A Feed Bundle Protocol for Scuttlebutt

Bachelor Thesis

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

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

Jannik Jaberg
jannik.jaberg@unibas.ch
2017-054-370

02.07.2020

Acknowledgments

I would like to thank Prof. Dr. Christian Tschudin for giving me the opportunity to work together with him on this thesis. He supported me during the whole process of planning and developping this thesis with his valuable and constructive suggestions and guidance, especially in these special times. In addition, I thank Christopher Scherb and Claudio Marxer for supporting my work with essential feedback. Finally, I want to express my gratitude to my whole family and friends for supporting me in any part of this thesis.

Abstract

Mini recap of the whole thesis
- environement
- problem
- goal
- outcome

Table of Contents

1 Introduction

The world is constantly changing, so is the internet. At this very moment, a movement in networking research is clearly seen. The movement gets away from the well known and proven practices and mesurements of the centralized web and strives for novelty: Distribution. Away from centralised servers, classical routing and routing into a new peer to peer driven, destributed and decentralized web. Secure Scuttlebutt is exactly one of these novelties, which facinate with refreshingly diffrent approaches to solve common networking problems. Yet they are still in developement with a not so easy way ahead.

1.1 Secure Scuttlebutt

Secure Sucttlebutt (SSB), invented and created by Dominic Tarr in 2014, is peer to peer communication protocol. His intension to develop such a protocol was a unreliable internet connection on his sailboat. So he decided to write his own offline-friendly secure gossip protocol for social networking, because this is what you do.[1]

Differing to other technologies, Secure Scuttlebutt does not offer a self explanatory out of the box onboarding princip. Comparatively, in other software the user gets either suggestions (e.g. Instagram) or the connectivity and its management is directly given in the software (e.g. default gateway DHCP). In SSB, the user has to connect manually to pub via an invite code, which him or her has to obtain on a different channel than SSB.[2]

1.2 Motivation

As already described, it is problematic for new users to connect to the SSB world, hence a very interesting and promising problem to solve. Further SSB is a great, refreshingly new technologie with a lot of potential. At the moment, it is still in an experimental state and mostly used in pilot projects where the technology is connected to existing domains (social

[1] Quellen

[2] Invide Code - https://ssbc.github.io/scuttlebutt-protocol-guide/

network, git, databases etc.)[3] Yet I want to explore its potential in a more comercial manner and surrounding.

1.3 Goal

This thesis explores the role of intermediary ”connectivity providers” which sell connectivity e.g. to Google or Facebook, through a prototype implementation of a Feed Bundling Protocol. It is based on SSB but rather diffrent in many concepts. Introducing these intermediary participant, where you are connected on start up, will make the onboarding easier, since they will hold all information to create new connections - casually said: The guy who knows some guy which might can help. With later described so-called feed pairs, the ID centric information gathering in one single feed from SSB will be splitted appart. This results in less data in each dialog of two participants and allow bundling.

1.4 Outline

First a more detailed description on SSB, with focus on the concepts and problems connected to the Feed Bundling Protocol will be given. Then the newly created and adapted concepts as well as architecturial idea of the FBP with respect to SSB are presented. After that we take a closer look on the implemented code and how it is solved. The evaluation covers solved as well as newly generated problems with these aproached strategies and how they could eventuall be fixed. Finally the conclusion and future work that showed during the process.

[3] Quelle

2

Secure Scuttlebutt

More detailed overview about SSB. Feed distribution problem General onboarding problem Paralellism between pubs and ISPs.

2.1 Append Only Log

2.2 Feed Distribution

2.3 Onboarding

3 Concepts and Architecture

3.1 Prerequisite

As describet in the chapter Secure Scuttlebutt we have an ID centric single feed driven environement. This prerequisite changes from the beginning. Central idea of the Feed Bundling Protocol is to split up this ID centric environement into replicated feed pairs, where two participants hold at least one of such a pair. This pair contains the whole dialog between two nodes which have a contract together.

