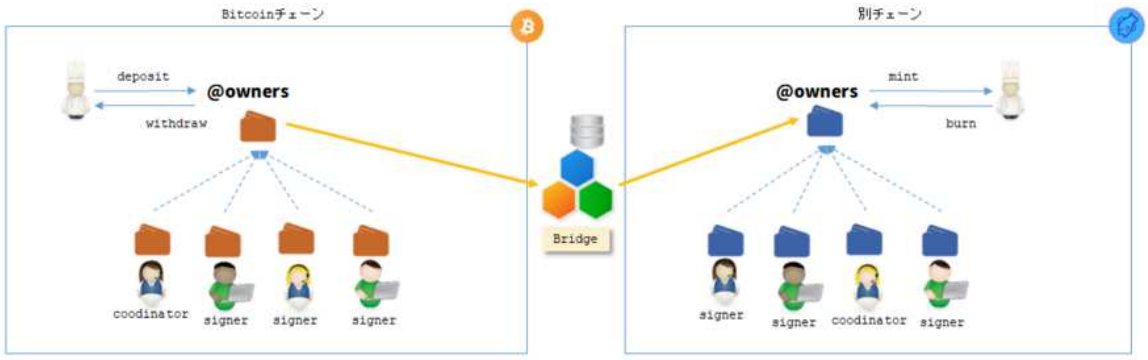
マルチシグ方式ブリッジの試作

このページでは試作しようとしているマルチシグ方式ブリッジの検討や課題についてのみ記載する。
詳細や作業の様子は別のページに書く。が、実装のほうが先に進むのであまり当てにならない。実装前に検討したものくらいの気持ちで見てほしい。

- マルチシグ方式のPeg-inを作る
- マルチシグ方式のPeg-outを作る
- マルチシグ方式ブリッジ作成中の問題
- マルチシグ方式ブリッジのBDK問題



仕様の概要

- Bitcoin側を基点とした Ethereum とのブリッジ
  - Ethereum 側は適当な ERC-20 トークンを発行する
- Bitcoin regtest と Ethereum のローカル環境で動かす

- deposit / withdraw のユーザと mint/burn のユーザは同一と仮定する。
- Ethereum 側では ERC-20 の wrapped BTC トークンを mint/burn する。
- 手数料は固定で peg-in 時に 1,000 sats、peg-out 時に 1,000 sats 取る。

実装していて思ったこと

単なる感想も含む。

- Peg-In/Peg-Outリクエスト者が申請したBitcoinアドレスとEthereumアドレスがその人のものかどうかわからないのでマネーロンダリングに利用しやすそうだと思った。
  - これはブリッジじゃなくてもそうだが、他のチェーンをまたぐので追跡が更にしにくくなる。
- マルチシグウォレットで運用するようにしたのでブリッジノードを複数立てているが、とても面倒。
  - 中央にサーバーを立てたりDBを立てたりすると関係が対等ではなくなるのでP2Pで通信し合うようにした
  - そうなるとノード間での情報共有が大変
    - 今回は10ノード程度と見積もって、タイムスロット・ラウンドロビン制にしている
    - 最終的にはノードから定足数の署名を集めるマルチシグなのだが、リクエストが正しいかどうかの判定が難しい
- ノード間の情報共有にEthereumを使うのが楽だが、L1はETHが必要なのでコストが掛かりすぎる
  - 自前のL2であれば実質的にコストが掛からないので中央DBとして使うことができる
  - 今回はP2Pでやっているので、スロットの切り替わりで担当ノードが重なってしまう可能性なども考えないといけない
- Ethereum側のコストがよく理解できていないので、スマコンが大きいのか小さいのかよくわからない
  - Bitcoinにしか親しんでいないともいう
- 最初は、リーダースロット当番→Peg-In/Outできそうな要求を探して見つかるごとにP2Pでリクエスト送信→レスポンス受信のたびに処理、というようにパラレルで行っていたが、いろいろ無理があったので、リクエスト送信してからレスポンス受信までは待つようにした(Peg-InとPeg-Out自体はパラレル)。
  - P2Pの再送が必要な状況になると、メッセージのシーケンス番号が入れ替わったり、最後に送信したメッセージと同じ内容かど

うかのハッシュ値チェックが使えなくなったりする。
P2Pの接続要求を1回しか行っていなくても複数接続されることがあり、順番にリクエストしても場合によっては順番が入れ替わることがある。

