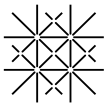


University of Basel

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 with him on this thesis. He has supported me during the entire process of planning and developing this thesis and has given me valuable and constructive suggestions and guidance, especially in light of this unique Covid-crisis time. In addition, I would like to 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 so many ways during the creation of this thesis.

Abstract

Aspiring new technologies emerge every day, one of which is Secure Scuttlebutt. Secure Scuttlebutt is a peer-to-peer communication protocol based on ID-centric append-only logs[1]. The aim of this thesis is to take mechanics from Secure Scuttlebutt and bring it in a more commercial environement by introducing new intermediary service providers (ISP) which sell connectivity to servers. Having a contract with such an ISP makes the initial onboarding much easier than in SBB.
By splitting up the ID-centric feeds into feed pairs for every connection, informations on the specific dialogs get bundled and stored independantly. since this is the smallest abstraction it allows again a form of bundling by multiplexing log entries together again into bigger feeds. So the problem of the imense replication work done by SSB is approached differently.

[1] Quelle

Table of Contents

1 Introduction

The world is constantly changing and so is the internet. At this very moment, a revolution in networking research is taking shape. This movement is leading away from well-known, proven practices and measurements of the centralized web and strives for novelty: distribution. The direction is away from centralised servers and classical routing and moving towards routing into a new peer-to-peer-driven, distributed and decentralized web. Secure Scuttlebutt is exactly one of these new platforms/apps/developments, which captivate with refreshingly different approaches to solving common networking problems. Yet they are still in development and have a future that is anything but sure.

1.1 Secure Scuttlebutt

Scuttlebutt (SSB), invented and created by Dominic Tarr in 2014, is a peer-to-peer communication protocol. His motivation to develop such a protocol was an unreliable internet connection on his sailboat and the result was his own offline-friendly secure gossip protocol for social networking. [2]

Differing from other technologies, Secure Scuttlebutt does not offer a self-explanatory out of the box onboarding principle. In other software, the user typically receives suggestions (e.g. Instagram) or connectivity and management are built into the software (e.g. default gateway DHCP). In SSB, the user has to connect manually to a hub via an invite code, which they must obtain on a channel other than SSB.[3]

1.2 Motivation

However, it is problematic for new users to connect to the SSB world, hence a very interesting and promising problem to solve has presented itself. SSB is a promising, inovative new technologie that has a great deal of potential. At the moment, it is still in an experimental

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

state and used primarily in pilot projects where the technology is connected to existing domains (social network, git, databases etc.)[4] I would like to explore its potential in a more commercial manner and environment.

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 also differs in many concepts. Introducing these intermediary participants, where you are connected on start up, will make the onboarding easier, since they will hold all information to create new connections. In plain English: *It's a guy who knows another guy who can help.* With "feed pairs", which are described later in this thesis, the ID-centric information gathering into one single feed from SSB is split into parts. This results in less data in each dialog of two participants and allows bundling.

1.4 Outline

First a more detailed description of 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 the architectural idea of the FBP with respect to SSB, will be presented. Subesequently I will take a closer look at the implemented code and how it is solved. This evaluation covers previously solved issues, as well as newly generated problems with the approach to these strategies and how they might possibly be solved. Finally, I will present a conclusion and highlight future challenges that discovered during the process.

[4] Quelle

2
Related Work

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

3

Concepts and Architecture

As described in the chapter Related Work, SSB is an ID-centric single feed driven environment, where onboarding is challenging. This prerequisite changes from the beginning. The basic idea of the Feed Bundling Protocol is to split up this ID-centric environment 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 with each other. Similar to the tin can phone from your childhood where you had two cans connected for every friend you want to communicate with. By introducing intermediary service providers, the onboarding happens at contract signing. Clients will have to possibility to connect to new servers via this ISP and create for each server a new feed pair, which is replicated over this very ISP. Since this approach means an enormous amount of feed replications between ISP and server, this feeds get bundled again.