Look at it as a tin can phone from your childhood where you have two cans, strapped together for every friend you want to communicate with. You start with one corded phone to your best friend, the one you trust the most. In one you talk and the can 'saves' everything you say to it. on the other side your friend has the same but can only hear things out of one can and can only talk into the other. Having this, the dialog needs a way or language to express expectations or requests from either side to communicate with each other, where you can declare what you want from each other. After quiet some time it will eventually get boring only talking to this one friend. Luckily your friend is the coolest kid in school and knows everyone and even tells you who he knows. So you decide tho ask your friend if he could introduce you to his other friends. Having that we got the introduce mechanic which adapts exactly this social human behaviour. After your friend has introduced you to one of his friends you and your friend start to build the new tin can phone. But because you are too far away from each other you cannot just have a cord from one to the other. So your best friend is ready as always and helps you by passing to cord trough his house. This corresponds to the replication of the feeds over intermediary connectivity provider. Yet another problem occures, you are not the only one. after quiet some time your bestfriend has so many strings in his house from al his friends which want to talk to that other friend. he repeats all the phones there so he has an enourmous amount of strings going to that other friend. SO he decides to combine all these strings into one and send all messages trough this single one with the information into which can it should come at the end. He multiplexes. Given that little story we can derive concepts and architecture for the tin can phones of the future.

3.2 Contracts

Deriving from that little story we see the base in the friendship. the friendship between you and your best friend and the friend ship between your bestriend and his other friend. So first and foremost of the whole connectivity, protocol and bundling, lay the business contracts between the nodes. Throughout the whole process of the these contracts and relationships between each participant got questioned and modified, since this is the most important aspect or basic building block for the whole thesis.

3.2.1 Client-ISP Contract

To build the tin can phone three basic identifiers are needed. First of all you have to trust each other. This accords to the whole juristical contract between the to parties. Next you need to know your names to lable the phone, so you know to who you talk. This refers to the public keys. Since everybody in your house can use the tin phone you also need some sort of secret so your friend know you send the message not your mom. This refers to the private key. Having that you need your two tin cans and two wires. One can stands for one feed and the wire for the replication. But if you do not know the address of your mate you do not know where to put that wire, hence you need that too. The address, as the name is well chosen, refers to the IP-address. To keep things a little easier we perform everything on a localhost in the file system so the address corresponds to a path or location to or of a folder. Last but not least, to distinguish all the cans you lable them accordingly. This leads to a contract like that.

Client-Contract	value	ISP-contract	value
actual public key:	cli001	actual public key:	isp001
actual private key:	******	actual private key:	******
actual feed ID:	cli001_isp001	actual feed ID:	isp001_cli001
ISP public key	isp001	Client public key:	cli001
ISP feed ID	isp001_cli001	Client feed ID:	cli001_isp001
ISP location:	./isp001/	Client location:	./cli001/

3.2.2 ISP-Server Contract

Since your best friend has an other friend the contract is the same as in the Client-ISP Contract.

ISP-contract	value	Server-Contract	value
actual public key:	isp001	actual public key:	ser001
actual private key:	******	actual private key:	******
actual feed ID:	isp001_ser001	actual feed ID:	ser001_isp001
Server public key:	ser001	ISP public key:	isp001
Server feed ID:	ser001_isp001	ISP feed ID:	isp001_ser001

Clearly seen, that client and server are indirectly 'connected' over isp001, which refers to your best friend. Initial idea of the thesis was a peer to peer Internet Connectivity Provider (ICP) network where the ISP-Company distributes the feeds internally between the ICPs. So in real life there is a contract the ISP e.g. Swisscom and the keys refer to connectivity provider stations or nodes within the ISP newtork of ICPs. Which means cli001 has a connection with icp342 and server has a contract with icp903. But both have a contract with Swisscom, which has internal routing to pass information from icp342 to icp903 - but more in the Outlook section.

