we have already established that we trust the previous packet, we can for that reason also trust the next packet. With this technique a trust chain is built that guarantees authenticity[2] and integrity[3]. [REF POC?]

2.2.5 Requesting Packets

An important concept in TinySSB is *how* packets are propagated through the network. Since the whole system builds on append-only-logs, every received packet has to be appended to a log. As a result of this, every node knows which packets it is expecting, namely for every feed the packet that follows the currently newest packet. We can make use of this convenient property by creating a list containing a 7 bytes long demultiplexing value[4] for every expected packet. With the help of this list, nodes can efficiently handle incoming packets. If the DMX field of an incoming packet is not contained in the DMX list, it can immediately be dropped. If it appears in the list, we can directly try to verify and append it to the corresponding feed. Using this DMX list, it is not necessary to include the feed ID, the sequence number or a backlink to the previous message in the packet. This info is concisely represented in the

[1] At least one anchor has to be initialized manually. It would be possible to implement a higher authority / admin feed, that appends further anchors to its own feed and that way propagates anchors through the network.
[2] No third party can alter the message.
[3] The message can only have been created by the owner of the feed.
[4] DMX value is calculated using the previous log entry, the feed ID, sequence number and a TinySSB specific version prefix. [REF TO TINY PAPER]

DMX value and therefore the packet size can be reduced to a minimum. (As opposed to the SSB implementation where this data and some additional information fields are included in every message [3])
Contrary to what Fig. 2.4 might imply on first glance, network nodes do not arbitrarily send packets and hope that some other node receives them. This would be a very inefficient method. Instead - because nodes know which packets they are expecting - nodes will request the packets with a 'want broadcast'. This broadcast contains the feed ID[5] and the sequence number of the desired packet. If a node that has the requested packet stored on its device receives such a message, it will send back the specified packet. This scenario is visualized in Fig. 2.5. If the want request is unsuccessful and no packet is returned, the node has to wait for a given amount of time before trying to request the same packet again.

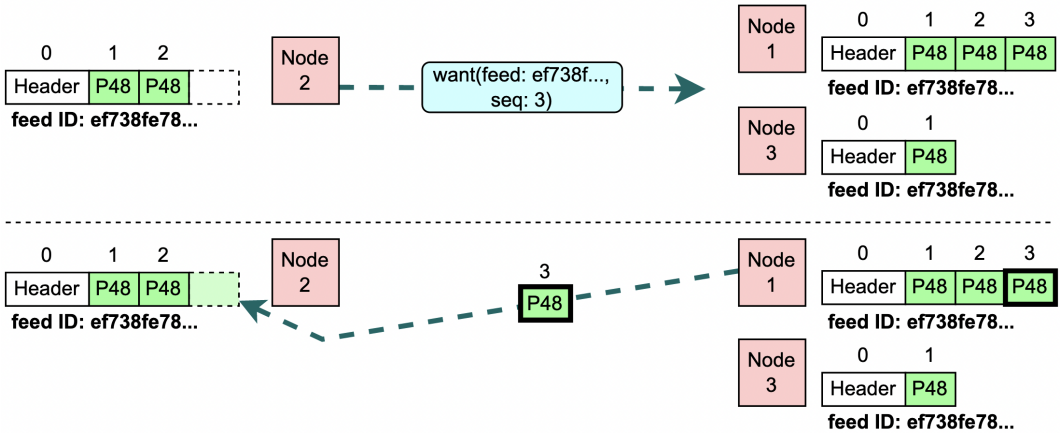

Figure 2.5: Node 2 requests packet with seq. nr. 3. Node 1 returns the requested packet. Node 3 does not have the requested packet and therefore does not answer.