3.1 Tin Can Analogy

This announced system seems very hard to understand but we can simplifiy it. Look at it as a tin can phone from your childhood where on either side 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, from the other can you can only hear things from your friend, it also saves everything said from your friend. On the other side your friend has the same but can only hear things out of the one can attached to your speaking can and can only talk into the other can, which is connected to your hearing can. This are the replicated feed pairs.

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. Leading to the simplified RPC protocol.

After a while it gets boring only talking to this one friend. Luckily, your friend is the coolest kid in school and knows everyone and even tells you about everyone he knows. Then you ask your friend if he could introduce you to his other friends. This introduction process

closely simulates real, human social behavior.

After your friend has introduced you to one of his friends you and your friend start to build a 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 allows you to route the cord through his house. This corresponds to the replication of the feeds over an intermediary connectivity provider.

But there is another problem: you are not the only one. After a while, your best friend has so many connections running through his house from all his friends who want to talk to their other friends, that here is an enormous number of strings going to that other friend. The solution is to combine all these strings into one and send all messages through this one, single bundled connection with the information into which tin can it comes 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

Given that little story, we see the foundation of the friendship. The friendship between you and your best friend and the friendship between your best friend and his other friend. By the same token, it is the business contracts between the nodes that provide the foundation for the entire connectivity, protocol and bundling. These contracts are to most important building block of the whole thesis since the define the behaviour of the replicated feed pairs, onboarding mechanics and bundling.

3.2.1 Contract Values

To build the tin can phone, three basic identifiers are needed. First of all you have to trust each other. This corresponds to the whole legal contract between the two parties. Next you need to know your names to label the phone, so you know who you are talking to. These are the public keys.

Since everybody in your house can use the tin phone, you also need some sort of code so that your friend knows that it is you who is sending the message and not your mother. These are the private keys. Having tha,t 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 your friend's address, you do not know where to put that wire, so you need that as well.
The address, as the name is well chosen, refers to the IP-address. Last but not least, to distinguish all the cans, you label them accordingly.

Therefore a contract consists of actual public key, actual private key, actual feed ID and the peers public key, feed ID and location. Since a contract has been established, we need to know what happens in the tin cans and the wire. This leads to the replicated feeds.

3.3 Replicated Feeds

The following sentence can be exctracted from the SSB API: "A feed is a signed append-only sequence of messages. Each identity has exactly one feed.[5]" So what do these properties mean?

- igned message: 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: this sequence can not be forged. So there is no possible way to modify or delete any entries that were appended at any time.[6] This append-only property is realized with a hash chain which references the hash value of the previously generated message.[7]
- dentity - one feed: the previously mentioned ID-centric architecture. Only one identity (key) is mapped feed, where every single bit of information you created or used in the SSB universe is stored.

A simplified diagram can be user to illustrate that there is a lot going on in the SSB feed, but an adapted simplified version is more than sufficient for this thesis.

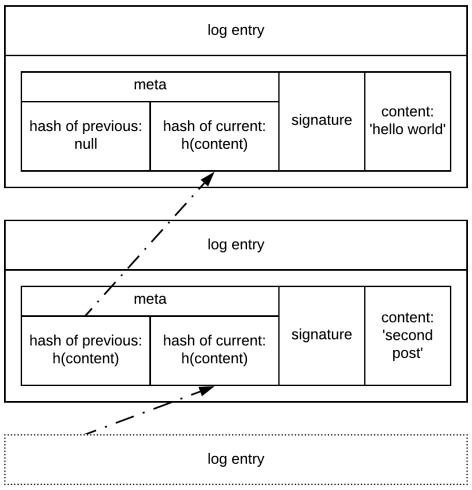


Figure 3.1: A schematic simplified feed.