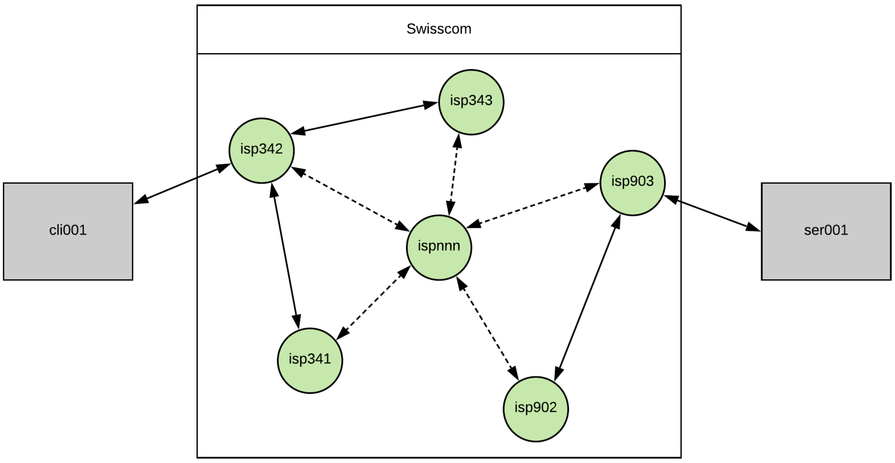


Figure 3.1: A simplified contract network.

Since now a contract is established, we need to know what happens in the tin cans and the wire. This leads to the replicated feeds.

3.3 Replicated Feeds

From the SSB API the following sentence can be exctracted: "A feed is a signed append-only sequence of messages. Each identity has exactly one feed.[4]" So what do these properties mean?

- signed message: Signing a message means the encryption of plain text with the sender's private key to a cipher text. The crypto text can be deciphered with the sender's public key.
- append-only: Under append-only we understand that this sequence can not be forged. So there is no possible way to modify or delete any entries that were appended at any time[5]. This append-only property is realzied with a hash chain which referenzes the hash value of the perviously generated message.[6]
- identity - one feed: Meant here is the perviously mentioned ID centric architecture. Per identity (key) is only a single feed mapped, where every, and I mean every, information you created or used in the SSB universe is stored.

Simplified a schema can be pictured, in the SSB feed is much going on but an adapted simplified version is more than sufficient for this thesis.

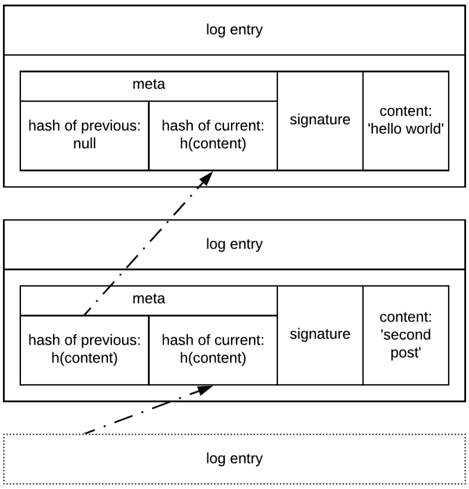


Figure 3.2: A schematic simplified feed.

So what can be derived from these information? Signing ensures that you can trust an identity. The append-only property underlines this trust by guaranteeing completness of the information read in a feed. ID centric feeds ensure that this feed belongs to exactly one identity but there is the sticking point. Since the replication of the SSB protocol always replicate the whole feed to all peers (hops noch angeben) of a single identity there is a

[4] https://scuttlebot.io/more/protocols/secure-scuttlebutt.html
[5] Feeds - https://ssbc.github.io/scuttlebutt-protocol-guide/
[6] Feeds - https://ssbc.github.io/scuttlebutt-protocol-guide/

massive load on the wire for big feeds. This causes latency and long scuttling time (feed update).
By splitting the feeds into smaller feeds this can be bypassed and the effective communication between two parties bundled in the already mentioned feed pairs. Resulting we have a schema like this: For the skae of clarity only the situation between client and ISP is given.

