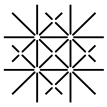


University of Basel

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 the mechanics from Secure Scuttlebutt and bring them to a more commercial environment by introducing new intermediary service providers (ISP) which offer 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, information on the specific dialogs gets bundled and stored independently. Since this is the smallest abstraction, it allows an additional form of bundling by multiplexing log entries together into larger feeds. Therfore the challenge of the immense 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

In this chapter we will see some of the key features which were taken account of to realise the feed bundle protocol. Befor taking a closer at the baseline for the protocol, which consists of parts from the Secure Scuttlebutt technologie and the Remote Procedure Call Protocol, we jump shortly into the blockchain and its properties, since by this very moment, everybody who reads this thesis will heard about.

2.1 Blockchain

The blockchain is well known as the foundation of the bitcoin. It has received an extensive attention in the recent years.
footnoteQuelle Zheng Xie etc 5 But what is the thing that makes the blockchain so impressiv and desired? The blockchain is often described as an immutable ledger which allows transactions to take place in a decentralised manner.
footnoteQuelle Zheng Xie etc 6 Exactly these key properties we also find in Secure Scuttlebutt.

2.2 Secure Scuttlebutt

Having the blockchain as a foundation and rather well know by the broad mass for an append-only log makes the jump into the the universe of Secure Scuttlebutt much easier. Secure Scuttlebutt (SSB) is a novel peer-to-peer event-sharing protocol and architecture for social apps.
footnoteTschudin Paper Aim of this section is to give a very high level overview about SSB, its ideas and properties, since the they are not quiet easy to understand.

2.2.1 Append-Only Log

2.2.2 Onboarding

2.3 Remote Procedure Call

Remote procedure calls, as the name implicates, are based on procedure calls but extended to provide for transfer of control and data across a communication network. There are two participants in the simplest manner, caller and callee. The caller wants to invoke a procedure with given parameters. The callee is the instance, which actually proceeds with the data and returns the result of that specific request. If an RPC is invoked, the the caller's environement is supended, all the information needed for the call transmitted through the network and received by the callee, where the actual procedure is executed with these exact parameters. The benefit of such an RPC-protocol is that the interfaces are designed in a way, that third parties only write the procedures and call exactly these procedures in the callers environement.Birrell and Nelson [1] This leads to a very promissing way to have a basic version of such an RPC-protocol for this thesis. Since it allows to invoke in the callers environement but are atually performend by a callee which returns the result back to the caller.Birrell and Nelson [1]

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.

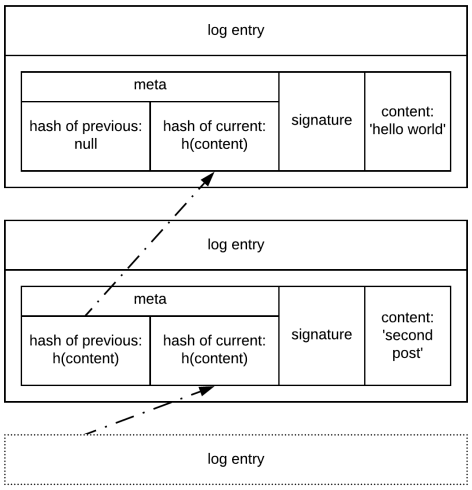


Figure 3.1: A schematic simplified feed.

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.

