tinySSB Proof-of-Concept 2: tinyChat

2022-04-06 christian.tschudin@unibas.ch

Abstract: We show a simple chat application between two or more peers (similar to Secure Scuttlebutt's "global chat channel") where content replication is fully functional both for log entries as well as blobs (sidechains). Limitation: we ship "plain mail content", i.e. we don't demultiplex log entries among different applications or encryption levels. The command-line interface ot tinyChat is a subset of the venerable UNIX mail command.

1) Setup and Usage

Create the local data repositories for three participants (Alice, Bob and Carla)

```
% ./tinyChat_init.py
```

Chat clients can now be started locally and will communicate via the UDP multicast group (224.1.1.1/5000). In independend terminal windows, start the different users' chat program, for example for Alice, as follows:

```
% ./tinyChat.py Alice
tinySSB Chat (tinyChat)
  creating face for UDP multicast group
    address is ('224.1.1.1', 5000)
  starting thread with IO loop
  starting thread with replicator loop
you have 0 messages
tinyChat> _
```

and start using it:

```
tinyChat> m
To: *
Subject: my first post
<< end message body with ^D at start of empty line
but nothing important to say.
cheers, Alice
<< message sent

tinyChat> s
sync: 1 new message
   1* my first post

tinyChat> p
--- #1
From: Alice (@Af1OSAI9uqVA1NW5zkx2G25cVTZHK4s+hd/iQviIcWk=.ed25519)
Date: Wed Apr  6 09:52:56 2022
To: *
Subject: my first post

but nothing important to say.
cheers, Alice
---
```

Assuming that in the meantime Carla wrote a reply (see column to the right), Alice can sync her mail store and print the new mail (column to the left):

```
# Alice's tinyChat client:

tinyChat> s  # sync mail store
sync: 1 new message

tinyChat> 1  # select the newest message, before printing
   1* Re: my first post

tinyChat> p
--- #1
From: Carla (@iZ/9nHmHGG3pZ1pMucXEQaEpqfgh2QC/v/Eh3YasCs0=.ed25519)
Date: Wed Apr  6 11:37:32 2022
To: *
Subject: Re: my first post

Alice, nice to meet you. Best, Carla
---
```

```
# Carla's tinyChat client:

tinyChat> s
sync: 1 new message
   1* my first post

tinyChat> r
To: *
Subject: Re: my first post
<< end message body with ^D at start of empty line
Alice, nice to meet you. Best, Carla
<< message sent
```

Enter `?` at the tinyChat prompt to learn about the (UNIX mail) commands.

2) Insights

The implementation of tinyChat provided several insights regarding:

- software structure
- need for replication policies

2.1) tinyChat Software Structure

Compared to the first PoC, we have now separated the replication logic from the underlying mechanics provided by the Node object.

```
.------------------.
| tinyChat frontend |  command line parsing, display of results
|                   |
|   mail backend    |  parsing of log entries, sorted list of mail messages
|                   |
|     replicator    |  replication protocol (want/blob requests)
|                   |
|       node        |  mechanics for demultiplexed callbacks
|                   |
|         repo      |  persistency of logs and blobs
|                   |
|         io        |  outgoing transmission queue, packet scheduling
|                   |
|           faces   |  link layer interface, currently UDP multicast
`------------------'
```

The `mail backend` layer is responsible for harvesting all mail-related log entries and creating a sorted list of mails for the frontend to display. In the future, the backend layer will also handle application demultiplexing and a CBOR-based representation of a chat message where, for example, the message could contain attachments, have backlinks, and/or the message could be encrypted.

2.2) Need for Replication Policies

Several decisions are currently hardcoded but would need either discussion about the fitness or become tunable:

- packet queuing: all packets are blindly queued, regardless if the same bits are already in the queue (for slow channels), or whether the queue should do tail-drop
- packet scheduling: there is no priorities among feeds, nor blobs vs log entries
- push of log entries only: a fresh log entry is immediately sent out on all faces but the blobs of the sidechain are not, meaning that they have to be fetched as otherwise they are not replicated
- ondemand replication of sidechain: we implemented eager blob chain replication i.e., if a received log entry has a sidechain, all chained blobs are immediately requested
- there is no retry limit (for fetching a sidechain, and for log entries), nor adaptive ARQ timeout
- a single timeout is used for both blob chains and log entries

---