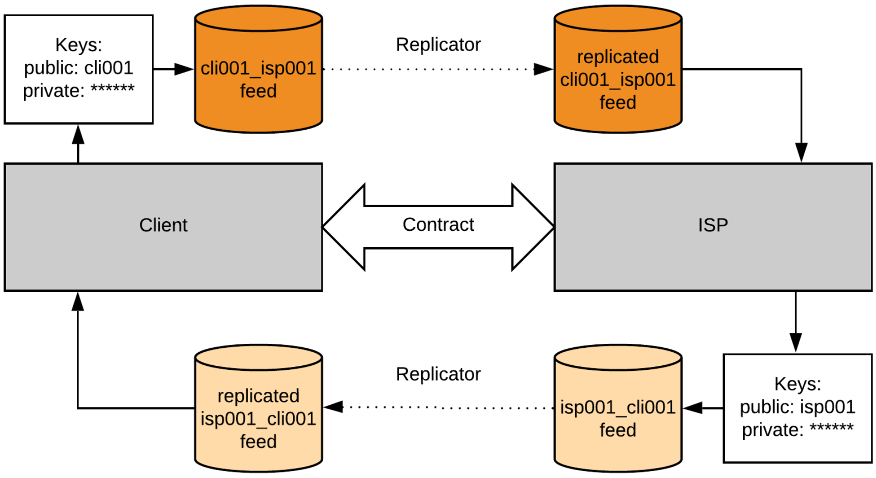


Figure 3.3: Full contract between client and ISP with feeds.

since for server and isp it is the same. As you can see. The replicator or the replication process has it first apperance. As concept, there is some sort of replicator instance or procedure that replicates the feeds to the corresponding address or location. We have a closer look in the implementation.

Having this setup the next step is to have a possibility to communicate, so the client can request information from the isps real database

3.4 Remote Procedure Call

General explanaiton of RPC.

3.4.1 Send Request

Format of request and api of method

3.4.2 Read Request

Format of request and api of method

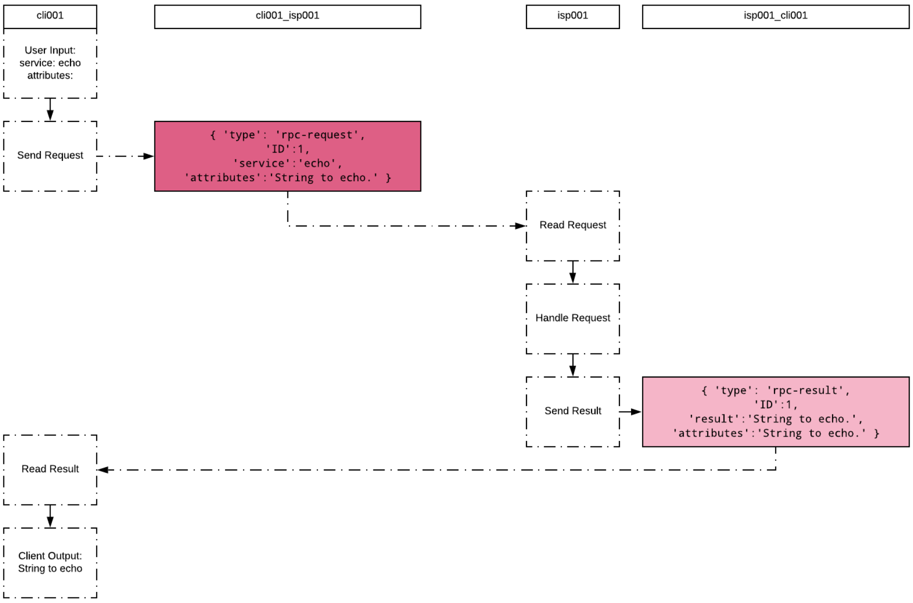
3.4.3 Send Result

Format of request and api of method

3.4.4 Read Result

Format of request and api of method

Resulting schema of the API and descriptions given above:



```
{ 'type': 'rpc-request',
'ID':1,
'service':'echo',
'attributes':'String to echo.' }
```

```
{ 'type': 'rpc-result',
'ID':1,
'result':'String to echo.',
'attributes':'String to echo.' }
```

Figure 3.4: A Remote Procedure Call by cli001 to isp001.

