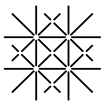

University of Basel

tinyISP: A Tunneling Negotiation and Feed Bundling Protocol Based on Secure Scuttlebutt

Master project

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

Examiner: Prof. Dr. Christian Tschudin

Jannick Heisch
jannick.heisch@unibas.ch
19-056-720

30.06.2023

Abstract

Today, almost all everyday applications rely on the Internet to collaborate and exchange information with others. The data is usually stored on central servers owned by large, individual companies. Furthermore, users who do not have access to the Internet infrastructure are excluded from using these applications. Decentralized protocols such as the Secure Scuttlebutt protocol offer an alternative. With this protocol, individual messages are replicated between participants using append-only logs. In cases where peer-to-peer connections are not available for data exchange, Pub servers are used. However, these servers are not selective and replicate all feeds of all clients, which can lead to scalability issues over time as the number of participants and published messages increases continuously.
Therefore, this project introduces the feed bundling and tunneling protocol called tinyISP, which allows clients to enter into a contract with an ISP that enables them to exchange data with other clients of the same ISP. The data from multiple feeds is bundled into one log and sent via the ISP to the receiving client, who then demultiplexes the data and assigns it to the corresponding feeds, reducing overhead in the replication layer. Once the contract between the ISP and the client is terminated, the feeds used for communication can be deleted to free up resources. Since the entire coordination of the protocol is based on append-only logs, it is independent of the Internet and can be transmitted through other media. A prototype implementation of tinyISP was integrated into existing Secure Scuttlebutt applications to demonstrate the benefits of such a protocol for everyday applications, such as a chat application.

Table of Contents

1 Introduction

Over decades, the Internet has revolutionized the way we communicate and connect with each other. It connects people across the world and enables us to communicate with friends and family and share information with virtually anyone. For most people, it is no longer possible to imagine their everyday lives without it.

Internet Service Providers (ISPs) provide the infrastructure and connectivity that is required for the Internet. They play a crucial role in facilitating the interconnection of millions of devices across the Internet. ISPs act as intermediaries between users and the vast network of interconnected computers and servers that make up the Internet, essentially providing the gateway to the Internet. Without them, the World Wide Web would fragment and users would no longer be able to access resources around the world.

However, as the Internet grew in popularity, concerns regarding privacy, security, and centralized control emerged. To address these issues, Secure Scuttlebutt (SSB) was developed as a decentralized peer-to-peer social network protocol that enables secure and private communication between users [2]. To ensure data ownership, privacy and control of users' own data, SSB does not rely on a central server. Users store their data in append-only logs, which are then replicated via peer-to-peer connections. To facilitate connectivity and onboarding in SSB, Pub servers are implemented that allow clients to find other peers and connect with them on the network. Clients can freely select which feeds they want to replicate from the Pub, while the Pubs are not selective and replicate all messages of all clients and offer them across the network. As the number of published messages increases, it can lead to storage problems. Furthermore, the Pub servers mainly replicate over the Internet, which limits access for those who do not have Internet access. Although the Internet is now taken for granted, only 66% of people have access to it [12]. Especially people who live far away from big cities lack the infrastructure for Internet access. Therefore, they are not able to replicate their data with other peers over a Pub server and are excluded from the dataflow, when no direct peer-to-peer connectivity is possible.

This project examines the feasibility of ISPs using SSB and append-only logs. As an intermediary service provider, clients can enter into a contract with the ISP that selectively replicates the feeds that are important for the contract and enables connectivity between the ISP's customers. The main goal of this project is to define and implement a protocol

for clients and ISPs to negotiate a contract and to facilitate the creation of tunnels between clients.
Because the entire protocol relies on append-only logs it is independent from the Internet and can be replicated via various ways, like LoRa, Bluetooth low energy and even via the exchange of hard drives.

1.1 Contribution

As part of the project, tinyISP, a protocol for creating and maintaining contracts between ISP and clients was developed, which completely relies on append-only feeds. Clients can use this protocol to request services from the ISP. After the client and the ISP have exchanged important parameters for contract fulfillment during the onboarding phase, the client can subscribe to other clients . The data between the clients is then tunneled through the ISP. To enable coordination between clients and the ISP, the control feeds previously negotiated in the onboarding phase are used. The data exchange between the client and the ISP is conducted via so-called data feeds. This allows feeds replicated between clients over the ISP to be multiplexed into a single feed, which is then demultiplexed at the other end and split back into individual feeds. As a result, the replication overhead is minimized.
To reduce the storage overhead, feed hopping is used to counteract the continuously growing data feeds. Similar to Pub servers in SSB, the ISP implemented in tinyISP improves connectivity between its clients. However, unlike Pub servers that replicate all feeds, ISPs can delete the feeds associated with the contract after it expires, resulting in improved scalability.
This protocol was then integrated into tinyTremola [3], an Android app that implements the SSB protocol, and into a Python implementation of tinySSB [10]. This allows a client to communicate with the ISP implemented in Python via the user-friendly graphical user interface of the Android App. Finally, the clients can subscribe to other users and chat with each other via the feeds tunneled by the ISP.

1.2 Outline

This report is structured as follows: First, an overview of the fundamental concepts and methods relevant to this project is presented. Then, the idea and specification of the tinyISP protocol are described. The next part focuses on the main features of the implementation of tinyISP in the Android app tinyTremola and the Python implementation of tinySSB. Finally, the results are summarized, and an outlook on future projects is given.

2
Background

2.1 Secure Scuttlebutt

In contrast to the increasingly centralized nature of the Internet today, the Secure Scuttlebutt protocol presents a decentralized alternative approach that was developed in 2014 by Dominic Tarr. The idea behind this protocol is to store data in append-only logs that are replicated between users, without the need of a central instance .