What can be derived from this information? Signing ensures that you can trust an identity. The append-only property underlines this trust by guaranteeing completeness 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 replicates the whole feed to all peers (hops noch angeben) of a single identity, there is a load on the wire for big feeds. This causes latency and long scuttling time (feed update). By splitting

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

the feeds into smaller ones, this can be bypassed and the effective communication between two parties bundled in the feed pairs mentioned previously. As a result, we have a diagram like this: For the sake of clarity, only the situation between the client and the ISP is shown.

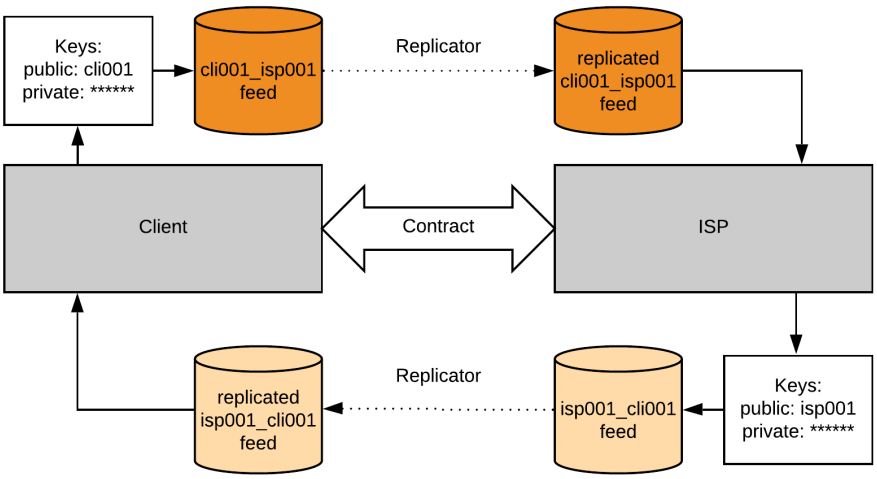


Figure 3.2: Full contract between client and ISP with feeds as it is the same for the server and the ISP as shown.

The replicator or the replication process has it first appearance. As a concept, there is some sort of replicator instance or procedure that replicates the feeds to the corresponding address or location. But let's have a closer look at the implementation part.
Having this setup, the next step is to have a possibility to communicate, so the client can request information from the ISP's real database.

3.4 Remote Procedure Call

General explanaiton of RPC.
Resulting schema of the API and descriptions given above:

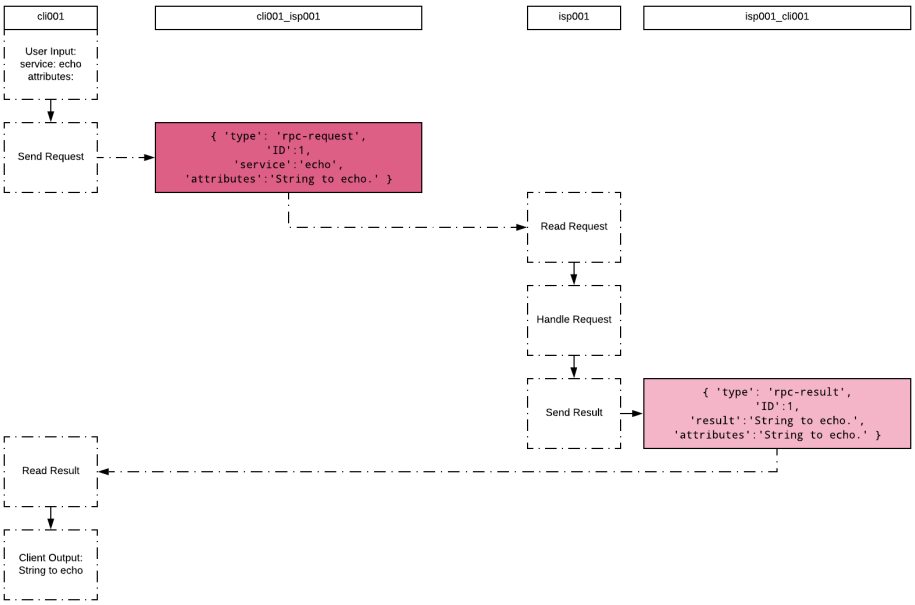


Figure 3.3: A Remote Procedure Call by cli001 to isp001.