2.2.6 Side Hashchains

When using 48 bytes payload packets, the size of data to be transferred is limited to 48 bytes or has to be spilt up into chunks of 48 bytes with each chunk being part of a different packet. This would lead to a lot of overhead in terms of storage and network use. Therefore a more concise and efficient method is to use side hash chains. The data of a hash chain is (for the most part) not stored in a feed. Only the initial packet that points to the start of the chain is appended to the feed as a packet with type chain-20 (See second packet in Fig. 2.3). This packet - apart from the DMX, type and signature fields - contains the length of the hash chain, the first part of the payload and a hash pointer to the start of the chain. For chain packets, all fields that are typically used in standard packets can be omitted. Of the 120 bytes 100 are used for payload and 20 for the hash pointer to next chain packet. A hash pointer is created by hashing the next chain packet. If the hash of the incoming chain packet matches the hash pointer of the previous packet, we can guarantee that the packet is the legitimate continuation of the chain. To be able to create such a chain, the sender

[5] Instead of the complete feed ID the reduced form sha256(feed ID + b'want')[:7] + seq is sent.

must start by creating the last chain packet first and and appending its hash value to the second last packet. This step has to be repeated until the whole payload is split into hash chain packets before the first packet can be sent. A visualization of a sidehash chain is seen in Fig. 2.6.

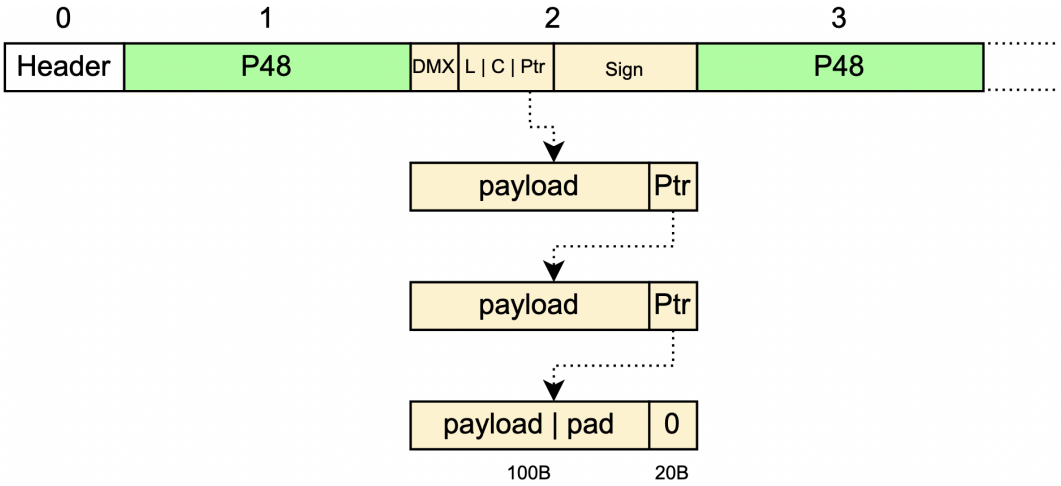

Figure 2.6: Feed containing three packets. The second packet is of type chain 20 with a pointer to the next chain packet. In the last chain packet, the pointer is set to zero and a padding is added between the end of the payload and start of the pointer.

When a chain 20 packet is at the end of a feed, the whole chain has to be completely loaded before the next packet can be appended. The DMX table must be updated by replacing the old DMX value with the hash pointer to the next chain packet. Therefore in a second iteration, the incoming packet has to be compared to the hash pointer values if no matching DMX value could be found.

2.2.7 Creating Child Feeds

Considering that every node has only its one ID Feed where it appends all data, this one feed could quite rapidly become large, confusing and impractical. To tackle this problem we add more functionality to the append-only-log concept. We add two packet types that are used to create new feeds from within an existing feed. The first packet type declares a new child feed and the second packet type is used in the newly created feed to affirm it is a child feed.

The advantage of having multiple feeds is not only to separate packets for different applications into different feeds, it also has the practical advantage of now being able to request packets for both feeds individually. Depending on the importance of the feeds we can now optimize the request by prioritizing certain feeds over others. [ADD REF] Creating a child feed increments the number of entries in the DMX table which has to carefully manage the want requests to not overload the network or to use too much computing power.

