p2pdb

Encrypted Peer-To-Peer Database

Basic Concept

Users run **applications** on their own systems using **remote p2pdb nodes** as a backend for storage of database entries. Applications can be with source or binary, and can be existing webapps that run a local “server” (so the user browses to localhost:5000 to access the app, which uses p2pdb as a db backend)

The Database - More Info

p2pdb stores database entries which are **encrypted** with the RSA key of a **recipient**, and **signed** with the RSA key of a **table**. To post to a table, you must therefore have the private key of the table (authorized to post). To read a message, you must have the recipient private key (authorized recipient).

The Database - More Info

p2pdb provides no guarantees about the retention or propagation of your message through the network. Its routing is “best effort.” p2pdb requires a proof of work unique to the combination of the message payload and table it’s in - the client solves a set of ~30s long SHA-based puzzles to prevent flooding/DoS

The Database - More Info

Our goal is to, for small sites, give the user total control over their data. Deniability and privacy is provided by an (eventual, not yet implemented) Tor interface between clients and nodes. Nodes have deniability about the data they store. Users have control over the recipient of their message and its scope

The Database - More Info

Users can easily change the table keys for an app. Imagine breaking out a private version of reddit with the same features as the full site for you and your friends with a simple key change. We also provide for an authentication and login base (through the user's recipient keys)

The Nodes - More Info

Nodes are users who are interested in the application and are technically capable of providing hosting for enough messages in the application. Savvy users with access to server or cloud hardware (relatively cheap to store up to 10 million entries)

The Developer - More Info

Developers create the application initially and distribute it to users. They likely also initially host a node, but as the application gains traction and interested users mirror it, the developers can stop hosting any nodes and be completely removed from the hosting of the application.

The Point - More Info

Applications would also be resistant to takedown. Ideal for controversial applications with small database - ThePirateBay, protest and counterculture sites, etc. Without taking down all nodes the application will keep running.

The Point - More Info

For this to be feasible and practical, only a small number of database entries need be supported.

What Works Today

A small scale, unoptimized demo of the backend server software run by the nodes is available. A script to test posting messages to these nodes is also available. No client implementations are currently available, though they are more trivial than the server.

Server Operation - Put

One possible operation is putting a message/db entry into the system. It requires a POST call to /put, with parameters including the message ID, a signature of the payload with the table key, a proof of work involving the payload and table key, a recipient key with whcih the message is encrypted

Sever Operation - Get

Get an entry from the DB. Must provide at least one but can provide many **filters** as GET parameters in request.

Filters include seconds_since, bucket, a list of recipient keys, a list of table keys, etc

Sever Operation - Index

Provides a list of buckets and the MD5 hashes of the message IDs they contain (in ascending order). Up to **5k** buckets (can change parameters). Bucket chosen based on MD5 hash of message payload. MD5 chosen for speed, security is unimportant as these are just optimizations. Nodes store hashes they have seen for each bucket, don't refetch messages in buckets whose hashes they have seen in the past. Currently, no limit to number of hashes stored. Eventually: 1k limit?

Sever Operation - Hello

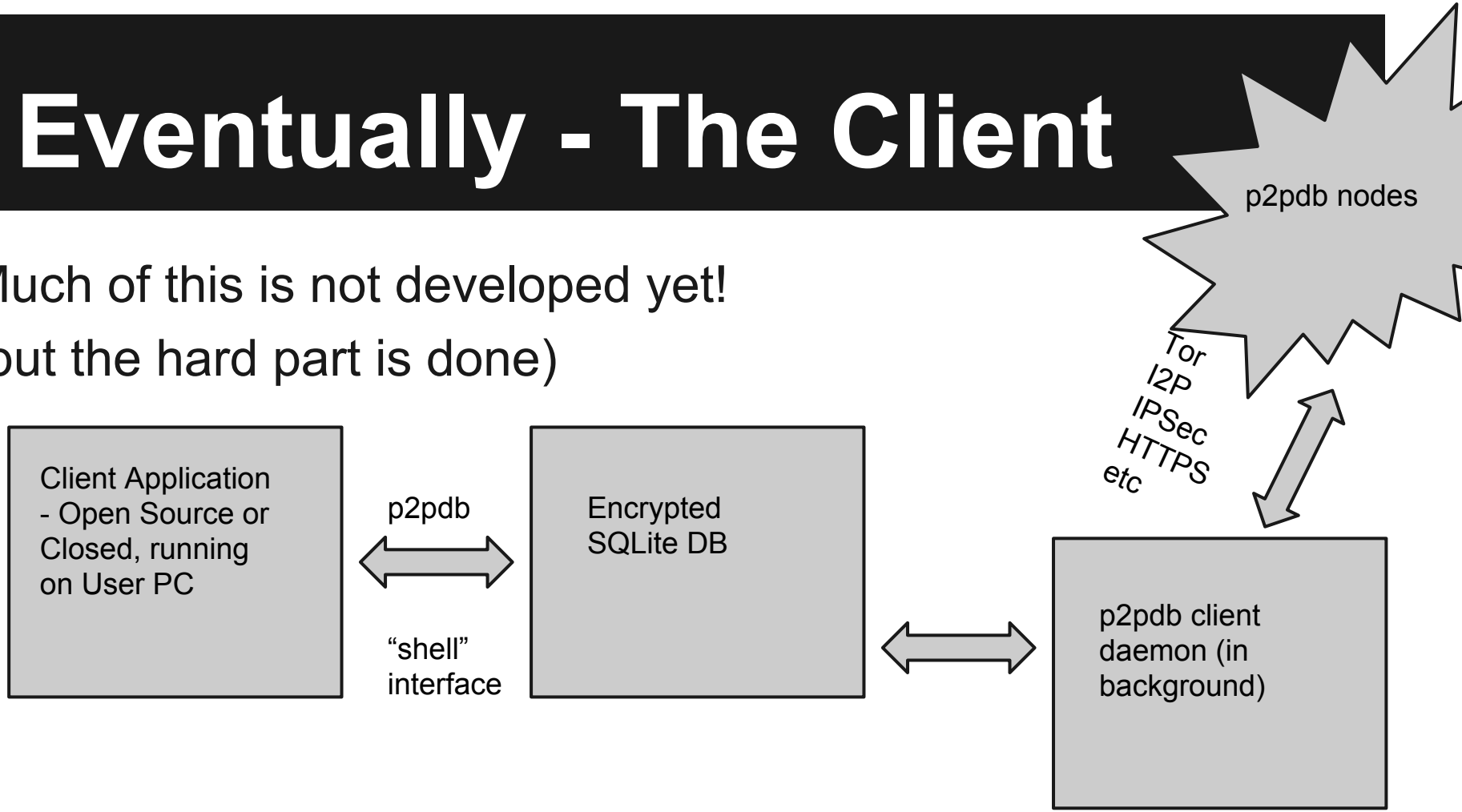
First message sent when nodes synchronize. Server A sends its available routes and supported tables to Server B, which stores Server A's information if it has not seen Server A before. Server A then sends info on all the nodes it knows about to Server B in response.

Eventually: Allow filtering to only servers which support a certain list of tables (servers can choose to mirror only certain tables)

Eventually - The Client

Much of this is not developed yet!

(but the hard part is done)



More Future Plans

Implement Tor support for nodes (“shadow nodes”). Implement Tor support for clients. Investigate I2P support. Conduct stress tests with millions of messages and implement appropriate optimizations. Test deployments on existing Flask and Django webapps.

(yes, I really will do these. this is my baby :))