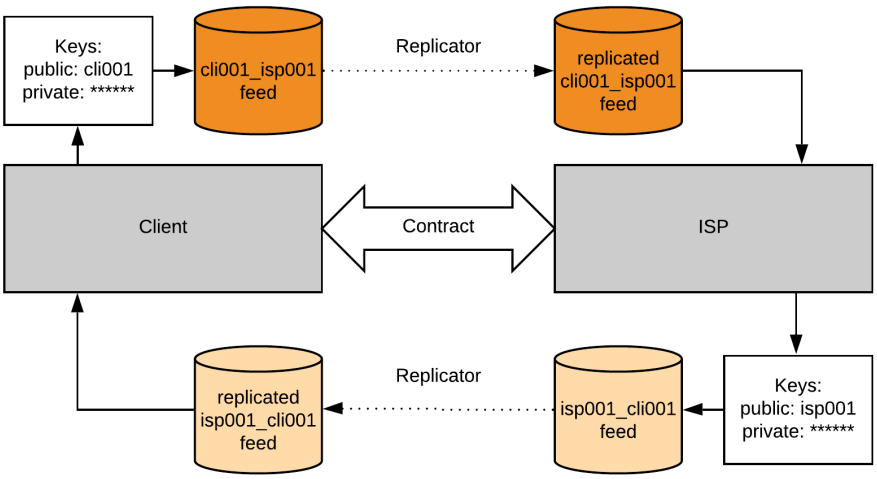


Figure 3.2: Full contract between client and ISP with feeds as it is the same for the server and the ISP as shown.

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

As explained in the section Related Work, Remote Procedure Calls are useful paradigm for providing communication across a network between programs.[8]. It faces many challenges, which are not important for this state of the developement of the feed bundle protocol. So the RPC used in this section is a very simplified version.
The Idea is to have a caller, in our case the client and a callee, the ISP or server. Having this kind of request–response protocol. An RPC is initiated by the caller, which sends a request to a callee to execute a specified procedure with given attributes. In our case these specified procedures are called services. By only having one such service e.g. the echo-service, which just echoes the attributes back to the caller we ensure the RPC-protocol works as defined. The, in the next section described, introducing and detrucing mechanics can also be summarized in such services, where the caller makes an RPC-request to the e.g. introduce-service with the needed parameters.

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

[8] Birrel, Nelson

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.

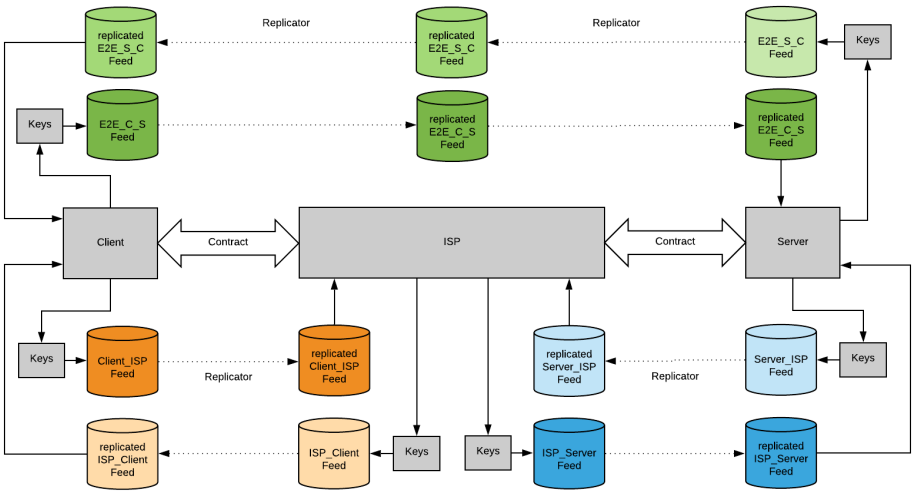


Figure 3.3: State after accepted introduce from cli001 to ser001

3.6 Bundling

Taking again a look at the real world problem, the ISP will have arbitrary many clients and many of them want to communicate with the same server. So instead of repeating each end to end feed pair over the ISP to the server, the new requests will be sent through a single feed pair between the ISP and server. This reduces the amount of replication work enormous.

3.6.1 Adapted Introducing and Detrucing

The introducing and detrucing idea stays the same, whereas the replication process is changed. After some server accepts a client, the server generates the whole feed pair. But instead of replicating to the ISP, nothing happens. To close the introduce request, the server sends the contract to the ISP and there the ISP generates the feed, which holds data from the server to the client. It is the same feed as in the server but it is not replicated over the general replication instance. Finally the client gets the result and generates the feed which holds the data from the client to the server. The feed pair for client-server communication gets normally replicated between client and ISP. Whereas between ISP and server a new way of replication is given.