検討や詳細など

分散合意

タイムスロット制

スロット番号は、現在のUNIXエポック秒数を固定のタイムスロット期間秒数で割った整数値とする。

タイムスロット期間秒数は処理時間と比較して十分に長いものとする。2026/01/13現在は 5秒。

リーダーの決め方

libp2p が最初に生成した UUID を全参加ノードで共有する。

UUIDを昇順でソートし、その順番がリーダーとなるノードの順番とする。

スロット番号をノード数で割った余りが指すUUIDが現在のリーダー

前準備

Ethereum 側で適当に ERC-20 のトークンをマルチシグコントラクトに mint しておく

peg-inはユーザに mint するのではなくマルチシグコントラクトから送金するタイプにした

total supply の管理はしやすいが発行量が不足する可能性がある

Bitlayerだったか他のL2だったか忘れたが、そちらはmintするタイプだった。一長一短だと思う。

Peg-in

ユーザはERC-20トークンを発行してもらう EOA と引き換えに Bitcoin の deposit 用マルチシグアドレスをもらう

UTXO を登録してもらうのは難しいので EOA だけにしよう

ユーザは自分の Bitcoin をマルチシグアドレスに支払う

管理者はマルチシグアドレスを監視していて、confirm 数が規定に達したら(3くらい？) Ethereum 側で登録済み EOA にトークンを発行する。発行額は送金額から手数料として 1,000 sats 引いておく。

マルチシグへの送金 UTXO はこのまま維持する？

Peg-out

ユーザは取り戻す Bitcoin 送金先アドレスと引き換えに ERC-20 の withdraw 用マルチシグアドレスをもらう

これはスマコンのアドレスでよいのか？

ユーザは自分の ERC-20 トークンをマルチシグアドレスに支払う

管理者はマルチシグアドレスを監視していて、confirm 数が規定に達したら(1でよい？) Bitcoin 側で登録済み Bitcoin アドレスに Bitcoinを戻す。戻す額は ERC-20 で支払われた額から手数料として 1,000 sats 引いておく。

Peg-in 時の UTXO を維持している場合はこのタイミングで使用する？

必要と思われる機能

基本機能

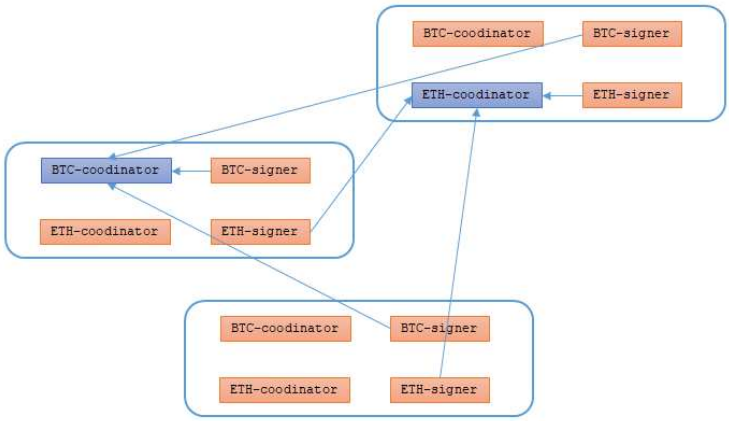
- ☐ Bitcoinウォレット(今回は P2TRのみとする)
  - ✔ 各signer
    - 秘密鍵管理(HDウォレット)
      - アドレス生成を含む
    - 残高管理
    - key path 署名
    - PSBT 署名
  - ✔ マルチシグcoodinator
    - signerの拡張公開鍵によるアドレス管理

- アドレス生成を含む
- 残高管理
- 署名用 PSBT 作成
- 各 singer からの PSBT 署名集約
- ✔ ブロックチェーン監視機能
  - ✔ DB に監視対象のマルチシグアドレスがあり、confirm 数が達したら通知する
- ✔ Ethereumウォレット
  - 各signer
    - MetaMaskに任せようと思ったが使いづらかったので最低限実装する
    - 秘密鍵管理(HDウォレット)
      - アカウントは固定するので世代管理などはしない
    - 残高出力
    - 送金
    - 署名
  - マルチシグcoodinator
    - Bitcoinの場合はマルチシグアドレスの UTXO から支払うが、Ethereum では ETHを使ってコントラクトを呼び出すので誰かの EOA を使うことになる。
  - ブロックチェーンの監視機能
    - マルチシグスマコン
- ☐ ノード間通信
  - ✔ P2P通信
    - プロトコル
      - ✔ PSBT署名集め
      - ✔ スマコン署名集め
    - ✔ libp2pによるP2P通信
  - ✔ リーダー交代機能
    - 不在になる可能性がある
    - タイムスロット制でP2PノードID順でのラウンドロビンにしている
      - Bitcoinウォレット
- ☐ ブリッジ機能
  - Peg-In (BTC → ETH)
    - ユーザにマルチシグBTCアドレスを提示し、そこに支払ったらトークンを送ることになるEOAを DB登録する
      - マルチシグアドレスは使い回しせず、終わったら使用済みにする
    - マルチシグアドレスに入金があると、使用済みでなければ登録した EOA に対してのものと判断してトークンを送る
    - アカウントがないのでいたずら目的で登録する可能性もある。今回は考慮しない。
    - タイムアウトの考慮
      - ユーザのtx feeが低くてなかなか承認されないのはユーザ責任だが、その状態といたずらや間違いで登録しただけという状況の違いを区別すべきか？
        - 監視するアドレスが増えるとそれなりにウォレットの負荷が高くなる。mempoolは追わなくても良いと思う
        - ユーザごとにpeginするBTCアドレスを割り振るのではなく毎回アドレスを割り振る方式のほうが常時追跡するアドレス数が少なくてよいのではないかと考えている。UIで同じユーザが同時に複数のpeginをできないようにするなら最大でもユーザ数と同数になる。タイムアウトがあるなら監視対象は少なくできるはず。
      - ETHトランザクションがなかなか承認されない場合
  - Peg-Out (ETH → BTC)
    - ユーザの EOA と送金するBTCアドレスを DB登録する
    - マルチシグのスマコンアドレスにトークン送金され、送金元が登録されているEOAであればマルチシグBTCアドレスから対応する BTCアドレスに送金する
      - スマコンアドレスは変更できないので使い捨てにできない
    - タイムアウトの考慮
      - トークンをスマコンに送信してもらうだけなので、どのEOAからの送金をどのBTCアドレスに送ればよいのかさえ登録し

てあれば

ノード間で共有しているデータがないので、DBも個別にしかもっていない。つまり Peg-In / Peg-Out リクエストしたノードが coodinator として働くことになる。

Bitcoin側もEthereum側もマルチシグなので、signer と coodinator の役割がある。
coodinator はラウンドロビンなりなんなりで回していくことになるだろう。となるとサーバ・クライアントではダメなので P2P になる。



DB

peg-in

peg-in だけであれば `btcm_addr` への UTXO をそのままにしておいてよい。

pegin

■	String	マルチシグアカウントから振りだしたBTCアドレス	primary key
dt	String	登録日時(UTC)。RFC-3339	登録時のみ
eoa	String	peg-inするEOAアドレス	登録時のみ
fee	u64	peg-in手数料。token転送時に差し引く。	登録時のみ
status	i32	peg-inステータス。0: 登録直後, 1: peg-in中, 2: peg-in済み, 3: ETH tx finalized	
out_point	String	btcm_addrに入金を確認した最初のout_point。<TXID>:<INDEX>	
token_sign	Array(Sign)	集めたEthereumスマコン署名の配列(Sign, SignerAddress)	
token_hash	String	token_signの元になったハッシュ	
token_amount	u64	eoaに転送するtoken額。マルチシグアドレスに送金されたBTCからfeeを引いた額。	
token_txhash	String	peg-inしたEthereum transaction hash	
psbt	String	out_pointから支払いリクエストによるPSBT	combineにより更新
psbt_finalized	bool	`psbt`がfinalizedしているかどうか	

peg-out

pegout-map

■	String	peg-outするEOAアドレス	primary key
dt	String	登録日時(UTC)。RFC-3339	登録時のみ
btc_addr	String	peg-outするBTCアドレス	

pegout

■	String	contract送金を確認したTxHash	primary key
dt	String	登録日時(UTC)。RFC-3339	登録時のみ
eoa	String	送金元EOA	登録時のみ
btc_addr	String	peg-outするBTCアドレス	登録時のみ

fee	u64	peg-out手数料。BTC転送時に差し引く。	登録時のみ
status	i32	peg-outステータス。0: 登録直後, 1: contrace送金済み, 2: peg-out中, 3: peg-out済み	
btc_amount	u64	btc_addrに転送するBTC額。contractに転送したtoken量からfeeを引いた額。	
psbt	String	署名されたPSBT	combineにより更新
txid	String		

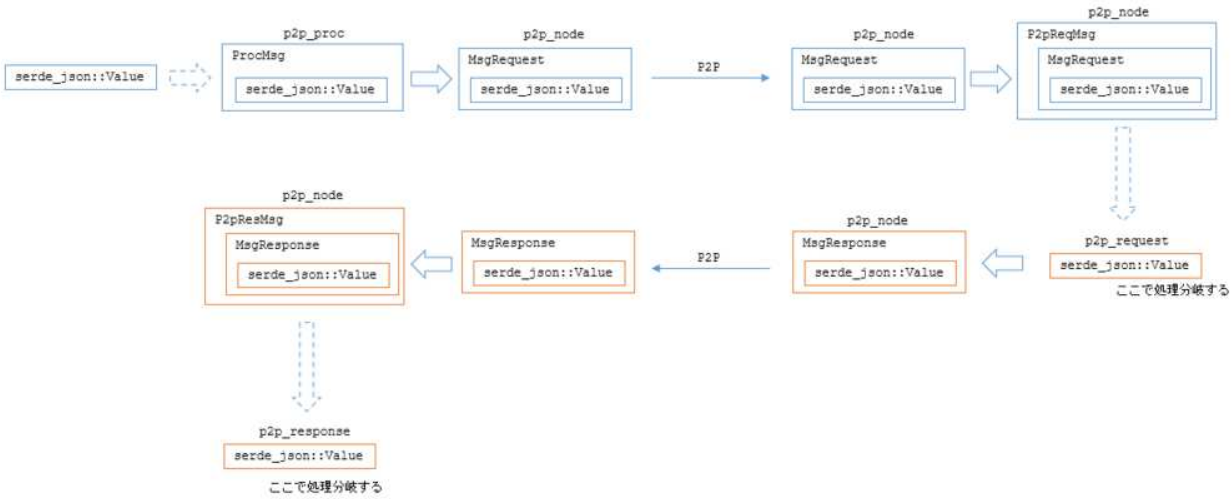
設計が必要

- peg-in と peg-out に時間差が生じる
  - Grok
- マルチシグの拡張公開鍵(Bitcoin)やアドレス(Ethereum)の集め方
- Ethereum のマルチシグで何に対して署名するのかを決める
  - Ethereum にはマルチシグという概念はないのでスマコンで作ることになる
  - 署名対象を適当に決めてしまうと使い回しできてしまう
  - 🌺 スマコンで nonce を持ち、リクエストごとにインクリメントして他のパラメータとあわせてハッシュ計算した値に署名させている。検証するスマコンもその値を使っている。
    - ❌ このやり方だと署名が全員終わるまで nonce に関わるリクエストができなくなるので再考が必要。
- Bitcoin のマルチシグは PSBT に署名するだけなので、どういう PSBT に署名しようとしているのか signer にわかりやすくした方がよい。
- Bitcoin と Ethereum の各人のウォレットは鍵を共通にした方がよい？ 別が良い？
  - 今のところ Bitcoin は拡張秘密鍵、Ethereum はニモニックをそれぞれ生成して別にしている。
  - Bticoin は raw transaction があればウォレットがなくても展開できるが Ethereum は EOA がないと展開できないので、Ethereum 側には ETH がいる。今のところマルチシグスマコンを呼び出すことができるのは署名で登録した EOA だけなので、ウォレットの EOA が ETH を持つ必要がある。
- 署名者の入れ替え
- Bitcoinマルチシグウォレットのコインセレクション
  - BDK でできそうだが、PSBT をどうすればよいかわかっていない
- Peg-in する EOAアドレス、Peg-out する Bitcoinアドレスをどう管理するか
  - トランザクションに載せるとみんなに見えてしまうので DBか？
  - 全部のノードが参照できる DB を用意するのは筋が悪いか？
- 署名が足りなかったときに再送するしくみ
  - 既にもらっているノードには送らないようにするなども
- BitcoinやEthereumにリクエストして失敗するときの対応策
- peg-in でのマルチシグアドレスは HTLC にすべきか
  - 十分 confirm したら peg-in を開始する、というのはわかりづらい
  - 「十分」をどこに定義するのか。ハードコーディングは避けるとしても設定ファイルだとまちまちになる。

メッセージ

メッセージの流れ。非同期で待ち受けるタスクは内部での通信なのでどうとでもなるが、P2Pは型が限られるのでJSONにしている。ついでなのでタスク間もJSONになっている。

- P2Pの送受信はタスクが別でないと動けないと思う。 `p2p_node` がP2P通信の根元で、その周辺に `p2p_proc` がいる。この2つは同じタスクである。
  - P2P送信要求メッセージを `p2p_node` が受け取ると `p2p_proc` の `send_request_handler()` を呼び出してコマンドごとにどうするか処理している。オブジェクト指向っぽくない実装になっている。
  - `p2p_request` と `p2p_response` はそれぞれ別のタスクである(`p2p_subscribe` もある)。受信したP2Pメッセージが `MsgRequest` か `MsgResponse` かの違いである。



P2Pメッセージの重複判定

今のリーダーからメッセージが来たとしても、それが前回と同じであれば処理は不要である。

これはメッセージにシーケンス番号を付与し、前回そのノードが送ってきたシーケンス番号以下であれば古いと判断して破棄すれば良い。

他のノードから同じメッセージが届いた場合はどうするか。

あり得るのはスロットの切り替わり前後でそれぞれが「自分はリーダー」だと思ってメッセージを送信する場合だ(悪意がない)。今のところPeg-Inは同じユーザ要求に対して同じメッセージになるのだが、Peg-OutはUTXOを自分で選択するPSBTを含んでいるためか同じメッセージにはならない。

分散合意について

ユーザからの要求をどう管理するか。

何かしらの中央データベース的なものを持つか、P2Pでメッセージを伝播させるかしかない。中央に持つのは権限が集中するのと同じになると考えるなら伝播させることになる。

伝播させる方式だと「早い者勝ち」なシステムではうまくいかない(伝播しようとしなくなるだろう)。そういう場合にどうするかというのは「分散合意」という分野になる。

https://chatgpt.com/share/6949ec4b-3350-8001-90cc-d7b1003c4d21

https://grok.com/share/c2hhcmQtMw_64578648-ed27-4360-98d1-d8ebae73b93e

どちらも同じような候補になった。

- RAFT
- ラウンドロビン
- PBFT
- Tendermint / Hotstuff
- VRF
- 提案者＝リーダー

10ノードはまだ数が少ないうちで、20ノードくらいになると中規模。

今回は**ラウンドロビン**で進める。

P2SH形式のマルチシグなら15人がスクリプトサイズ的に上限となるがP2TRでは297人くらいがトランザクションサイズの上限になる。トランザクションサイズ上限＝ブロックサイズ上限なので運用的に無理があるし手数料も高い。15人くらいが上限と考えた方がよいだろう。それ以上となると集約するタイプのアルゴリズムに変更がいる。MuSig2かFROST threshold signatureになるだろう。Ethereumの方はサイズの制約はないのでBitcoinにあわせる。

今回合意を取るのは、ユーザからのPeg-In/Peg-Out要求に対して要求を認めるかどうか。要求を認める＝自分の鍵で署名してリーダー

に返すということになる。リーダーは署名してほしいデータをノードに伝え、集まった署名を使ってスマートコントラクトを呼び出すなりする。

リーダー以外ができる検証
    リーダーが送ってきた署名してほしいデータが妥当かどうか
    リーダーがちゃんと処理したかどうか

署名するデータの妥当性確認はリーダーと同じ情報を持っていないとできない。ブロックチェーンのデータは自分で取得できるとして、ユーザからのリクエストはどうするかというのが最初に述べた中央データベースかP2P伝播かという話につながる。

Peg-In登録
Bitcoinマルチシグアドレスとユーザのの EOAを紐付ける。
その後、ユーザがBitcoinをマルチシグアドレスに送金し、十分confirmしたらリーダーがEthereum側で署名を集めてマルチシグウォレットからユーザのEOAにtransferする。

Peg-Out登録
ユーザのEOAとユーザのBitcoinアドレスを紐付ける。
その後、ユーザがトークンをスマートコントラクトアドレスに送金し、finalizedになったらリーダーがBitcoin側でPSBT署名を集めてマルチシグウォレットからユーザのBitcoinアドレスにsendtoaddressする。

ラウンドロビンの順番
ノードのメンバーを固定にする。入れ替えるにしても合意を取ってから行うので、入れ替え方法はまだ考えていないがとりあえず固定とする。
libp2pのノードIDがあるので、ラウンドロビンの順番はノードID順とする。
では誰がリーダーになっているのかをどうやって決めるのか。

方法はいくつかあると思うが、共有する情報を減らすため 1970年からの秒数(epoch)を適当なタイムスロットに分ける方式にする。そのまま秒数をノード数N で `timestamp %N` などとすると切り替わりが多くなって境界付近でのリクエストに対してリーダーが複数になる可能性があると考えたためだ。
https://chatgpt.com/share/6949ec4b-3350-8001-90cc-d7b1003c4d21
    ノード数が少ないのであれば、方向性として悪くない
    リーダーが複数になる可能性は常にあるので考慮が必要
        リクエストにはスロット番号を付ける。
        TODO同じリクエストであればスロット番号が大きい方のリクエストを受け付ける。小さい番号の方は破棄。
            といっても自分が把握しているスロット番号と大きく逸脱してはダメだろう。
            受け付けるのも危険だが除外するのも迷うというところ。
    最後に自分が合意したスロット番号(`last_committed_slot`)を記憶しておく
        時間がずれて過去スロットの提案を受け付けないようにするため
        リクエストを「自分が」合意した(確定した)スロット番号。みんなと共有するものではない。
        空スロットの場合は進めない。最後のリクエストについてのみ。
        ```
        max(now_slot, last_committed_slot + 1)
        ```

タイムスロットとポーリング
最初はBitcoinチェーンの更新とEthereumチェーンの更新はそれぞれタスクを作って30秒程度の間隔でポーリングしてチェックし、更新されたらそれぞれがメッセージ通知するようにしていた。しかしそれではポーリングの周期とリーダーの周期が一致せずにいつまでも処理を行わないという状態が発生してしまった。
考え方を変え、ポーリングは周期ではなくタイムスロットで自分がリーダーになったときに行うようにした。

リーダー重複に対する考察
タイムスロットの切り替わりにメッセージを受け取った場合、時計の誤差などでスロットの境目に前リーダーと新リーダーが同時に動

いてしまう可能性がある。ような気がする。

- ユーザからのPeg-In登録、Peg-Out登録
  - ユーザからの登録要求を受け取ったノードがP2Pでブロードキャストする。
  - これはリーダーが誰かは関係ない
- Peg-In Bitcoinトランザクション検知
  - 自分がリーダーのときにBitcoinチェーンをチェックし、Peg-In処理を進められるトランザクションを検知したらP2Pで署名リクエストメッセージを展開する。署名が集まったら署名してEthereumチェーンのコントラクトにトークン転送をリクエストする。
  - これはスロットの先頭でしか行わないようにする想定。
  - 自分がリーダーになるまでの間に別のリーダーからP2Pリクエストを受け取れば処理して DBを更新するので、その後でリーダーになっても処理済みと思って重複処理はしない。
- Peg – Out Ethereumトランザクション検知
  - 自分がリーダーのときにEthereumチェーンをチェックし、Peg-Out処理を進められるイベントを検知したらP2PでPSBT署名リクエストメッセージを展開する。署名が集まったら署名してBitcoinチェーンにトランザクションを展開する。
  - これはスロットの先頭でしか行わないようにする想定。
  - 自分がリーダーになるまでの間に別のリーダーからP2Pリクエストを受け取れば処理して DBを更新するので、その後でリーダーになっても処理済みと思って重複処理はしない。

スロットの先頭でしかチェーンのポーリングを行わず、スロット期間が十分な概要であれば何もせずに2つのリーダーがリクエストを展開するようにはならないはずである。
可能性としては、自分が起動する前に現リーダーがリクエストを処理したのでその結果を受け取らず、自分がリーダーになってそのリクエストがまだ残っていると思ってメッセージを展開する場合か？ 前リーダーでリクエストを受け取っていたノードはDBを更新しているので新リーダーのリクエストは無視するから問題なさそう。

タイムスロット時間が足りない

BDKで監視するアドレスやUTXOが増えたり受信したトランザクションが増えるとDBの探索に時間がかかるようになる。
https://claude.ai/share/f6fd1243-0629-43c0-b681-71c16122c267
マルチシグ方式ブリッジのBDK問題で検討したが、これはBDKへの対処だけでは難しいと考えている。可変長のタイムスロットにするなど。

テスト環境

- 実行環境の時計誤差
  - タイムスロットのずれによる動作を評価するため。
- Bitcoin / Ethereum チェーンを分ける
  - foundryのanvilはそういうことができないらしい。Gethなどを使うしかないのだが、GethのCliqueもdeprecatedで今のバージョンでは使えなさそうだ。
    - https://chatgpt.com/share/6965b916-f158-8001-8641-cfde58bc84b2
    - Private networks - Besu documentation
  - Bitcoin Coreはあまりやったことがないができたはず
    - https://chatgpt.com/share/6965b9f6-e9a0-8001-b0a7-5b3833a0fa59