Picture description: -¿ will be splited into above sections

Seen in the picture above, client cli001 makes a request of the echo service with the attribute: String to echo. This is passed to the send_request function which assigns an unique ID for this request, resulting from the already given ids in the feed pair. Also merges type, id, service, and attributes into a request (a defined datastructure), writes appends this as a new log entry to the feed and saves the ID for keeping track of open requests. isp001 now detects a change on the feed cli001_isp001 and invokes the read_request method which takes the request appart and evaluates the service. Now the request gets handled and the wanted value returned. After that send result is invoked. It is similar to the send request, difference is the type and the result is also appended to the result datastructure. Now again, writing a new log entry to the feed with the given content. cli001 is notified that the isp001_cli001 feed is changed. so read result is invoked and the result gets either to the program, where it was requested or printed to the client.

3.5 Introducing and Detrucing

3.5.1 Introducing

Recaping the tin can phone story: The idea of introducing is to get in contact with a new friend, where your bestfriend introduces you to him. You and your new friend handcraft a new tin can phone. Since the cord only is as long as it reaches your bestfriend, he connects you to your newly aquired friend.

So the general idea of introducing in context of the feed bundling protocol is onboarding to a new server over your ISP. This approach differs from the commonly spread pubublish and subscribe (pub/sub) architecture. Whereas the server has no choice to decline a client in the pub sub model this is a foundation of the introduce detruce model. More technical General Idea - onboarding to a server. Instead of following server - introduce to server. differs from pub sub pattern in the way that the server has the choice to accept or decline a client. As we were talking of rpc before, in either way accept or decline an answer is provided to the client, else it would violate the rpc clauses.

So lets tank about the procedure in more detailed way: cli001 sends a rpc request to the introduce service of his ISP. this request needs an attribute which specifies the server the client wants to be intrduced to. In this particualr case ser001. isp001 invokes the introduce service, which now makes rpc request with information about client and the fact, that it wants to introduce itself to ser001. and sends this to the server. Now the server has the coice to either accept or decline the introduce inquiry. If ser001 accepts the introduce it will directly create the feed pair on his side (the two tin cans). afterwards it sends a confirmation or acceptance back to the ISP. Additionally to just the statement that the client was accepted the whole contract information for client is given by server: feed ID etc. Or the server declines the introducing approach, so the result is rejection followed by no or some sort of empty contract.

Either way isp001 gets the result and passes this result to the initial rpc request from the client. The client now gets his result. According to the state of acceptance or rejection it builds his feed pair in accordance with contract. finally the connection is established. Now if client wants to use a service from server it only writes the request in the corresponding feed and the procedure is the same as described in RPC Section.

An important addition, only client can introduce itself. The server has no knowledge about clients and also no way to get knowledge about clients, so only client can ask server for a contract.

3.5.2 Detrucing

Detrucing as a newly invented word by me, since normally after you introduced yourself to a person there is no way to make this unhappened. It acts the same for unfollow in a pub sub domain. But in contrary to the introduce both parts of the contract can detruce.
precisely either client or server can send a rpc request to the ISP service detruce which gets propagated to the opponent descibed above in the introducing part and results in terminating the whole contract. Result of this action is deleting keys and feeds. There is no way to decline a detruce service request.
Also here an important addition, after detrucing from either side, the client can yet again introduce itself to server.
Taken these descriptions a new schema of the network derives.

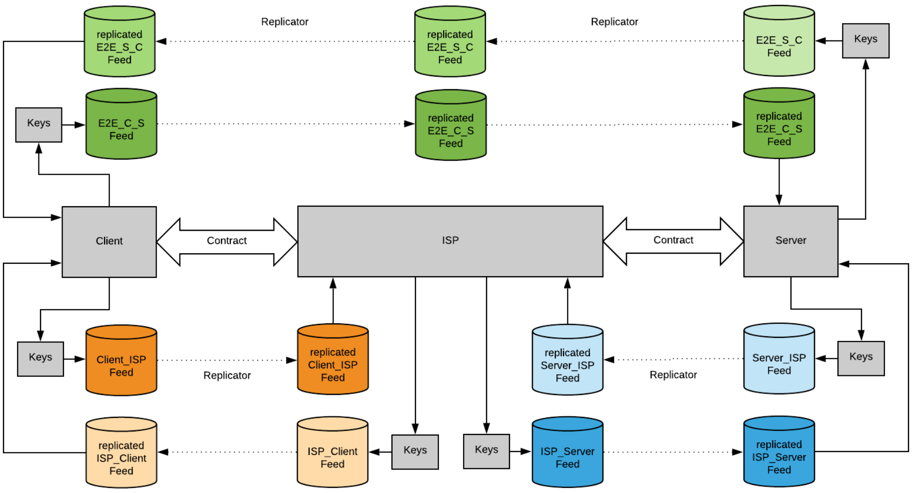


Figure 3.5: State after accepted introduce from cli001 to ser001

3.6 Bundling

taking again a look at the real world problem the isp has arbitrary many clients and many of them want to cumminicate with the same server. Idea: bundling all e2e feeds to same server into isp-server feed. -¿ load on wire problem

3.6.1 Adapted Introducing and Detrucing

introducing to server same - after server accept server generates both feeds request and result feed - sends contract to isp and isp generates result feed and replicates to cli - finally client generates request feed and replicates to isp.

3.6.2 Multiplexing and Demultiplexing

requests from client same way to isp. but isp does not replicate feed anymore. it takes whole log entry, signed by client and mulutiplexes into a new log entry with a mux type content and signs this - mutliplexing. at server, server takes this mux type and extracts whole request feed entry and appends bytes to request feed from client - demultiplexing. handles request and writes result to result feed. from there the whole entry gets again multiplexed in the ispser feed. at isp isp demux entry and also appends bytewise to result feed. replicates to cli and cli can use result.
Resulting Schema:

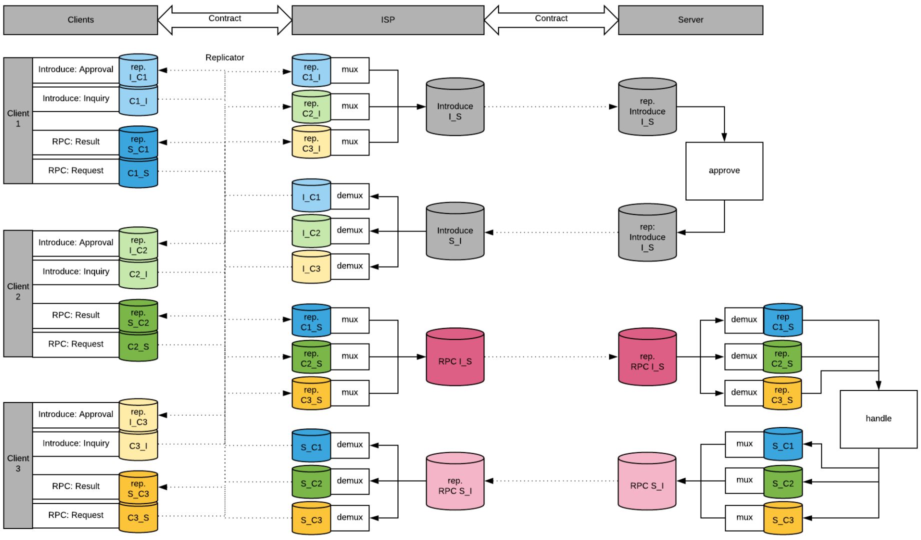


Figure 3.6: multiplexing

3.7 Outlook

3.7.1 P2P ISP Nodes

3.7.2 Contract between ISPs

4 Implementation

4.1 Feeds

Given by Prof. Tschudin explain structure

4.2 RPC

API with all methods in context of feeds and peers Adapted from RPyC so in the service class all services can be defined and have no impact on

4.3 Introducing

how is it implemented

4.4 Replication

Replicator Class that is given to each 'feed' so every time on this feed is operated by wr replicate to predefined location

4.5 Multiplexing

mux package idea is to only pass through the request then write in 'replication' feed and work from there. also for answering channel

5
Evaluation

5.0.1 Indexing - Keep Track of Progress

using sequenze numbers for keeping track and ID in entries is only

5.0.2 Stability - Robustness

6
Conclusion and Future Work

7 Body of the Thesis

This is the body of the thesis.

7.1 Structure

7.1.1 Sub-Section

7.1.1.1 Sub-Sub-Section

Paragraph

Even Sub-Paragraph This is the body text. Make sure that when you reference anything you use labels and references. When you refer to anything, you normally capitalise the type of object you reference to, e.g. Section 7.1 instead of section 7.1. You may also just use the `cref` command and it will generate the label, e.g., for Section 7.1, we did not specify the word "Section".
Hint: Try to structure your labels as it is done with `sec:my-label` and `fig:machine`, etc.

7.2 Equations

A Turing Machine is a 7-Tuple:

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle \tag{7.1}$$

A Turing Machine is a 7-Tuple even if defined in the text, as in $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$.

7.3 Tables

Some tables can also be used as shown in Table 7.1[7]. Remember that tables might be positioned elsewhere in the document. You can force positioning by putting a `ht!` in the definition.

[7] Table captions are normally above the table.

Table 7.1: Frequency of Paper Citations. By the way: Make sure to put the label always after the caption, otherwise LaTeX might reference wrongly!

Title	f	Comments
The chemical basis of morphogenesis	7327	
On computable numbers, with an application to the ...	6347	Turing Machine
Computing machinery and intelligence	6130	

7.4 Figures

Figures are nice to show concepts visually. For organising well your thesis, put all figures in the Figures folder. Figure 7.1 shows how to insert an image into your document. Figure 7.2 references a figure with multiple sub-figures.

Missing: Description figure.

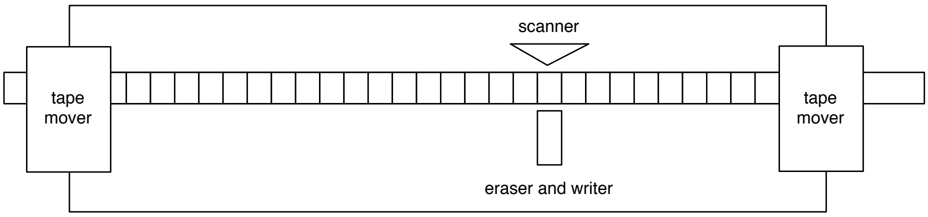
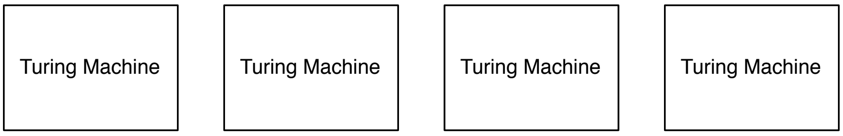


Figure 7.1: A Turing machine.



(a) Turing Machine 1 (b) Turing Machine 2 (c) Turing Machine 3 (d) Turing Machine 4

Figure 7.2: Plots of four Turing machines

7.5 Packages

These packages might be helpful for writing your thesis:

caption to adjust the look of your captions

glossaries for creating glossaries (also list of symbols)

makeidx for indexes and the back of your document

algorithm, algorithmicx, algpseudocode for adding algorithms to your document

8 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

A
Appendix

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Jannik Jaberg

Matriculation number — Matrikelnummer
2017-054-370

Title of work — Titel der Arbeit
A Feed Bundle Protocol for Scuttlebutt

Type of work — Typ der Arbeit
Bachelor Thesis

Declaration — Erklärung
I hereby 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.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 02.07.2020

Signature — Unterschrift