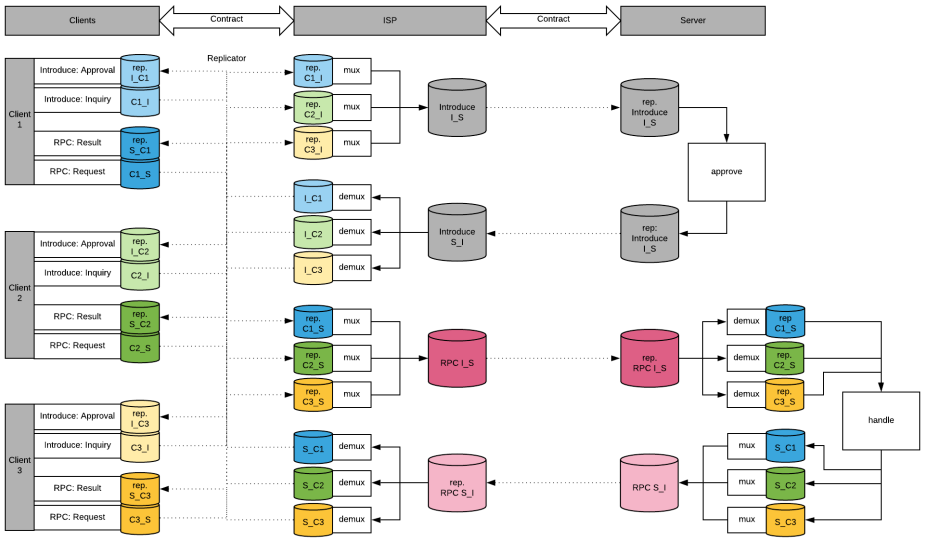


Figure 3.4: multiplexing

3.6.2 Multiplexing and Demultiplexing

So now we look at the communication between a client and a server. Requests from the client get transfered the same way to the ISP as before. Now instead of just forward the updated feed, by replicating to the server, the ISP detects new log entries and multiplexes these into a new log entry. More accurate, the ISP generates a log entry signed by itself, where the content of it is the whole log entry signed by the client. This log entry is written

to the ISP-server feed and replicated. The server detects the change on the ISP-server feed and takes this log entry. The multiplexed log entry, which belongs into a client-server feed is extracted and appended to the client-server feed. This step is called demultiplexing. From here we are again at the situation before. A change in the client-server feed is given and the request gets handled. The result is written to the server-client feed and again not replicated. From there the whole story is repeated. The log entry gets again multiplexed in the server-ISP feed. At the ISP the log entry is demultiplexed and appended to the server-client feed. From there it is replicated to the client and the client got its result for the request.
Schema von log entry in log entry

3.7 Outlook

Having all these concepts and architecture, we see the whole process is simplified on a single central ISP. In the real world this is not the case, hence as prove of concept it is more than enough. In the next steps the system has to be expanded by splitting this very ISP into a net of ICPs. They act as as effective connectivity stations and an architecture has to be found where clients and servers connect to these conectivity nodes. Also adding more ISPs, more companies is needed. The dynamic of contracts between ISPs has to be explored, but more in the Future Work section.

4 Implementation

In this chapter will be discussed, how a prototype implementation could be realised by applying the above outlined concepts and architectures. This implementation or better said pseudo code shall give a high level overview what can be derived from the concept into the software, by discussing the key elements.

4.1 Contracts

Implementing the contracts is a rather easy task on a prototype implementation than it would be for a real world application. As mentioned it consists of the knowledge about each other and where they are located.

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

In this table we can see a basic contract with all the information needed. This contract can still be broken down even more, since the feed-IDs are just appended public keys. Here a first abstraction to the real world application is made, the public and private keys would be curve 25559[9] key pairs. The terms here act as simplification and easier distinguishable keys. Having this setup with a curve 25559 key pair it is best practice to store them in a secrets or key file. So the most basic contract can look like this:

[9] Quelle

Client-Contract	value	ISP-contract	value
key file:	cli001.key	key file:	isp001.key
ISP public key	isp001	Client public key:	cli001
ISP location:	./isp001/	Client location:	./cli001/

This even more simplified base can be stored in some file and build the rest of the contract by the programm, but there needs to store it somehow.

4.1.1 ISP-Server Contract

remove

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 Replicated Feeds

This implementation was given by Prof. Dr. Tschudin. It is a very simplified version of an append-only log in the pcap format, generated from a curve 25559 key pair. Every log entry is signed by some private key, which leads to integrity but not security. The whole security part was left out during this thesis.

4.2.1 Structure

The Feed is a list of log entries. Each log entry consists of three main parts: meta data, the signature and the content. The meta holds information about the current log entry such as the feed ID, its sequence number, which is the internal possition in the feed of the log entry, a hash reference to the log entry before, its own hash value of the content for the next log entry. Next is the signature which signs the meta data. The content part is what is actually put into the log entry.
Since all the information is stored in the cbor2 format and the saved in a pcap file, the result is a binary array which hold important properties usful for the bundling. Either a new log entry can be written with a key or an existing log entry can be appended to the binary array without validation. This mechanism is a key feature for the bundling aspect. *BILD*

4.2.2 Replication

The replication mechanism gets invoked after each write operation to a feed. Generally speaking, this could be realised easily with TCP or UDP in a real network. In this basic implementation replicates feeds in the filesystem this was solved by just copy the feed to

the corresponding folder given in the contract.

4.3 RPC

Having the contract and replicated feeds, the type of RPC-protocol plays its turn. To communicate between two participants four general methodes are needed as listed below. By having a simple serialisable datastructure requests and results are generated. Requests call services which use the given attributes and produce a result, this connects to the given idea of the real RPC-protocol[10]

4.4 Datastructure

A suiting datastructure or format is a dictionary or a JSON-String, having keys that reference a field, as well as beeing serialisable. In this structure an ID has to be given to identify the request or result, a type has to be set to distinguish request and result, further the service to be called is needed as well as the attributes or the result of the call. This results in a minimal set of keys for request and result:
{'ID':0, 'type':'request', 'service':'echo', 'attributes':['An echo']}
{'ID':0, 'type':'result', 'result':'An echo'} Having the ID as identifier, caller can look up the request made and map the result to the call.

4.5 Services

Services are the procedures called by the caller and executed by the callee. In the feed bundle protocol there are some key services which have to be implmented:

- echo - It just returns the given attributes.
- get_service_catalog - the caller needs a list of all services the caller has available.
- introduce - This service passes a request to the server specified in the attributes and introduces the caller(client) to it and sign a contract.
- detruce - This closes an alread established contract and ereases all information built on it.
- get_servers - To call the introduce service a server is needed. Since the caller has no knowledge about servers, this is essential.

4.5.1 Send Request

pseudo method

[10] Birrel

4.5.2 Read Request

pseudo method

4.5.3 Send Result

pseudo method

4.5.4 Read Result

pseudo method

4.6 Introducing

how is it implemented

4.7 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

Evaluating a system, which takes baby steps into a completely new environement, was quiet challenging. Since most of the work was to come up with the concepts and architecture, the implementation is a little bit chaotic and often not exactly as the given concepts. Nevertheless, the code has been tested and important conclusions were drawn.

5.1 Testing Environment

As seen in the implementation part, the client can call services via requests from the ISP as well as from the server after introducing itself to it. Given this, the main testing aspect was to see that ISP and servers can keep up with the work load and distribute the results back to its origin. The test was as followed:
First there were three nodes involved, one client, one ISP and one server. After initialising all nodes the client went into a loop, where it first introduced itself to the server. The server automatically accepted this request and the feed-pair was created. After this, a random number was created between 5 and 20, which were the amount of service requests sent to this now connected server. To mimic a human interaction at first delays between the 1.0 and 4.0 seconds, randomly distributed with one decimal place, were implemented. This was the basic evaluation on functionality, performance, reliability and correctness.
Unfortunately after about 50 requests, either ISP or the server had an exception on the cbor2 library. Something with the bytestream seemed to be broken. Any other solution than just ignoring this exception and leave this specific log entry hanging could not be found. This resulted in open, unresolved request which the client waited for.
In a next step, after ignoring inconsitencies in the log entries, the system was tested to its limits, by having delays lowest at 0.1, adding more clients and up to a thousand iterations per client. Where at the last test with a delay of 0.1 seconds, 5 clients and a combined thousand requests per client, my personal machine nearly broke down. The tests lead to interesting results.

5.2 Results

5.2.1 Functionality

The functionallity was tested manually on different linux distributions[11][12] where as on Windwos and Mac OS heavy problems occured. The filesystem poller could not detect any changes on feeds, even they were made. As far as concerned and tested on linux distributions the functionallity is complete, the whole process of requesting services from the ISP as well as introducing to diffrent servers is given and works as intended. But only if the user follows strictly through the documentation how the system must be used. The user experiance is rather not intuitive and false use can lead the system to corrupt. This is founded in the very early stage of the project, by iterating over it these issues can be found and eliminated.

5.2.2 Performance

The performance begins with an astonishing speed and collapses over time. This fact was already know at the implementation point. To distinguish already handled and completed requests, the systen cycles through the whole feed, every time a change is detected. This problem can be solved by indexing the feed and saving the position, where everything already has been done. This factor has been left out intentionally to concentrate on the underlying concepts and architecture in the big picture.

5.2.3 Reliability and Correctness

As already teased in the section Testing Environement, some *undefined* or *undiscovered* fault between this implementation and the cbor2 library caused to ignore requests. Having this information, the tradeoff between the reliability and correctness of the feed bundle protocol lay on the hand. Having the underlaying simplified RPC protocol, the fact that request do not get a response violates the consensus if it is not exactly specified. All these findings generalise one big problem.

5.2.4 General Colaboration of Components

Again having this completle new technology, developed in just a few months, having the main focus on the general aspects results in a patchwork of diffrent libraries, common technologies and new technologies which are not coordinated on each other. To generally improve the very broadly open architecture and concepts need to be thighted with more rules and less freedom resulting in a more specific implementation where the components optimally are written to function well with each other.

[11] Ubuntu
[12] Arch

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 the client is indirectly connected to all the ISP's servers after signing a contract with an ISP. With the new introduce-detruce architecture, clients can connect and diconnect to new servers in a simple manner. Within this process new feed pairs are created, which bundle all the information for that specific connection. To ease the load on the wire and the process of replicating every feed through the ISP directly to the server, requests are multiplexed into a single feed pair between the ISP and the Server. Since this system is so new an has been developed completely "out of the blue", there are many ways to improve it, one of which is to use the feeds properties primarily to define the state of doneness inside the single log entries. There are still many avenues to be explored and important key features to be added to in order to generate relieable Client-ISP-Server Network, some of which are discussed within the next section.

6.2 Future Work

Apart from improving the general system, we have only examined the connection between a single ISP with a single connectivity node with a random number of clients and servers connected. In the real world, however, this is not the case. There are many ISPs on the market which have connectivity stations all over their respective countries. Therefore internet service providers (ISPs) and internet connectivity providers (ICPs) can be separated. Contracts between two ISPs would also be conceivable. In the process of creating the simplified version of the feed bundle protocol, we always kept the big picture in the foreground and decisions were made keeping this in mind.

6.3 Combination of Log Entries

Seen in the concept and the implementation, only a single new log entry is multiplexed into the ISP-server feed pair, resulting in a replication after each request. A diffrent approach can be made. We combine log entries in the multiplexing system. Meaning, instead of only one, a defined number of new log entries will be multiplexed to the server. This gives room for more efficient replication, since the whole feed gets replicated to the peer every time. Deriving from this the multiplexing feed pair can be splited up into arbitrary many sub feed pairs, linked to the priority of the request.

6.4 ISPs and ICPs

As mentioned previously, the ISP was always a single node, with contracts to clients and servers. However, we could also look at the ISP as a company with a network of ICPs where the physical connection between the servers and clients takes place. In simple terms, the client and server were indirectly 'connected' through the ISP. The initial concept of this 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 with the ISP, e.g. Swisscom, and this same ISP has connectivity provider stations or nodes within the ISP network of ICPs. This means that a client has a connection to icp342 of Swisscom and the server has a contract with icp903. But both have a contract with Swisscom, which provides internal routing to pass information from icp342 to icp903. In light of this fact, a

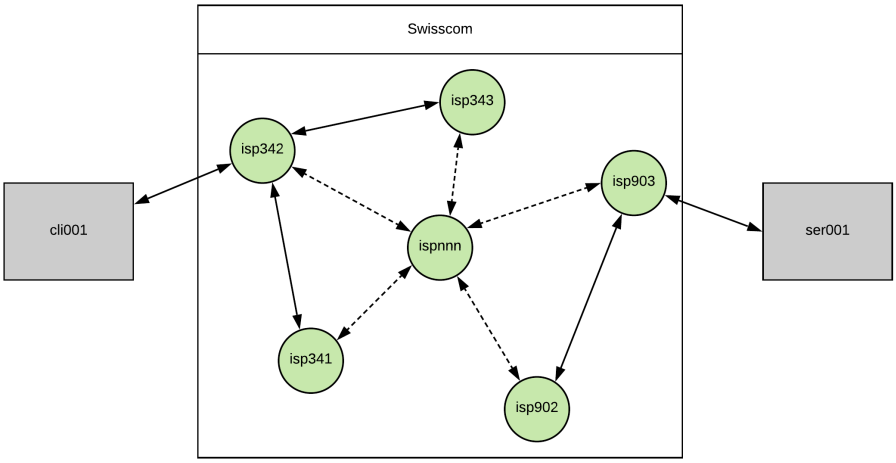


Figure 6.1: A simplified contract network.

new challenge emerges. How are the feeds replicated? Either with the same approach as is currently the case, where every ICP node stores a replication of each feed-pair which it routes to the next node, or only by appending the multiplexed log entries to the ICP-ICP feed-pair. Additionally, it should be possible for a client to change the connectivity provider, for example when traveling from Basel to Zurich. In this case, the ICP will change, since it routes to antennas in the respective cities. New algorithms need to be developed tho handle

exactly such use cases.

6.5 Contracts between ISPs

Yet again, we can take this distribution to the next level where ISPs have contracts with other ISPs. This provides a way to bypass the current requirement that each ISP must have a contract with each server. This has a very special impact on the system however, since new contracts are generated, when the business aspect hase not yet been defined. This problem is more easily explained with an example.
rewrite 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[13]. Remember that tables might be positioned elsewhere in the document. You can force positioning by putting a `ht!` in the definition.

[13] 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.

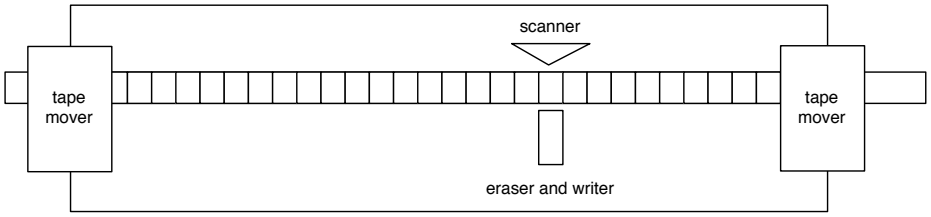
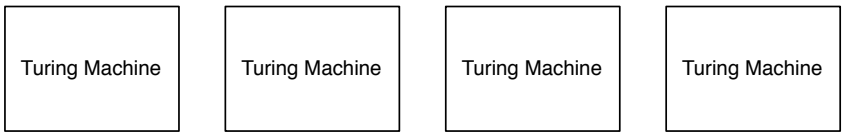


Figure 7.1: A Turing machine.



(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

[1] Andrew D Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59, 1984.

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