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

Recapping the tin can phone story: The idea of introducing is to get in touch with a new friend, to whom your best friend introduces you. You and your new friend create a new tin can phone. Since the cord is only long enough to reach your best friend, he connects you to your newly acquired friend. Therefore, 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 common publish and subscribe (pub-sub) architecture. Where the server has no choice to decline a client in the pub-sub model, this is the foundation of the introduce-detruce model. 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.
A more detailed description: cli001 sends an RPC request to the introduce service of his ISP. This request needs an attribute which specifies the server which the client wants to be intrduced to. In this particular case ser001. isp001 invokes the introduce service, which now makes an RPC request with information about the client and the fact that it wants to introduce itself to ser001. and sends this to the server. The server has the choice to either accept or decline the introduce inquiry. If ser001 accepts the introduction, it will directly create the feed pair on its side of 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. The client now gets his result. Depending to the state of acceptance or rejection it builds its feed pair in accordance with the contract and finally the connection is successfully 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 distinction: only the client can introduce itself. The server has no knowledge of clients and also no way to acquire knowledge of clients, so only the client can ask the 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 as an unfollow in a pub-sub domain. But contrary, to the introduce both parts of the contract can detruce.
Either client or server can send an RPC request to the ISP service to detruce, which is propagated to the opposite end descibed above in the Introducing section and results in the terminaton of the whole contract. The result of this action is deleting keys and feeds. There is no way to decline a detruce service request.
An important note: after detrucing from either side, the client can yet again introduce itself to the server.
A new diagram of the network can be derived using these descriptions.

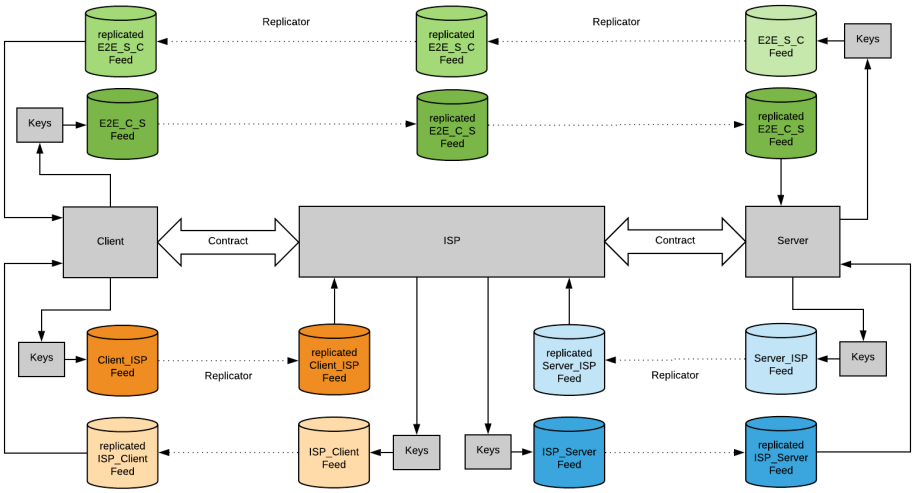


Figure 3.4: State after accepted introduce from cli001 to ser001

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:

3.7 Outlook

3.7.1 P2P ISP Nodes

3.7.2 Contract between ISPs

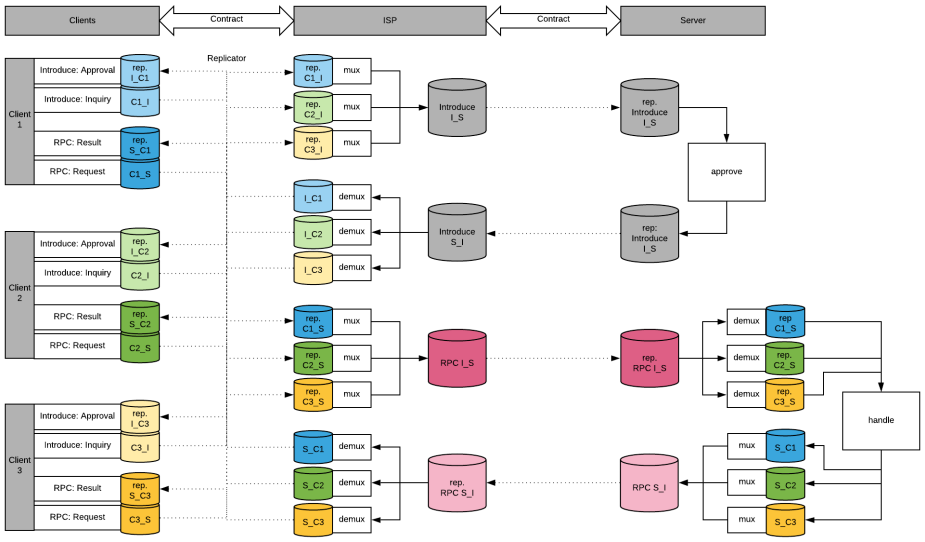


Figure 3.5: multiplexing

4

Implementation

4.1 Contracts

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/

4.1.1 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

4.2 Feeds

Given by Prof. Tschudin explain structure

4.3 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

4.3.1 Send Request

Format of request and api of method

4.3.2 Read Request

Format of request and api of method

4.3.3 Send Result

Format of request and api of method

4.3.4 Read Result

Format of request and api of method

4.4 Introducing

how is it implemented

4.5 Replication

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

4.6 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

6.1 Conclusion

The goal of this project was to introduce new intermediary service providers and replace the ID-centric append-only log from Secure Scuttlebutt with a feed bundle protocol. This extension or modification allows a much easier onboarding experience, since after signing a contract with an ISP the client is indirectly connected to all the ISP's servers. With the new introduce-detruce architecture, clients can connect and diconnect in a simple manner to new servers. Within this process new feed pairs are created, which bundle all the information for this specific connection. To ease the load on the wire and the process of replicationg every feed trough the ISP directly to the server, requests get multiplexed into a single feed pair between ISP and Server. Since this system is developed newly completely out of the blue there are many ways to improve it, one of which is to use the feeds properties more to define the state of doneness inside the single log entries. There are still many ways to improve and expanded the system with important key features to generate a relieable client-ISP-server network. Some of them are discussed within the next section.

6.2 Future Work

Apart from improving the general system, since it is not as solid as expected, we only ever looked at a single ISP with a single node to connect to with arbitrary many clients and servers connected. But in the Real world this is not the case. There are many ISPs on the market and they have connectivity stations all around their country. So internet service providers (ISPs) and internet connectivity providers (ICPs) can be taken appart. Also contracts between ISPs could be possible. In the process of creating the simplified version of the feed bundle protocol, always a look at the big picture was takens and decisions were made with respect or kept in mind to this great picture.

6.3 ISPs and ICPs

As said, the ISP was always a single node, with contracts to clients and servers. But we could look at the ISP as a company with a net of ICPs where the physiscal connection to the servers

and clients takes place. In simple terms, the client and server were indirectly 'connected' over the ISP. The initial concept of the thesis was a peer-to-peer Internet Connectivity Provider (ICP) network where the ISP-Company distributes the feeds internally between the ICPs. In real life there is a contract to the ISP, e.g. Swisscom, and this very ISP has connectivity provider stations or nodes within the ISP newtork of ICPs. This means that a client has a connection with icp342 of Swisscom and the server has a contract with icp903. But both have a contract with Swisscom, who provides internal routing to pass information from icp342 to icp903. Given this a new challenges emerge. How will the feeds be replicated?