The process of creating a new child feed is similar to creating the ID feed. First a new key-pair has to be created where the public key is used as the feed name and the secret key

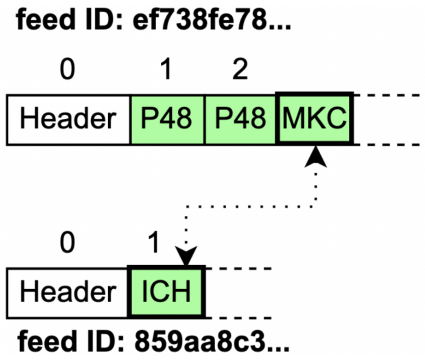

Figure 2.7: The packet MKC declares a new child feed and points to the address 859aa... The packet ICH in the new feed indicates that it is a child feed and has an upward pointer to its parent feed.

to sign packets. The node however now has to keep track of the new secret key as well. All key pairs that belong to the node's child feeds are stored in a config file and loaded into memory on start up. If the MKC (make child) packet reaches a node that replicates the feed, it will automatically create and replicate the child feed as well.

2.2.8 Creating Continuation Feeds

A feature with similar functionality is the creation of a continuation feed. The main idea behind this feature is to prevent feeds from becoming too long. There may exist feeds that get appended a lot of data and therefore grow at a rapid pace. When the feed gets bigger than a manageable size it becomes infeasible to propagate the whole feed through the network and store multiple copies of it on every device. For this use-case a packet type that declares a new feed as a continuation and a type that points to the previous feed are introduced.

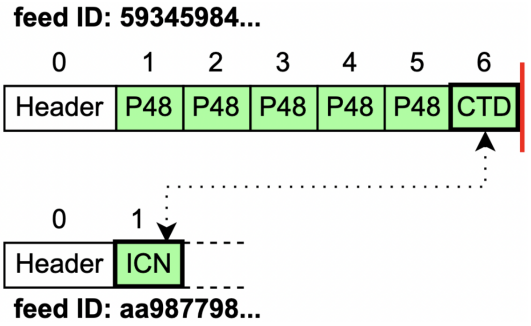

Figure 2.8: The packet CTD declares a new continuation feed and points to the address aa987... The packet ICN in the new feed indicates that it is a continuation feed and has an upward pointer to its previous feed. The red line indicates that no further packets can be appended.

After a continuation packet is appended to a feed, log entries can no longer be appended, thus new packets for said feed do not need to be requested anymore. However the node still has to respond to incoming want request for the feed to be able to further propagate

it through the network. The trust chain in this case is continued through the continuation packets. Using just continuation feeds as in Fig. 2.8 therefore does not free up storage space or come with any advantage, as still every node needs to build up the complete trust chain (now using two shorter separate instead of one longer feed). How continuation feeds can be useful will be discussed in [ADD REF TO SECTION]. As with child feeds, continuation feeds also have their own key-pairs that are stored in the same config file. If the address to the continuation feed is set to zero, this will be interpreted as ending the feed without continuation (No new feed will be created).

2.2.9 Packet Types

There are a number of different packet types in TinySSB, some of which were added on top while working on this thesis. The first packet which can also be seen in Fig. 2.3 describes a typical 48 bytes payload packet where the entire 48 bytes can be used for arbitrary data. The second packet seen in the same Figure is used to start a blob chain.

2.3 Pycom 4

3
Conclusion

This is a short conclusion on the thesis template documentation. If you have any comments or suggestions for improving the template, if you find any bugs or problems, please contact me.

Good luck with your thesis!

Bibliography

[1] Anders Carlsson, Ievgeniia Kuzminykh, Robin Franksson, and Alexander Liljegren. Measuring a lora network: Performance, possibilities and limitations. In Olga Galinina, Sergey Andreev, Sergey Balandin, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, pages 116–128, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01168-0.

[2] Jansen C. Liando, Amalinda Gamage, Agustinus W. Tengourtius, and Mo Li. Known and unknown facts of lora: Experiences from a large-scale measurement study. *ACM Trans. Sen. Netw.*, 15(2), feb 2019. ISSN 1550-4859. doi: 10.1145/3293534. URL https://doi.org/10.1145/3293534.

[3] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, ICN '19, page 1–11, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369701. doi: 10.1145/3357150.3357396. URL https://doi.org/10.1145/3357150.3357396.

A

Appendix

University of Basel

Faculty of Science

Declaration on Scientific Integrity

(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assesor: _____

Name Student: _____

Matriculation No.: _____

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Place, Date: _____ Student: _____

Will this work be published?

☐ No

☐ Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)

Publication as of: _____

Place, Date: _____ Student: _____

Place, Date: _____ Assessor: _____

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .