# Bridging BRC-20 to Ethereum

*Abstract*—In this paper, we design, implement, and (partially-) evaluate a lightweight bridge (as a type of middleware) to connect the Bitcoin and Ethereum networks that were heterogeneously uncontactable before. Inspired by the recently introduced Bitcoin Request Comment (BRC-20) standard, we leverage the flexibility of Bitcoin inscriptions by embedding editable operations within each satoshi and mapping them to programmable Ethereum smart contracts. A user can initialize his/her requests from the Bitcoin network, subsequently triggering corresponding actions on the Ethereum network. We validate the lightweight nature of our solution and its ability to facilitate secure and seamless interactions between two heterogeneous ecosystems.

*Index Terms*—BRC-20, Interoperability, Bitcoin, Ethereum

## I. INTRODUCTION

The emergence of Bitcoin revolutionized the field of financial technology by introducing a decentralized network for value transactions. Ethereum further advanced this concept by bringing smart contracts and decentralized applications (DApps). As of July 2023, the market capitalization of Bitcoin is approximately US$584.97 billion, while Ethereum stands at around US$223.32 billion (CoinMarketCap). These cryptocurrencies have not only created significant value themselves but have also paved the way for the development of numerous upper-layer tokens and DApps, which can reach market scales in the thousands. However, the structural differences between Bitcoin's UTXO model and Ethereum's account model have resulted in isolated ecosystems. Users are unable to freely transfer tokens between these heterogeneous blockchains and often rely on external intermediaries, such as centralized exchanges (CEX) or decentralized exchanges (DEX), which come with high costs and limitations. This lack of interoperability hinders the widespread adoption and evolution of these technologies, limiting their full potential.

Existing solutions have made efforts to facilitate interoperability among different blockchains. They often rely on various cryptographic techniques (e.g., zero-knowledge proofs [1] and hash-lock [2] [3]), external hardware (e.g., TEE [4] [5]) or reconstructing the entire system (e.g., Polkadot [6], Cosmos [7]). However, these approaches come with explicit limitations. Cryptographic approaches are computationally intensive and may introduce significant overhead. External hardware solutions like TEEs can be complex and difficult to implement. Reconstruction of the system requires extensive changes, bringing additional assumptions and complexities. As a result, current solutions suffer from various degrees of impracticability, which impede their wide adoption.

**Contributions.** To fill the gaps, we propose an innovative lightweight middleware protocol designed to bridge the gap between Bitcoin and Ethereum. The middleware takes advantage of BRC-20 [8] [9], a standard for digital tokens on the Bitcoin network akin to Ethereum's ERC-20. Our idea is to interpret the BRC-20 operations inscribed on Bitcoin's blockchain and reflect them on the Ethereum network, effectively extending Bitcoin's functionalities within Ethereum's EVM and enabling the possibility of integrating Bitcoin assets in DeFi applications. Specifically, we approach the goal by completing the following steps:

- *We present a lightweight middleware,* MIDASTOUCH (Sec.III), designed to bridge the Bitcoin network and the Ethereum network. MIDASTOUCH enables seamless communication (majorly from Bitcoin to Ethereum) between these networks, empowering users to interact with Ethereum smart contracts through Bitcoin inscriptions defined in the recent BRC-20 standard.

- *We have developed the preliminary version* of MIDASTOUCH (Sec.IV) to demonstrate its functionality. The prototype includes the implementation of functional methods on both the Bitcoin and Ethereum sides and the featured events in intermediate middleware, providing detailed insights into key operations.

- *We conduct a partial evaluation* to assess the effectiveness and efficiency of MIDASTOUCH (Sec.V), focusing specifically on smart contract-related operations on the Ethereum testnet. Our evaluations are based on three key aspects: scalability and performance with varying committee sizes, gas usage for different contract functionalities, and the frequency of validator processing for requests. The results shed insights into the system's behavior under different scenarios, which align with people's intuitive expectations. Additionally, we have discussed the security aspects and potential limitations.

We emphasize two additional aspects in our designs:

- *U shape*. The workflow within our design takes the shape of a "U": users initiate their requests by inputting inscriptions on the Bitcoin network. The action on inscriptions triggers a state transition within Ethereum. Eventually, the Ethereum contract concludes by furnishing a receipt to Bitcoin, serving as a record for the settlement process.

- *Lightweight*. The operation of the validator-maintained middleware does not inherently demand the involvement of supplementary participants. The validators responsible for upkeeping our middleware can either constitute the same group or form a subset of the Ethereum committee.

♠ We metaphorically refer to the task achieved by our middleware as MIDASTOUCH (cf. our title), drawing inspiration from the tale in Greek mythology: *everything King Midas touched turned to gold*, symbolizing a valuable connectivity effect.

## II. Before Construction

### A. Building Blocks

**BRC-20.** This Bitcoin-native standard [8] [9] parallels the Ethereum ERC-20 token standard [10] and signifies a significant shift within the Bitcoin ecosystem, particularly with the emergence of Bitcoin Ordinals. Bitcoin Ordinals revolutionize Bitcoin transactions by assigning an index to each Satoshi (the smallest unit of Bitcoin, 0.00000001) based on its mining order. These indices can be utilized for various purposes, such as unique identifiers or metadata, thereby unlocking new possibilities, including Non-Fungible Tokens (NFTs) [11]. Once a Satoshi has been inscribed (TapScript, around 4MB), it can be utilized to create a BRC-20 token. In essence, the BRC-20 standard enables three primary operations: deploy (creation of a new token type), mint (increasing the supply of the created token), and transfer (trading tokens). We provide a brief overview of each function below, with detailed information available in [12]. These functions collectively enable the creation of a simplified NFT implementation over the Bitcoin network, albeit with some limitations in terms of extensibility.

#### Listing 1: Deploy-Mint-Transfer

```
# Onchain Inscription (Identical items omitted)
"p" : "brc-20" # protocol name

"op": "deploy" # operation
"tick": "ordi" # token name
"max": "2100000" # total amount of token to be issued
"lim": "1000" # maximum amount of token minted each round

"op": "mint" # operation
"amt": "1000" # the amount of token being minted

"op": "transfer" # operation
"amt": "100" # the amount of token being transferred

# Off-chain update (mapping to operations)
"deploy":
if state[tick] NOT exists:
    state[tick]={<inscription info>,
                "balances": {addr: balance_val}}

"mint":
if state[tick] NOT exists OR
                "amt" > "lim" OR sum("amt") > "max":
    raise errors
else
    account_state[tick]["balance"][minter] += amt

"transfer":
if state[tick] NOT exists:
    raise errors
if state[tick]["balance"][sender] >= amt:
    account_state[tick]["balance"][sender] -= amt
    account_state[tick]["balance"][receiver] += amt
```

Due to the shortage of formal research on BRC-20, Rodarmor [13] was one of the first to introduce a scheme for assigning serial numbers to Bitcoin satoshis, and a comprehensive introduction to ordinal theory can be found in [14]. Additionally, Binance Research has published several pioneering reports [9] [15] [16] that explore the development of BRC-20. Bertucci [17] conducted an early analysis of transaction fees related to ordinal inscriptions.

**Smart contract.** A smart contract (SC) is a distinct form of contract where the agreement's terms are directly encoded into executable code. Operating as a self-contained *white-box*, a smart contract guarantees the synchronization of input and output, effectively eliminating the reliance on trustworthy third-party intermediaries [18]. Deployed primarily on blockchain platforms such as Ethereum, smart contracts are executed automatically once predetermined conditions encoded within the code are fulfilled. The versatility of smart contracts enables automation across diverse domains, spanning from financial transactions [19] and governance systems [20] to decentralized organizations [21] and beyond. With their ability to enforce transparent and trustless transactions, smart contracts offer enhanced efficiency, security, and persistence.

**Ethereum token standards.** Tokens play a vital role in incentivizing users and developers within blockchain ecosystems. These tokens adhere to specific standards, which define the methods for creating, deploying, and issuing new tokens. Ethereum, with its robust smart contract capabilities, has established itself as a leader in token standards [22], driving versatile applications within its ecosystem. The ERC-20 fungible token standard [10] has gained significant traction, leading to the proliferation of ICOs [23] and a flourishing token market [24]. However, different blockchain ecosystems employ incompatible token standards. For instance, a token adhering to the BRC-20 standard on the Bitcoin network cannot be utilized on the Ethereum network. This limitation has motivated us to explore the construction of a potential connection between these disparate ecosystems.

### B. Concurrent Solutions

**Interoperability in blockchain.** Polkadot [6] enables the interconnection of subnetworks (the relay chains) through the cross-chain message passing protocol (XCMP). Within this context, a relay is a smart contract residing in a target blockchain that operates as a lightweight client of a source blockchain. Cosmos [7] achieves cross-chain communications via the inter-blockchain communication protocol (IBC) [25]. IBC is designed to consist of two major layers that establish secure connections for data transportation (*a.k.a.* TAO) and defines the way of packaging data (APP layer). However, these solutions are restricted to facilitating interoperability among blockchains within the same ecosystem. Cacti [26] is an integral part of the Hyperledger project. The scheme relies on a network of interoperable validators that validate cross-chain transactions and are entrusted with the task of signing them. To ensure the validity of transactions, a consensus among a quorum of validators is required for their successful signing. Hermes [27] is a middleware for blockchain interoperability that is built on the Open Digital Asset Protocol (ODAP) (recently merge into SATP [28]). The protocol draws inspiration from the two-phase commit protocol (2PC) [29] and goes beyond that by incorporating a flexible log storage API that provides multiple storage options, including local, cloud, and on-chain storage. CheaPay [30] and Herdius [3] focus on a payment channel network that enables off-chain settlement of transactions between blockchains by utilizing the Hash Time-Lock Contract (HTLC) scheme to ensure atomic swaps of

assets. Tesseract [4] is an exchange protocol that operates in real-time and utilizes trusted hardware as a reliable relay. It facilitates the tokenization of assets and enables the pegging of these assets to cryptocurrencies. Similar solutions also leverage TEEs [5] [31] to perform cross-chain transactions. More classifications on interoperability can refer to [32] [33].

TABLE I: Mainstream interoperable blockchains

| Projects | Communication | Architecture | Witness | Implementation |
|----------|---------------|--------------|---------|----------------|
| Polkadot | XCMP | Parachains | Relay chain (SC) | Substrate |
| Cosmos | IBC | Hybrid (TAO/APP) | Relayers | Tendermint |
| Hermes | ODAP(-2PC) | Gateway-based | Middleware | - |
| Hyperledger | Trusted party | Hybrid | Validators | Cactus |
| CheaPay | Sidechain | Layer-2 | Hash-lock | - |
| Herdius | Sidechain | Layer-2 | Hash-lock | - |
| Tesseract | Trusted hardware | Hybrid | TEE | (Exchange) |

**Selection of technical routes.** The presence of reliable witnesses is crucial for the successful implementation of a dependable interoperable protocol, especially in ensuring the *all-or-nothing* settlement of digital assets, also known as atomic swaps. Existing solutions, as we have discovered through our investigation, rely on either trusted parties, such as relayers and validators, or automated algorithms/machines like smart contracts, middleware, TEEs, and hash-locks, to achieve reliable witnesses. However, relying on trusted parties poses a significant risk of compromise. Therefore, we have chosen the alternative route. Nonetheless, Hash-locks are actually a common construct used in atomic cross-chain swap protocols (e.g., [34]) which require strict network requirements, while TEE-based solutions tend to be complex. As a result, we have been motivated to develop a contract-based middleware that serves as an efficient bridge between Bitcoin and Ethereum, providing the desired functionality.

### C. Threat Model and Assumption

**Blockchain model.** We assume the blockchains, both applicable to Bitcoin and Ethereum chains, consist of a group of nodes that operate the system. In our analysis, we simply consider a fraction of nodes that may behave arbitrarily, but the total number of these unfaithful nodes is below the security threshold of the consensus protocols (e.g., less than 50% in PoS/PoW settings). The blockchain system adheres to the robust assumption as established by previous research [35].

- *Consistency.* Once an honest node commits transaction $tx_1$ before $tx_2$, no honest node will ever commit $tx_2$ before $tx_1$.
- *Liveness.* Once a valid transaction is submitted to an honest node, it will be eventually committed on-chain.

**Smart contract model.** By integrating the fundamental functionalities offered by underlying blockchain systems, we present a simplified abstraction of the key features to simulate a smart contract.

- *Contract deployment.* The contract is deployed on-chain, establishing its presence within the blockchain network.
- *State update.* The state undergoes transitions triggered by input transactions, evolving as new transactions that are packed into the new proposed blocks.

- *State consensus.* Blockchain maintainers execute consensus to reach an agreement on the global view of the state, ensuring consistency among distributed nodes.
- *State query.* With a confirmed state, users can retrieve specific transactions and blocks at a given height for analysis or reference.

**Cryptography primitives.** We require conventional unforgeability for digital signatures (multi-signature [36] included) and collision-resistant hash functions [37].

**General honesty assumption.** We make the assumption that the majority of group members in our described systems, whether they belong to the blockchain networks (such as Bitcoin and Ethereum) or the random validator committee, will faithfully adhere to the designated tasks.

### III. MIDDLEWARE CONSTRUCTION

### A. Technical Challenges

**Challenge-I: Managing different data formats within heterogeneous blockchains is a non-trivial task.** The primary challenge lies in reconciling the stateless UTXO-based model of Bitcoin with Ethereum's stateful, account-based approach. To address this, we propose leveraging the BRC-20 standard as a middleware to establish a lightweight bridge.

In our implementation, BRC-20 utilizes inscriptions to record the state transitions of UTXO states. These inscriptions serve as verifiable proofs and are used to trigger smart contract functions/events. We incorporate a series of operation indicators within the inscriptions and provide corresponding events in the smart contract. This allows users to initiate transactions on the Bitcoin network, which in turn triggers changes in the inscriptions and subsequent state transitions in the smart contracts on the Ethereum network.

**Challenge-II: Determining which side should bear the deduction of fees can give rise to debatable arguments.** State-of-the-art cross-chain solutions often overlook the costs involved in exchanges, which is a crucial aspect that users are concerned about.

In our approach, the actions are initiated from the Bitcoin side, and the actual transactions are triggered on this network, leading to corresponding state transitions on the Ethereum side. Consequently, users on the Bitcoin side are responsible for bearing the associated exchange fees, which are separate from (equiv. in addition to) the basic transaction fees incurred during the consensus procedures.

**Challenge-III: Implementing cross-chain transactions may pose significant complexity.** Existing schemes often rely on a series of complex cryptographic operations (e.g., [4]) or the reconstruction of intricate systems (e.g., [6] [7]). Unfortunately, this level of complexity renders the system impractical for widespread adoption and use.

Rather than introducing complex dependencies, our approach focuses on establishing a lightweight middleware that seamlessly bridges actions initiated on the Bitcoin side with state transitions on the Ethereum side. Our implementation

leverages the native editable field in Bitcoin, as defined by the BRC-20 standard, and programmable functions written in smart contracts. MIDASTOUCH can work harmoniously with both blockchains, ensuring smooth interoperability without the need for additional intricate dependencies.

### B. Warm-up Construction

**Roles.** The protocol includes four roles: *transaction orginator*, *contract owner*, *validator*, and *operator*.

- *Transaction originator* (Bitcoin). The Bitcoin transaction originators are the users initiating transactions on the Bitcoin network. Their main role is to inscribe the transaction with specific information regarding Ethereum contract interactions, such as contract address and operation data. This inscription is embedded within Bitcoin transactions and is scanned by validators.
- *Contract owner* (Ethereum). The Ethereum contract owners control the smart contracts on the Ethereum network, which the middleware protocol interacts with. They define the contract operations that can be invoked through inscriptions on the Bitcoin network. Furthermore, they monitor the state updates broadcast by validators.
- *Validator* (middleware). The validators are responsible for the accurate execution of the middleware protocol. Their duties include registering themselves on the list, validating transactions from the Bitcoin network, and managing the update of Ethereum contract states. They also participate in consensus processes. Notably, validators have to deposit an amount of money in the contract.
- *Operator* (middleware). The operators are responsible for setting up and maintaining the middleware protocol. They set the system parameters, such as the size of the validator committee, the block intervals for consensus and state updates, the rules for multi-signature validation, and other security features. They also take care of system upgrades.

Note that in a typical case, middleware *validators*, *operators*, and the Ethereum *contract owner* can be played by the same group of users, wherein settings established by the middleware *operator* are commonly predefined in the genesis or a designated checkpoint block.

**System overview** (Fig.1). For the initial setup, the contract developer deploys a domain-specific smart contract on Ethereum. In this protocol, each token does not have its own smart contract. Smart contracts are organized based on functionality, where each functionality (such as the auction function in DeFi) has its own smart contract. Each contract records the state information of all tokens that use that particular functionality.

Then, ① any Bitcoin user who is keen to become a validator of the middleware layer will deposit a certain amount of ETH to this smart contract with a valid Ethereum address that is bounded with his Bitcoin address. Until enough validators get registered (while the validator committee size is predefined and can be ranged from one to multiple), the system initialization is completed. The committee will take charge of all interaction with the related smart contracts, on behalf of the
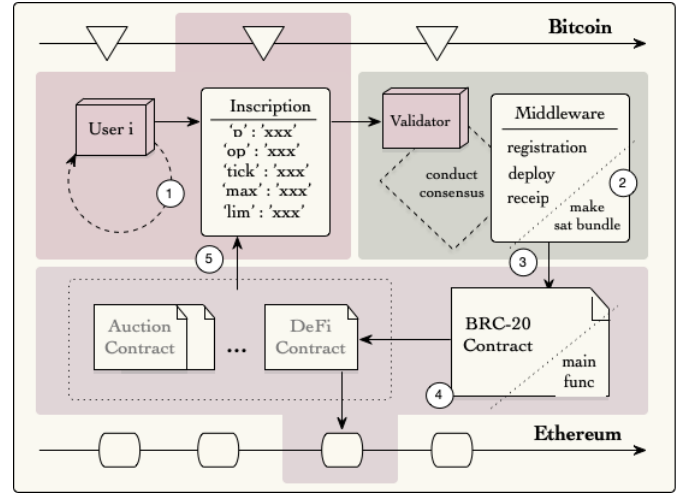


Fig. 1: System overview

Bitcoin transaction originators who thus do not need to own any Ethereum addresses. A user sends an inscribed satoshi that claims the validity of any functions (equiv. operations). This sat should contain a script (TapScript). The functions, formatted as *func_signature* (as in the example), need to match the unified interfaces defined in corresponding smart contracts. ② The committee consistently monitors the Bitcoin network and periodically collects the inscribed satoshi (acts as an explorer), sorts them as per the timestamp, and constructs a sat bundle. ③ For every increment of $\varepsilon$ block heights in the Bitcoin network, the committee engages in a consensus process related to the sat bundle. Following this, it invokes the respective Ethereum network's smart contracts using the contract addresses corresponding to each sat bundle. ④ The protocol employs a multi-signature approach to update the state in the smart contracts. Auditing is processed to ensure that the penalty is properly applied (by slashing the deposit) to any misbehaviors. Meanwhile, the gas fee awarded to validators will be deducted from the satoshi bundle with a certain percentage, e.g., 5%. Note that the gas fee will be calculated for each individual satoshi. ⑤ Finally, the committee gathers the emitted events from the invoked contracts and broadcasts the post-operation inscriptions on the Bitcoin network. These broadcasts act as receipts for the executed bundle, signaling the completion of the originated inscription requests.

Listing 2: Example format

```
"operation": name # matched to interfaces in contracts
"smart contract address": xxx # Ethereum
"deposit": xxx ETH # Ethereum token
...
```

### C. Detailed Construction

- **Validator registration.** The MIDASTOUCH protocol initiates with defining core parameters and proceeds to the validator registration phase. Validators are required to register and deposit a specified amount of ETH into a designated deposit contract. The registration is inscribed on the Bitcoin network, after which the newly

registered validator is added to the validator set. The size and requirements of the validation committee can vary significantly based on the desired level of system security. For instance, a system requiring high security might necessitate a large committee and more intricate consensus mechanisms. Conversely, a system prioritizing efficiency over security might operate with a smaller committee or even a single validator.

- **Inscription-contract interactions.** Once the validator committee is established, the middleware protocol begins managing transactions from the Bitcoin network. For each output in every transaction of the newly obtained Bitcoin block, it searches for potential inscriptions. Valid inscriptions are added to the *inscription bundle set* $\mathbb{B}$, and the corresponding contract addresses are accumulated into the contracts set in terms of different functionalities.

- **State update.** The consensus process (if the validator committee size reaches the lower bound for consensus) and state update occur at predetermined block intervals. During this consensus process, the inscriptions bundle is sorted based on the timestamp, and validators reach a consensus on the legitimate inscriptions. The system then fetches the latest state for each contract in the set from the Ethereum network, which commonly include a balance record for each unique Bitcoin address associated with various tokens as an entry point for handling necessary operations upon the token amounts.

- **Multi-signature validation.** The protocol processes each inscription within the bundle, subtracting the gas fee from each and distributing it among validators based on their respective Bitcoin addresses. If the operation within the inscription proved valid in the Bitcoin network, it is executed with multi-signature validation, leading to the state and address balance updates in the Ethereum network. The degree of validation and the consensus mechanism used for this process can be adjusted according to the security requirements of the system.

- **Inscription publication.** After processing all contracts, validators republish the outcomes of operations as inscriptions back to the Bitcoin network. The block index is incremented, indicating the protocol's readiness to manage the next Bitcoin block.

## IV. IMPLEMENTATION

### A. Basic Operations

We provide three concrete instances to clarify the proposed MIDASTOUCH protocol. The operations cover registration, token deploy, and receipt production.

- Registration. The operation includes details such as the *protocol name*, *operation* (registration), *signature*, *token name*, *deposit amount*, and *Ethereum address*. Following the inscription, an update occurs on Ethereum. If the Ethereum address does not exist in the validator set, it is added with the associated Bitcoin address and balance information. The connection between Bitcoin and Ethereum addresses forms the backbone of the middleware.

Listing 3: Registration

```
# Bitcoin Inscription
"p" : "middleware", # protocol name
"op": "registration", # operation
"op_signature": "registration(...) return (...)",
            # operation signature
"tick": "eth", # token name of deposit
"max": "32", # total amount of deposit
"c_addr": <eth_addr> # Ethereum contract address of
                    # the designated deposit contract
"eth_addr": <eth_addr>
            # Ethereum address used by a validator
            # in registration

# Ethereum update
# Contract_address: c_addr
function registration(...) return (...) {
    ...
    if validator[eth_addr] NOT exists:
        validator[eth_addr]={"btc_addr": <btc_addr>,
                "balances": <balance_val>} }
```

- Deploy. The operation represents issuing a new token on the Ethereum network. The Bitcoin inscription includes information including the *protocol name* (in this case, the BRC-20 token standard), *operation* (deploy), *operation signature*, *token name*, *total token supply*, and the *upper bound for tokens* to be minted in each round. The Ethereum network responds to this inscription by creating a new corresponding token state if it does not exist. This state contains the inscription information and an initial balance state for the token addresses.

Listing 4: Deploy

```
# Bitcoin Inscription
"p" : "brc-20", # protocol name
"op": "deploy", # operation
"op_signature": "deploy(...) return (...)", # signature
"tick": "ordi", # token name
"max": "2100000", # total amount of tokens to be issued
"lim": "1000" # maximum amount of token being minted
"c_addr": <eth_addr> # Ethereum contract address of
                    # the targeted contract

# Ethereum update
# Contract_address: c_addr
function deploy(...) return (...) {
    if state[tick] NOT exists:
        state[tick]={<inscription info>, "balances":
                    {addr: balance_val}} }
```

- Receipt. The operation represents the closure of an Ethereum event cycle and its report back to the Bitcoin network to guarantee the finalization of the originated inscription requests such as the above Deploy inscription. On the Ethereum side, after a function defined in the smart contract is executed, events are emitted. These events are captured by validators, who then publish an inscription on the Bitcoin network, signifying an operation of receipt. This inscription includes the *protocol name* and a collection of *events*, each corresponding to an *inscription ID* and carrying information about the operation's *execution results* (e.g., true/false, values). Unlike other algorithms, there is no "op_signature" in this case, as this algorithm is simply forwarding the Ethereum events back to the Bitcoin network without executing a particular operation itself. With this operation, only those inscription requests which are included in an receipt will be committed as a success and ready to be further used.

## Listing 5: Receipt

```
# Ethereum events emitted
# Contract_address: <eth_addr>
function <func_signaure> {
    ...
    emit();
}

# Bitcoin Inscription
"p" : "middleware", # protocol name
"op": "receipt", # operation
"events": {inscription_id: (<t/f>, <return_value>)}
```

### TABLE II: Notations

| Symbol | Meaning | Scope |
|---|---|---|
| $\mathbb{V}$ | validator set, instantiated by $v_i$ | MIDASTOUCH |
| $\mathbb{S}$ | state set, instantiated by $s_i$ | Ethereum |
| $\mathbb{C}$ | smart contact set, instantiated by $c_{addr}$ | Ethereum |
| $\mathbb{B}$ | inscription bundle set | Bitcoin |
| $c_{addr}$ | contract address/identifier | Ethereum |
| $\Lambda$ | receipt set | Bitcoin/Ethereum |
| $\Psi$ | address balance | Bitcoin |
| $H$ | block height/index | Bitcoin/Ethereum |
| $p$ | penalty rate | Bitcoin |
| $g$ | gas fee | Ethereum |
| $\varepsilon$ | constant value | Bitcoin/Ethereum |
| $ins$ | short for inscriptions | Bitcoin |
| $tx$ | short for transactions | Bitcoin/Ethereum |
| $\phi$ | validator mapping topology | Bitcoin/Ethereum |

### B. Algorithms

We implement and present the major workflow of MIDAS-TOUCH (cf. Algorithm 1), featured with a multi-party validator committee. Notations are listed in Table II.

**Become committee members.** Becoming an eligible committee member requires CommiteeRegistration to be involved. The CommiteeRegistration function is responsible for registering validators who hold both a Bitcoin and Ethereum address and have deposited a specific amount of ETH into a contract. This registration is then inscribed onto the Bitcoin network, and the newly registered validator is added to the validator set $\mathbb{V}$. The function also confirms the successful registration of these validators and ensures the completeness and correctness of the information provided during the registration process.

**Action on Bitcoin.** The primary function involved in the actions on the Bitcoin network is HandleInscription. This function scans each transaction output from the new Bitcoin block for potential inscriptions. Valid inscriptions are appended to the transaction bundle $\mathbb{B}$, and the corresponding contract addresses are accumulated into the contracts set $\mathbb{C}$. Additionally, HandleInscription is also responsible for broadcasting post-operation inscriptions, which serve as receipts for the executed transactions, indicating their completion. This dual functionality ensures that all potential inscriptions are evaluated for validity and corresponding receipts are issued, keeping the system secure and transparent.

**Action on Ethereum.** The main function involved in the actions on the Ethereum network is UpdateEVMState. This function is responsible for retrieving the most recent state $\mathbb{S}$ for every contract $c$ within the set $\mathbb{C}$ from the Ethereum network. For contracts related to BRC-20 or those possessing similar token-managing functionalities, it additionally retrieves a balance record $\Psi$ for each distinct Bitcoin address associated with various tokens. During the consensus process that occurs every $\varepsilon$ block, this function processes each inscription within the bundle $\mathbb{B}$, distributing the gas fee $g$ among validators based on their respective Bitcoin addresses and updating the state $\mathbb{S}$ and address balance $\Psi$ via a multi-signature validation process. Furthermore, UpdateEVMState oversees the gathering of emitted events from the invoked contracts on Ethereum. These events are later broadcasted on the Bitcoin network as part of the receipt operations, marking the completion of inscription.
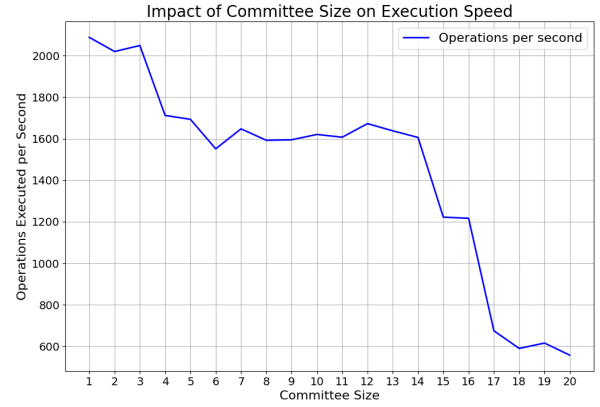