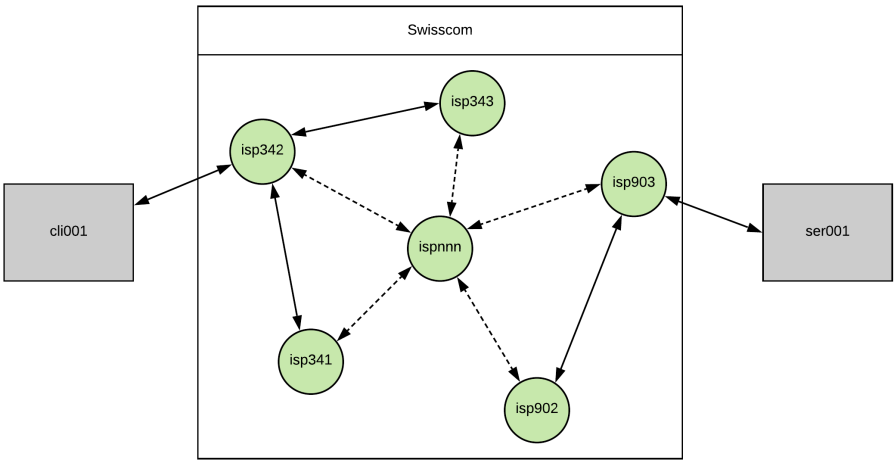


Figure 6.1: A simplified contract network.

Either with the same approach as now where every ICP node stores a replication of each feed pair which it routes to the next node or only the multiplexed log entries will be appended to the ICP-ICP feed pair. Also it should be possible for a client to change the connectivity provider. If we say when traveling from Basel to Zurich also the ICP changes since they refer to antennas in the cities. There need to be new algorithms which handle these exact usecases.

6.4 Contracts between ISPs

Yet again, we can take this distribution to a next level where ISPs have contracts with ISPs. This leads to a way to bypass that every ISP needs to have a contract to every server. This has a very special impact on the system since then new contracts are generated, where the business aspect is not defined. The problem there is easier to explain by an example. If we say Google has a contract with Swisscom and Sunrise wants to have a contract to Google. There will be some money involved. Let's say Google pays Swisscom a hundred swiss francs for every introduced client. So Google has now the opportunity to make a contract with Sunrise, but they do not like each other very much and had difficulties earlier. So Google offers Sunrise fifty swiss francs per connected client. So Sunrise probably will not sign, since they have a good relationship with Swisscom. So now Swisscom offers Sunrise

75 swiss francs for every client they provide to Google. Now you see the dilemma. In the first scenario Google would make more profit, where as the second scenario is better for Swisscom and Sunrise. These dynamics are not simple, hence very interesting and have to be clarified.

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[8]. Remember that tables might be positioned elsewhere in the document. You can force positioning by putting a `ht!` in the definition.

[8] 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.

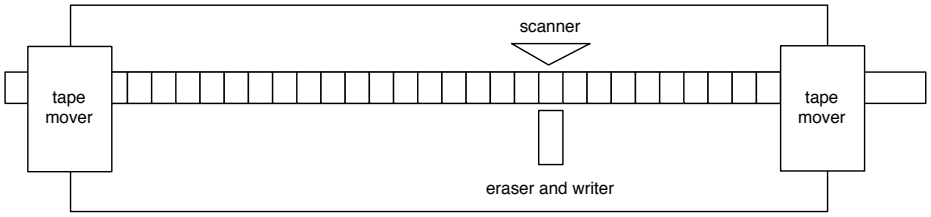
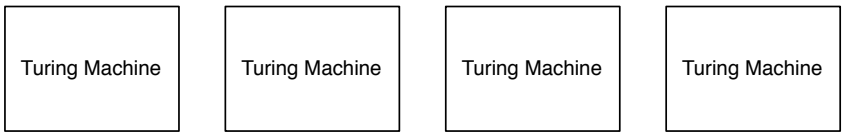


Figure 7.1: A Turing machine.



(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