2.1.1 Append-only log

In Secure Scuttlebutt, each user is identified by the public key of his/her Ed25519 key pair. In order to share new messages, a user appends a new entry in his single-writer append-only log, also called feed. As the name suggests, this data structure allows only to append new content at the end of the log, deleting or modifying existing content is not possible. In order to maintain these properties, each entry appended to the log contains additional metadata: the timestamp of the creation of this message, the cryptographical hash of the previous log entry, the sequence number, the public key of the author, also called feed ID, and the signature of the message [4].

The cryptographical hash values of the previous log entries, also called backlinks, form a linked-list like data structure. With the signature of the message other users can verify, that the log entry was actually created by the feed author and that the content of the entry has not been subsequently modified. The backlink of the message is also signed by the author's private key, so that it becomes immutable. This ensures that the entries of a feed cannot be modified or deleted and that new entries can only be appended at the end of the append-only log.

If the entry has been subsequently modified, the signature cannot be verified by other peers, resulting in an invalid feed which will not be further replicated. Figure 2.1 shows an example of such a Secure Scuttlebutt feed.

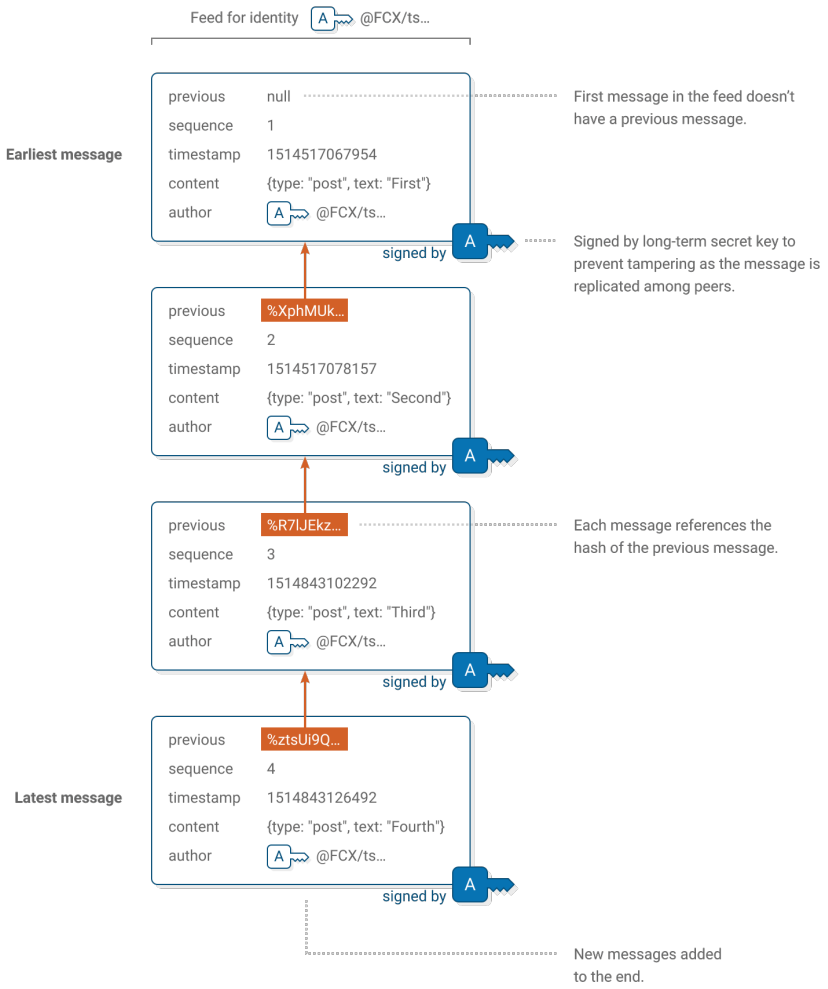

Figure 2.1: Example of a Secure Scuttlebutt feed [1].

2.1.2 Replication

To share feeds between users, Secure scuttlebutt is based on a gossip protocol, which enables data exchange via peer-to-peer connections. The SSB protocol follows the subjective reader principle: clients do not have to replicate all global available logs, but can decide which feeds are relevant to them. Users can follow others in order to replicate their feed, which forms a social graph as shown for example in Figure 2.2.

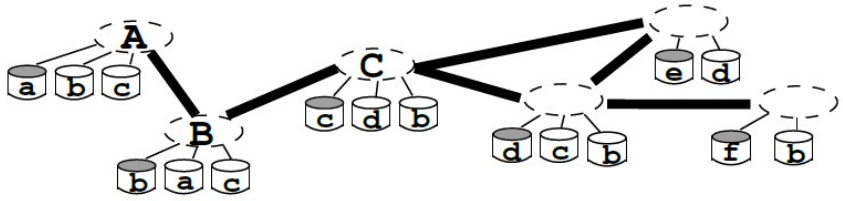

Figure 2.2: Social graph: A and B follow each other so A contains the log b. A does not have a log c, because A does not follow C. If A and C follow each other, A will receive the log c via B [4].

2.2 tinySSB

As previously described, in the classical Secure Scuttlebutt protocol, additional metadata is attached to each message in order to append the log entry to the correct position of the correct feed. By realizing that most of this additional information is already given by the trust of the previous log entry, tinySSB does not send all the metadata of the message [10]. In addition to the content of the message only the signature as metadata is sufficient, to reconstruct the other metadata based on the previously received log entries. This method significantly reduces the overhead per log entry.

The feed ID is implicitly given through the signature of the received log entry. It can be derived by trying out all trusted public keys, only the correct key is able to verify the signature. In addition, the sequence number and the hash of the previous message have to match the last received log entry of this feed, otherwise the message will not be considered. Even if not all metadata is sent along with the message, it must be taken into account when calculating the signature so that the recipient can verify the authenticity of the message. For this purpose, the recipient combines the content of the message (except the signature) with the implicitly given feed ID, the hash of the previous message, and the sequence number in order to check whether this matches the signature. If it does, the log entry can be appended to the corresponding feed.

