Fedimint Grant Proposal

Justin Moeller

Overview

My name is Justin Moeller and I am seeking funding for getting started working on open source Bitcoin development, specifically working on Fedimint. I recently learned about Fedimint through the company Fedi's founding announcement and discussions on the podcast Rabbit Hole Recap. Since learning about the overall vision, the project has piqued my interest in numerous ways. I see Fedimints as the natural evolution of Hal Finney's Bitcoin banking proposal from Bitcoin Talk, but with the ability to provide improved *privacy*, *scalability*, and a different tradeoff balance for Bitcoin users when choosing how to custody their coins.

In preparation for this proposal, I have been trying to learn the code and figure out areas that I would be able to help contribute. With my background in databases and in consensus protocols, I would be interested in helping contribute to the robustness of the **database** integration into Fedimint as well as contributing to the core **consensus** module of the federation.

Who am I

As mentioned above, my name is Justin and I'm looking to get into open-source Bitcoin development full time. This section is intended to briefly describe my background which should provide some motivation for why I'm interested in certain areas of Fedimint. I'm currently a senior software engineer at Microsoft working on Azure SQL Database and Service Fabric, which is Microsoft's backbone for Azure. From my experience at Microsoft (and interning at Google) I have had exposure to almost every kind of database from key-value stores, document stores, to SQL engines. A large portion of my experience at Microsoft has been "micro-servicing" components inside of Azure's stack, which has required solving hard distributed systems problems such as shared state across independent processes, bootstrapping the microservice, and developing a network protocol. Additionally, I do have some experience with replicated state machine protocols, such as Raft, Paxos, and Zab (here is a brief video of me diverging replica data in Zookeeper, which uses Zab as it's consensus protocol).

Since most of my background has been working on databases and distributed systems, the database portion and consensus protocol of Fedimint is attractive to me. I think my background fits nicely with some of the problems that Fedimint will face during development.

Database

Fedimint uses RocksDb to store federation transaction state. Right now, the integration is in an early stage – the transaction support is limited (currently, this means during a consensus epoch, the mint cannot "read our own write") and investigating the database state is currently difficult. Unlike other database clients, RocksDb does not provide a command line interface for inspecting the database state. It will be useful for debugging Fedimint to be able to dump and inspect the current state of the database, but in order to do this, it will be necessary to write a tool that is capable of reading the database state and displaying it in a readable format for the developer. I would like to contribute to this tool and make it easy for developers to use (and potentially allow it to be used by federation members as well to inspect epoch history).

Beyond the RocksDb integration, I think there are other areas worth exploring regarding the database, such as fast migration/recovery (if a federation member needs to migrate hardware) and encrypted backups. There will need to be tooling built that enables federation members to easily perform certain actions on their database.

To summarize, I would like to be involved in the integration of the RocksDb implementation into Fedimint by providing code review, testing, and my own contributions to the code. I am also interested in helping with the dbdump tool and making that a robust tool for debugging epoch history state.

Federation Consensus and Performance

The other area that I would like to contribute to would be the core consensus module and evaluating its performance. Fedimint uses the "Honey Badger BFT" algorithm for achieving consensus between epochs. Each epoch a set of transactions are proposed and if all the nodes come to agreement, then the transactions are deemed to have been confirmed.

Given that the consensus part of Fedimint is the core of the system and needs to be robust, I think my main contribution here would be to get ramped up with how Fedimint uses HBBFT and be able to help with code reviews to ensure that the quality of the implementation remains high. I would also like to add integration tests and potentially refactor some of the code to allow for mocking of a federation. This would allow more acute testing of specific failure scenarios (i.e 2 nodes in a 7 node federation are faulty and 1 is malicious).

Another interesting project regarding consensus that I would like to work on would be doing a comprehensive performance evaluation of a Federation's throughput (similar to Eric Sirion's evaluation based on transaction latency). I see Fedimint as a scalability option for Bitcoin so it would be interesting to evaluate the transaction throughput that Fedimints are able to achieve. Additionally, the original HBBFT paper makes the claim that the consensus protocol scales with the underlying network. This performance evaluation would attempt to discover if this claim is true in a real system and if there are any overheads that prevent Fedimint from achieving this. The intuition behind doing this is that, if Fedimints are going to be deployed in the wild, they might be in areas with low bandwidth between federation members. This evaluation might provide some insight on how Fedimints are expected to perform when deployed in the wild. Below are some example setups I would be interested in evaluating

1) In a real deployment (i.e multiple VMs), saturate the federation with transactions and vary the bandwidth between nodes. Measure the overall transaction throughput as the bandwidth between nodes decreases.
2) In a real deployment (i.e multiple VMs), vary the size of the federation and saturate the federation with transactions. Measure the overall transaction throughput as the number of Federation members increases.

Conclusion

As described above for this grant proposal, I would like to focus my efforts on the database integration into Fedimint as well as learning the consensus protocol and orchestrating a performance test to measure transaction throughput. I anticipate a steep learning curve at the beginning (likely around 2 months). I am an experienced C++ developer but do not have any Rust experience and I know learning a new language as well as a new project can be challenging. I also know that working on Fedimint full time would require me taking a large pay cut in comparison to my current position. Despite this, I am really

excited to take my next step into Bitcoin and open-source development. Bitcoin currently consumes all my free time and my thoughts, so an opportunity to independently work on a project that can help advance the adoption of Bitcoin and push Bitcoin forward is the most valuable aspect to me. I hope that I can spend more of my time working on Fedimint.