Through this optimization, tinySSB messages are sent in packets of 120 bytes, of which 48 are used for the actual content. If more data needs to be sent in one entry, the content is divided into packets of 120 bytes, called chunks. The log entry then contains the cryptographic hash of the first chunk, and each chunk contains the hash of the following chunk. In this way, the messages are broken down into small packets that can be sent over connections without high data transmission rates like LoRa.

2.3 Grow-Only Set

A grow-only set is a data structure that belongs to the class of Conflict-Free Replicated Data Types (CRDTs). It is similar to a conventional set but only allows adding elements. Deleting elements is not possible.

Two grow-only sets can be merged using a merge operation, which is the union operation of both sets. Since the union of two sets is commutative and associative, the order of elements in the set does not matter. Furthermore, due to the idempotent nature of the

operation, applying unions multiple times yields the same result. Therefore, merging two sets is conflict-free, ensuring a single unique outcome [6].
These properties enable multiple replicas to concurrently add elements to the grow-only set and subsequently merge their set with that of another replica. Over time, this grow-only set converges to a consistent state across all replicas, without the need of additional forms of coordination common in other distributed systems.

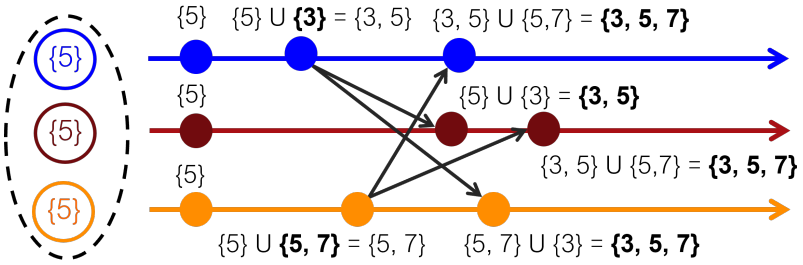

Figure 2.3: Example of a grow-only set exchanged between multiple replica [1].

3
Related Work

The concept of Secure Scuttlebutt is based on gossiping, whereby friends share their feeds via direct peer-to-peer connections. By doing so, they also get to know their friends' friends and can start sharing their feeds as well. In this concept, data flows between users who follow each other, thus forming a social graph.
However, if a connection cannot be established because the friends are not in physical proximity and are also not directly reachable via the Internet, e. g. due to Network Address Translation (NAT), one is excluded from the data flow. To solve this problem, pubs were introduced, which are essentially normal SSB identities that replicate all feeds. Users can connect to a Pub server and replicate the desired feeds from there. But the Pubs are not selective and have to store the feeds of all users, which causes a significant storage overhead and limits scalability. This approach is therefore only feasible for small communities.
To circumvent these limitations, SSB-Room was introduced [7]. Clients can connect to a room server via the Internet, which serves as an intermediary and establishes a tunneled connection between the clients. Unlike Pubs, in this case no feeds need to be stored on the server, as replication between the clients takes place via the established tunnels. But this approach also has disadvantages, as at least two clients must be online simultaneously for replication to occur. Additionally, communication with the room server is exclusively done via the Internet, which excludes people without Internet access.
Initial approaches for an ISP based on Secure Scuttlebutt and append-only logs were presented by Jannik Jaberg in his bachelor thesis in 2020 [5]. The focus was on an intermediary between a client and a server of a service provider like Google or Facebook, through which clients can send requests to a server via an ISP based on append-only logs.
In this project, the approach of Jaberg is generalized, with the ISP not mediating between a server and a client but between clients which are all equal. Furthermore, the protocol allows clients to request the service of an ISP in the onboarding phase and exchange the required information to conclude the contract. Afterwards, clients can establish a contract with other clients through the ISP as an intermediary, in which they agree on a shared pair of feeds for client-to-client communication. These feeds are multiplexed over the data feeds negotiated in the onboarding phase. The bundling of feeds relieves the replication layer as only the data feed pair needs to be replicated instead of multiple client-to-client feeds. Tun-

neling multiple client-to-client feeds causes very large data feeds and can affect scalability. Therefore, a feedhopping mechanism was introduced to avoid this storage bottleneck.
To gain a better understanding of the feed tunneling and bundling protocol, it was implemented in an Android app featuring a graphical user interface that enables to test the protocol in the context of everyday applications such as a peer-to-peer chat.

4
tinyISP

In order to enable clients and ISPs to communicate with each other and for the ISP to perform its services correctly, a special protocol is required that regulates the communication and data exchange between both parties.
This project developed the feed bundling and tunnel negotiation protocol called tinyISP. ISPs and clients can implement this protocol which allows them to agree on shared communication channels in the form of append-only logs, which facilitates data tunneling between clients. The following chapter presents the basic idea and functionality as well as the specification of the tinyISP protocol.

4.1 Overview

When two SSB users are not able to exchange their data via peer-to-peer connections, for example if they are not in physical proximity to each other, replication using traditional gossipping methods becomes impractical. To overcome this limitation, Pub server were introduced. These servers can be accessed via the Internet and they replicate all feeds from all clients. However, this solution requires an Internet connection, which is not available everywhere. In addition, all feeds are replicated, which leads to a considerable storage overhead and limits scalability, especially if data is also exchanged between multiple Pub servers.
TinyISP, on the other hand, tries to overcome these limitations by drawing inspiration from the existing functionality of Internet Service Providers (ISPs). The latter is based on the principle that a customer enters into a contract with the ISP in which all the details of their relation are specified. The ISP then provides the client with its infrastructure, which allows him / her to send and receive data over the Internet via the ISP. The ISP acts as an intermediary and forwards the data stream to the desired parties.
A similar principle is used in tinyISP. Clients have the possibility of requesting services from the ISP. It is assumed that the individual terms of the contract have been agreed in advance. Such as the financial contribution of the customer, since the ISP pursues economic interests and also has to finance the necessary infrastructure. The scope and duration of the service must also be specified, as is the case with an ISP on the Internet.
Once the request has been accepted, the communication channels, in the form of append-

only logs, required for the contract are exchanged between the client and the ISP. After the contract is successfully established, the customer can additionally request a communication channel with another client of the ISP. If the other customer accepts it, the data traffic between the two is tunneled by the ISP.
When the contract between ISP and client is terminated, all data associated with that contract can be deleted as it is no longer needed and thereby reducing the storage overhead. Furthermore, the entire protocol is based on Secure Scuttlebutt and append-only logs which adhere to the local-first principle. This ensures that in case of a temporary connection interruption, the functionality of the protocol is not affected. In addition, replication can take place over various media, from direct peer-to-peer connections to storage media such as a hard drive.
To enable communication between the client and the ISP, communication channels in the form of append-only logs are established as part of the signed contract. These logs consist of a pair of control feeds and a pair of data feeds. Since Secure Scuttlebutt is based on single-writer append-only logs, there are multiple feeds in each pair: one for messages from the ISP to the client, while the other feed for messages runs in the opposite direction, from the client to the ISP. In addition, one feed pair is created per client-to-client subscription, which is tunneled through the ISP.
The control feeds are used to transmit important information about the creation, receipt and termination of the contract. Ideally, these feeds should be replicated with higher priority so that both parties can react in a timely manner.
Since a client may subscribe to several clients depending on the terms of the contract, many different feeds would have to be replicated between the customer and the ISP. This could potentially lead to overhead in the replication layer, as the current replication progress would have to be exchanged for each feed. Therefore, in the tinyISP protocol, all feeds that are tunneled through the ISP are bundled into one data feed. Once received on the other side, the log entries can then be reassigned to their respective feeds.
The control/data feed pairs are only valid in connection with the currently existing contract. When the contract is terminated, they can be deleted on both sides. This frees up unnecessarily occupied resources, which reduces the storage overhead. If the customer decides to use the same ISP again, a new contract with new feed pairs can be set up again.
Each individual message of the tinyISP protocol consists of a command and an optional list of arguments needed to execute the command.
The tinyISP protocol can be divided into three different phases: Onboarding, Operative and Farewell. How these individual phases are structured and specified is shown below.

4.2 Onboarding

Before the client can use the service of the ISP, both parties have to agree on the parameters that apply to the contract as part of an onboarding process. The onboarding process also faces the common bootstrapping problem in Secure Scuttlebutt: the customer and the ISP must exchange each other's public keys to replicate the other's feeds. In this project, it is assumed that the customers sign up with the local ISP in person and also exchange their

keys in the process.
Figure 4.1 gives an overview of the onboarding phase.
To request the ISP's services, the client can write an onboard request in its log. The request includes the ID of the ISP, with whom he / she wishes to enter into a contract. It also contains the feed ID of the control feed that the client will use to communicate with the ISP if the contract is successfully established.
After receiving the client's request, the ISP has two options:

1. Reject the contract and notify the client of its decision. The client deletes the previously created control feed and the dialog ends.

2. Accept the contract and respond with the feed ID of its control feed.

In addition, the cryptographic hash of the request is sent so that the clients can correctly assign the response to their request. Since this hash is "unique" (except in the case of a very unlikely hash collision), it forms the so-called contract-ID, which allows the ISP and the client to refer to the same contract.
Since the client and the ISP are now aware of the control feed pair that applies to the contract, further communication is handled exclusively via these feeds.
After the client receives the response, he / she confirms the receipt of the response with an onboard-ack. At the same time, it signals to the ISP that the client's control feed is valid and active. The onboard-ack also includes the ID of the client's data feed used for data transmission between the client and the ISP. The message exchange is very similar to the 3-way handshake [8], which is used, for example, in TCP to establish a connection. Additionally, the ISP also confirms the validity of its control feed with another onboard-ack and transmits its data feed to the client. As demonstrated by the famous Two Generals' Problem [9], there exists no protocol where both sides can be sure that the messages have arrived. However, by relying on the eventual consistency guarantee of the Secure Scuttlebutt protocol, it is assumed that the messages will eventually arrive. With the additional assumption that both parties are not malicious, once these four messages have been exchanged, the contract between client and ISP is successfully completed. The control and data feed pairs created in this process now enable the ISP to successfully provide its services.

4.3 Operative-Phase

The operating phase refers to the stage in which the main tasks of the protocol are executed. In contrast to the onboarding phase, data is now mainly transferred via the data feeds and less via the control feeds. Clients can subscribe to other clients over the ISP to exchange data with each other. The main task of the ISP is to act as an intermediary and to tunnel the data traffic between the clients. This also includes the task of managing the data flows between the ISP and the clients.

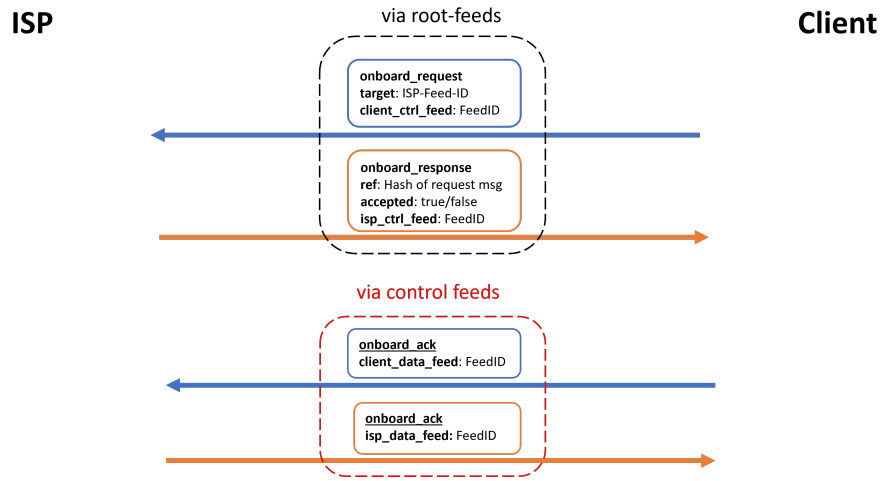

Figure 4.1: Overview of the onboarding phase.

4.3.1 Subscription

When a client requests a subscription to another client, he / she expresses his / her interest in exchanging data with this client over the ISP, whereby both must be registered with the ISP. The ISP acts as an intermediary between the two clients, who jointly conclude a contract specifying the channels through which the data will be transmitted.

Figure 4.2 gives an overview of the single steps for establishing a subscription.

To initiate the process, the client creates a subscription request, which is replicated via the ISP control feed. The request contains the target client and the feed ID of the client-to-client data feed. The cryptographic hash of the request message serves as a reference throughout the further dialog, so that all parties involved can assign the messages to the correct request. The ISP receives this request and notifies the specified client about the subscription request. The notification message includes the ID of the requesting client. If the specified ID is not a client of the ISP, a negative response is returned instead. The other client receives the forwarded request and can accept or reject it. If the request is accepted, the client also sends his / her client-to-client data feed ID back to the sender. The ISP receives the response and forwards it to the requesting client.

With the help of the ISP as an intermediary, both clients have now concluded a contract and agreed on the client-to-client feed pair that will be used to communicate with each other. This contract is exclusively between two clients, which allows them to jointly agree on the terms. These terms could include additional parameters. For example, in addition to the client-to-client feed ID, a sequence number could be specified to indicate from which point this feed should be replicated. It is the responsibility of the other client to verify and accept or reject these parameters. The ISP's role is solely to facilitate the mediation of this contract, not to verify it.

By forwarding the subscription request/response, the ISP now knows which feeds need to be tunneled. However, the content of this feed is not replicated as usual, but multiplexed via the data feed to reduce replication overhead.

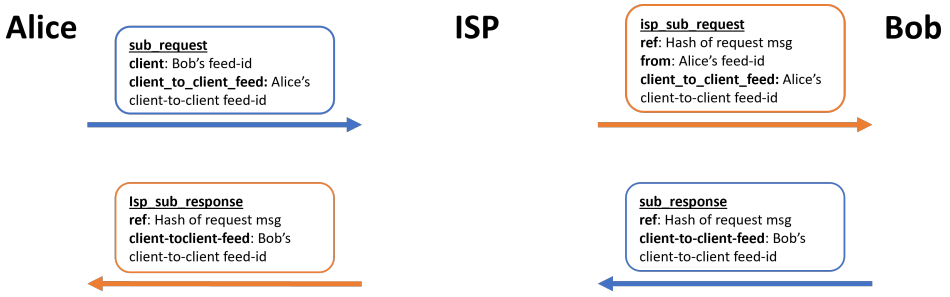

Figure 4.2: Process of a subscription request.

4.3.2 Data Feed Hopping

Once a client has set up multiple subscriptions, data is continuously appended to the data feeds between the client and the ISP. Over time, these feeds become larger and larger, leading to storage problems and not scaling well in the long term.

Since the data feeds only serve to multiplex log entries from several feeds to be replicated, and these are in turn demultiplexed into the associated feeds on the receiver's end, data feeds contain many replicas that are no longer needed. Due to the characteristics of append-only logs, it is not trivial to remove entries from a feed without compromising its integrity and the chain of trust.

Therefore, tinyISP uses a technique called feed hopping. At the end of the feed, a special entry is appended that contains the ID of the next feed to be used. Subsequent messages are then appended to this new log. In addition, the ID of the previous feed is indicated at the beginning of the new feed. This allows to check whether the transmission from and to the correct feed has been performed correctly. Furthermore, by including these additional messages, the chain of trust is transferred to the next feed without interruption (Fig. 4.3).

Figure 4.3: Example of feed hopping in tinyISP.

In tinyISP, the initial data feeds are exchanged during the onboarding phase, with their previous feed ID set to null, indicating the start of the data feed. After the current data feed reaches a certain size, a feedhopping message is appended containing the ID of the next log. This new feed also contains a pointer to the previous feed, validating that it is indeed the subsequent data feed. Feed hopping is performed for both logs in the data feed pair specified in the contract.

Determining the optimal feed size is challenging and depends on user behavior as well as

the scope of services defined in the contract with the ISP. It is recommended to specify the feed size in the contract.
When one side reaches the end of a data feed, it confirms the successful replication of the current data feed by appending a "feedhopping-fin" message in the control feed. This informs the other side that it no longer needs to replicate this feed and can start replicating the new data feed. In addition, the message indicates that the content of the feed has been successfully transmitted, allowing the sender to delete it on its end.
If the ISP and the client have been disconnected for a long time, this may lead to the creation of numerous data feeds that cannot be replicated at the moment. Since each entry in the data feed is a replica originating from another feed, this causes unnecessary consumption of storage space on the part of the sender.
To prevent this, a flow control mechanism is introduced. It limits the maximum number of data feeds that a client or ISP can create without having the previous data feed being fully read by the other side. If this occurs, replication across the data feeds is paused. Replication will only resume when the other side confirms the completion of reading a data feed, allowing the replication process to continue.

4.4 Farewell

Eventually, all the services specified in the contract are fulfilled and the contract can be terminated. This may be the case, for example, if the contract has a fixed term and is not renewed. Due to the eventual consistency property of the Secure Scuttlebutt protocol, it is possible that the ISP still has data for the client that has not been replicated by the end of the contract. Since these messages may still be part of the services provided under the contract, any pending data must be transferred before the ISP and the client can terminate their communication.
Both the client and ISP can inform the other party of their intention to terminate the contract by sending a farewell-initiate message, indicating that they will no longer send new data via the data feeds. Only the data sent before this message was transmitted will continue to be replicated. The other party acknowledges the start of the farewell phase with a "farewell-ack". From this point on, only the "old" data is replicated on both sides.
A "feedhopping-prev" message containing null as the next feed ID serves as an indicator that the end of the data feed has been reached. In this case, the other party is notified with a farewell-fin message indicating that the data feed has been completely read. Once both parties have received and sent a farewell-fin message, this is interpreted as a successful termination of the contract. At this point, communication is terminated and all feeds associated with the contract can be deleted to free up resources.

5
Implementation

To gain further insights into the functionality and features of tinyISP and to analyze its challenges in the context of a real use case, tinyISP was integrated into existing Secure Scuttlebutt (SSB) implementations.
tinyTremola is an Android application developed as part of the vossbol project [3] that allows users to participate in chat and collaboration activities through a user-friendly graphical user interface (GUI) based on the tinySSB protocol. This application provides a suitable basis to test the tinyISP protocol as a client of the ISP.
Since ISPs are usually run on servers and do not require a graphical user interface, an Android implementation is less suitable for this use case. Therefore, the ISP implementation of the tinyISP protocol was integrated into the existing Python implementation of tinySSB [10].
Since tinyTremola and the Python implementation are two separate projects, compatibility, especially at the replication layer, was not given. The Android app implements the ScuttleSync protocol [11], which enables the automatic exchange of public keys between users and thus facilitates onboarding. The public keys are added to a lexically sorted grow-only set ensuring that each user eventually has the same grow-only set. To exchange data, each user sends want- vectors containing the index of all feed IDs in the grow-only set and the sequence number of the next required log entry from this feed. In addition, chunk- vectors are sent that include not only the index of the logs and the sequence number of the entry, but also the next missing chunk index for this entry. As a result, the participants know which entries are missing for the other user and are able to start replicating them. This exchange method avoids sending all feed IDs with each want-vector, which would be very bandwidth-intensive.
In the Python version, on the other hand, individual want-vectors were sent, each containing a single feed ID and the corresponding required sequence number. The system with grow-only sets was not implemented. Therefore, the grow-only data structure with its required methods was implemented into the Python implementation of tinySSB. Based on this, want and chunk vectors were also added, which enable replication with the Android app via UDP multicast. Furthermore, the filesystem for storing feeds and their entries was improved, resulting in significantly better performance when accessing individual chunks of a sidechain

without having to go through the entire chain starting from the log entry, as in the original implementation.
After numerous adjustments, it is now possible for the tinyTremola app and the Python implementation to exchange data with each other. However, since the focus of this project is replication via the ISP and not peer-to-peer exchange between clients, the global ScuttleSync grow-only set was disabled.
In order for the client and ISP to exchange data, a grow-only set is created for both during the public key exchange, which includes the IDs of both parties. Clients do not create shared grow-only sets between themselves to prevent them from replicating in a test setup where all devices are on the same network. Instead, all data traffic should be handled exclusively by the ISP. To make this possible, the grow-only set and replication logic were extended to handle multiple grow-only sets simultaneously.
In addition to various minor improvements, an internal publish/subscribe model has been implemented that enables subscribing and unsubscribing of different feeds to multiple callback methods. These methods are then called when a new entry is received in the subscribed feeds. This allows more efficient processing of new log entries and dynamic assignment of different callbacks during runtime.
Now that both implementations can communicate with each other and the basic infrastructure is in place, the tinyISP was implemented exactly as described in the previous chapter. In the following, the most important implementation-specific details of the protocol are discussed.

5.1 Replication Between Client and ISP

As already described, replication is based on grow-only sets that contain public keys and enable the exchange of missing entries via want and chunk vectors. The want and chunk vectors are only valid in the context of their associated grow-only set. If these vectors would refer to incorrect grow-only sets, this could lead to unnecessary transmission of redundant log entries. Therefore, each want and chunk vector contains an additional identifier, which is the hash calculated from the bitwise XOR of all keys in the grow-only set. This identifier assigns the want or chunk vector to a unique grow-only set (assuming no hash collisions occur). This means that feeds in the set can only be replicated if two or more clients share the same grow-only set.
To establish communication between the client and the ISP, they exchange their identities (root feeds) with each other. Both create a grow-only set that contains the public key of the ISP and the client. This allows to initiate the onboarding process. After the first phase, the client and the ISP successfully negotiate the control feed IDs. These IDs are added to a second grow-only set for both parties, enabling the replication of the control feeds independently of the root feeds. In the final phase of onboarding, the initial data feeds are also negotiated and added to a separate third grow-only set. Now the ISP and the client can exchange data and control messages with each other.
In previous versions, both the data feeds and the control feeds were added to a single grow-only set. While this approach also worked, it had one significant drawback: replication of

data and control feeds was interdependent. Recognizing that control messages are crucial to the proper execution of the contract and are therefore more important than the messages in the data feed, the grow-only set was divided into two parts as described above. This division allows the control feeds to be replicated independently of the data feeds with higher priority.
Client-to-client feeds are not included in any grow-only sets because they are multiplexed by the data feeds and replicated together with the data feeds. This has a significant advantage: assuming a user follows 20 other users through the ISP, a total of 40 different feeds would need to be exchanged between that user and the ISP. This in turn would mean that the want vectors would contain 40 feeds with their entries, which leads to a significant network load, especially in slow wireless networks such as LoRa. However, by multiplexing through data feeds as specified in tinyISP, only two feeds need to be replicated between each client and the ISP, regardless of the number of followed users. This relieves the transmission medium and the saved bandwidth can be used for the actual data transmission.

5.2 Feed Hopping

To avoid the continuous growth of the data feed, feed hopping was introduced in tinyISP. This allows to jump to a new, empty data feed and delete the previous one after all tunneled log entries have been successfully demultiplexed.
This feature has been implemented in accordance with the tinyISP specification in both the Android and Python implementations. When reading a "feedhopping-next" message, the current data feed is deleted and the new feed is added. However, this process contradicts the nature of grow-only sets, which, as explained earlier, can only add new elements and would lead to conflicts between replicas if removal were allowed. Therefore, an epoch counter was added to the implementation, which serves as an additional identifier for the grow-only set. When a key is deleted from the set, the epoch counter, which starts at zero, is implicitly incremented by one, effectively creating a new grow-only set.
The epoch counter of both the client and the ISP must match to enable content replication. When reading the "feedhopping-next" entry, the other side is informed with a feed hopping fin, allowing him/her to increase their local epoch as well. The epoch acts as a grow-only counter and guarantees eventual consistency, enabling continuous replication of data between the client and the ISP.

5.3 User Interface

Both implementations provide a user interface for user interaction, with the Python implementation relying on command-line commands while the client implementation offers a graphical user interface.
Since the Python implementation handles the logic for an ISP, it is intended to function mostly autonomously with minimal user input. However, there are four different commands that can be called via the command line. Firstly, "/me" displays the SSB ID of the ISP along with its corresponding QR code. With "/follow", an SSB ID can be entered to start

replication. This creates a grow-only set with the IDs of both the ISP and the specified client.

Using "/whitelist", IDs can be added to the ISP's whitelist. The whitelist contains all the IDs that are authorized for onboarding, while all onboarding requests from non-authorized IDs are rejected. The whitelist is disabled by default for testing purposes and becomes active as soon as the first ID is added. This feature is particularly useful for ISP operators who want to offer their services only to registered customers.

Finally, with "/farewell", the ISP can terminate the contract with a particular client. The console outputs are mainly for debugging purposes and allow the analysis of the data flow through the ISP.

The user interface in tinyTremola is designed to allow users to intuitively add ISPs and connect to other users to chat with each other. To achieve this, the individual steps specified in tinyISP have been abstracted from the user interface. The user can send various commands to the backend through the GUI. The backend processes these commands autonomously according to the protocol, and the user can monitor the status of the individual operations through the interface.

A new ISP tab has been added to the existing user interface (Fig. 5.1). In this tab, all current contracts with ISPs are displayed with their respective status. To add an ISP, the user can click on the plus symbol and either enter the SSB ID of the ISP or scan the corresponding QR-code. The backend automatically in itiates the replication and creates the onboarding request message. The current status of the ISP is displayed in the list.

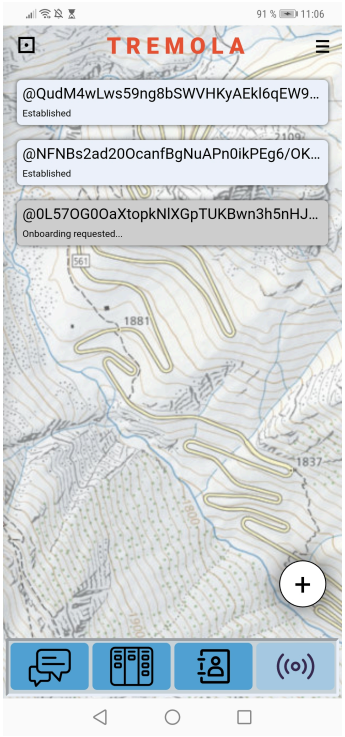

Figure 5.1: ISP tab in tinyTremola.

After successfully concluding a contract, the user selects an ISP from the list to open the corresponding ISP menu (Fig. 5.2). On the left side of the menu, various options are available to the user. Clicking on "Subscribe"-button opens a list of all saved contacts and allows the user to request a subscription to a specific contact. By opening the Requests-Menu, all received subscription requests are listed. Here the user can decide whether to accept or reject the subscription request. In addition, the user can terminate the contract with the ISP via the menu and initiate the Farewell phase of the protocol.
On the right side of the ISP menu, all active subscriptions and pending requests are listed, providing the user with an overview of the contacts they are connected to via the ISP. This ensures that the users have a clear overview of their current subscriptions and all pending subscription requests.

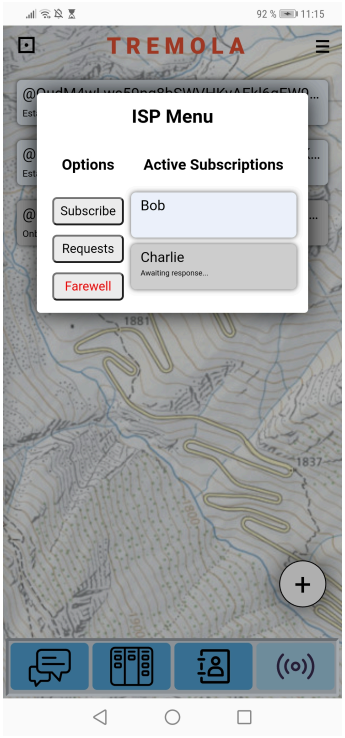

Figure 5.2: ISP menu in tinyTremola.

Since Secure Scuttlebutt is an identity-centric protocol, it would be cumbersome for users to assign individual client-to-client feeds to the correct user. Therefore, the existence of client-to-client feeds is completely abstracted in the application layer. When a user wants to send a chat message to a specific user, the backend checks whether there is a client-to-client feed for that user through an ISP. If such a feed exists, the message is appended to that feed and replicated through the ISP. Otherwise, it is appended to the user's own feed and transmitted via direct peer-to-peer connections, as is common in SSB.
The origin of different feeds is not displayed in the chat app. For the user, it appears as if each message comes from a single ID, namely the SSB ID of the other user.

5.4 Limitations

In traditional SSB protocols, log entries are received and forwarded to the application layer sequentially. In tinySSB, however, the replication of log entries and their sidechains is delayed, which means that log entries with smaller sidechains may be loaded faster and forwarded to the application layer earlier than larger log entries sent previously.
Since tinyISP is based on the assumption of sequential and complete incoming log entries, complex complications arise when implementing it on tinySSB. For example, it is possible that the end of a data feed is read before all sidechains of the feed have been received. The client mistakenly assumes that it has read and demultiplexed this data feed completely and discards it, even though there are still pending replications for this log. This leads to premature feed hopping and missing data. To prevent this, a complete restructuring of the backend in the two tinySSB implementations would be necessary.
In addition to the peer-to-peer chat, the tinyTremola app also offers a Kanban board application that allows multiple users to collaborate. Unlike the tinyISP protocol, which is based on client-to-client communication via the ISP, the Kanban board application is characterized by the collaboration of more than two users. Currently, there is no compatibility between these two structures.

6 Conclusion

This project shows the feasibility of ISPs implemented in the Secure Scuttlebutt protocol and append-only logs. For this purpose, the tinyISP protocol was developed to regulate communication and data exchange between ISPs and clients, as well as between the clients themselves. Clients can conclude contracts with ISPs and also enter into contracts with other clients. By extending the classic identity-centric system and splitting the communication into smaller feeds, the tinyISP protocol reduces the replication-intensive onboarding process for new SSB users. Furthermore, the feeds are linked to contracts so that they can be deleted by expiration of the contract. Together with other optimizations, such as data feed hopping, this overcomes the main limitations of traditional Secure Scuttlebutt Pub servers. In addition, users do not need to be continuously connected to the ISP to receive data as the entire protocol is based on append-only logs.
To enable Secure Scuttlebutt for larger communities, the scalability of the protocol, especially replication, needs to be improved. This project demonstrates a possible solution approach using tinyISP and provides an implementation in an everyday application. By addressing the challenges of replication-intensive onboarding, optimizing data feed management and introducing the concept of ISPs, tinyISP offers a potential solution to improve the scalability of Secure Scuttlebutt. The prototype of this project serves as a demonstration of how such improvements can be applied in real-world scenarios and shows the benefits as well as the challenges of the proposed approach.

6.1 Future Work

An outlook on further possible improvements of the presented tinyISP protocol is discussed below.

6.1.1 Multiple ISPs

In this prototype, every user must be a client of the same ISP to establish communication between them. But it is unrealistic to assume that a single ISP can provide such an extensive infrastructure on its own.

Just as the Internet relies on data exchange between ISPs to enable connectivity between clients of different ISPs, the tinyISP protocol would need to be extended to allow one ISP to be a client of another ISP. This would facilitate connectivity between the ISP's own clients and clients of other ISPs, thus expanding the reach and connectivity of the network.

6.1.2 Malicious Actors

When developing the tinyISP protocol, it was assumed that both ISPs and clients are well-meaning users. But in reality there are a variety of malicious actors who can exploit different vulnerabilities in the protocol. For instance, they could launch Denial of Service attacks (DoS) to disrupt connectivity in a particular region.
Therefore, the various ways of exploiting the protocol should be thoroughly analyzed, and appropriate countermeasures developed.

6.1.3 Data Exchange between More than Two Users

The tinyISP protocol enables client-to-client communication via ISPs. But there is a large number of applications that require data exchange between an entire group with more than two users. It therefore remains to be investigated whether compatibility can be established between the tinyISP protocol and such collaborative applications.

Bibliography

[1] Scuttlebutt Protocol Guide. URL https://ssbc.github.io/scuttlebutt-protocol-guide/index.html. Retrieved: 25.06.2023.

[2] Secure Scuttlebutt. URL https://scuttlebutt.nz/. Retrieved: 25.06.2023.

[3] Vossbol. URL https://github.com/tschudin/vossbol. Retrieved: 20.06.2023.

[4] Aljoscha Meyer Dominic Tarr, Erick Lavoie and Christian Tschudin. Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Applications. In *Proceedings of the 6th ACM conference on information-centric networking*, pages 1–11, 2019.

[5] Jannik Jaberg. A Feed Bundle Protocol for Scuttlebutt. Bachelor thesis, University of Basel, 2020.

[6] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-24550-3.

[7] André Staltz. SSB Room. URL https://github.com/staltz/ssb-room. Retrieved: 25.06.2023.

[8] Andrew Tanenbaum. *Computernetzwerke*, pages 560–565. Always learning. Pearson, München, 5., aktualisierte auflage edition, 2012. ISBN 3868941371.

[9] Andrew Tanenbaum. *Computernetzwerke*, pages 517–522. Always learning. Pearson, München, 5., aktualisierte auflage edition, 2012. ISBN 3868941371.

[10] Christian Tschudin. Low-Level Secure Scuttlebutt Packet Spec. URL https://github.com/tschudin/tinyssb/blob/main/doc/03-packet-spec/packet-spec.pdf. Retrieved: 25.06.2023.

[11] Christian Tschudin. A connectionless grow-only set crdt. In *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good*, DICG '22, page 25–30, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450399289. doi: 10.1145/3565383.3566110. URL https://doi.org/10.1145/3565383.3566110.

[12] International Telecommunication Union. Statistics: Individuals using the Internet. URL https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx. Retrieved: 25.06.2023.