Fig. 2: Evaluation on scalability

## V. EVALUATION AND ANALYSIS

### A. Performance Analysis

**Scalability.** Fig.2 presents a detailed visualization of the impact of the (validator-) committee size on the execution speed of our proposed MIDASTOUCH protocol. The x-axis represents the size of the committee, which ranges from 1 to 20 members[1]. The y-axis illustrates the number of operations that can be executed per second. We utilize the well-known Practical Byzantine Fault Tolerance (PBFT) [38] protpcol for our committee and factor in the transaction processing capabilities of both Bitcoin and Ethereum networks. Specifically, we consider Bitcoin's Lightning Network which can process up to the order of 10,000 transactions per second [39], and Ethereum 2.0 where the Casper-PoS [40] [41] and sharding technology are enabled, capable of handling 64 times higher transactions per second than the single-sharded Ethereum when the projected Phase-1 is activated. Given these parameters, the operational speed of MIDASTOUCH cannot surpass the minimum throughput of these two networks.

As depicted in Fig.2, the execution speed corresponds to Ethereum 2.0's average throughput until a committee size of 4, the minimum requirement for consensus in our configuration.

---

[1]A validator committee with a size smaller than 4 is regarded as a central entity with no consensus process being done.

**Algorithm 1:** The MIDASTOUCH Protocol

```
1  Initialization
2  H : block index, V : validators ← ∅, S : state ← ∅, Ψ : Bitcoin's balance ← ∅,
   p : penalty rate, g : gas fee rate, B : Bundle, C : Contracts ← [], [];
3  Procedure MainFunc ()
4      CommitteeRegistration ();
5      while True do
6          Block ← V.GetBitcoinBlock(H);
7          foreach tx : transaction in Block.tx do
8              foreach o : output in tx.outputs do
9                  ins : inscription ← V.ParseInscription(o);
10                 if ins ≠ None then
11                     B.append(HandleInscription(ins));
12                     C.append(HandleInscription(ins)[c_addr]);
13                 end
14             end
15         end
16         if H mod ε == 0 then
17             B ← V.Sort(B, key=timestamp).ConsensusProcess(B, p);
18             foreach c_addr in C parallelly do
19                 S, Ψ ← V.GetStateInfo(c_addr), V.GetBalance(c_addr);
20                 S, Ψ, True/False ← UpdateEVMState(B, S, Ψ, g);
21                 (True/False) events of c_addr are cached in the receipt set Λ;
22             end
23             V.PublishInscription(Λ), each event referred to its own ins;
24             B : Bundle, C : Contracts ← [], [];
25         end
26         H ← H + 1;
27     end
28 end
29 Procedure CommitteeRegistration ()
30     while V.count() < min_committee_size do
31         φ : (map[BTC_addr] ⇔ ETH_addr) ← V.WaitForRegistration();
32         ins : inscription ← V.ParseInscription(φ.BTC_addr.tx);
33         if V.GetDeposit(φ.ETH_addr) ≥ k(ETH) in a dedicated deposit contract
           and ins is for registration and finalized then
34             V.append(φ);
35         end
36     end
37 Procedure UpdateEVMState (B, S, Ψ, g)
38     True/False ← [], []
39     foreach ins : inscription in B do
40         gas_fee ← g × ins[value];
41         ins[value] ← ins[value] − gas_fee;
42         foreach φ : validator in V do
43             Ψ[ins[c_addr]][φ.BTC_addr][ins[tick]] ←
               Ψ[ins[c_addr]][φ.BTC_addr][ins[tick]] + gas_fee;
44         end
45         if ins[operation] ∈ C[ins[c_addr]].GetInterfaces() then
46             Invoke ins[operation] with multi-sig validation;
47         end
48         True.append(ins) if succeeded else False.append(ins);
49     end
50     return S, Ψ, True/False;
51 Procedure HandleInscription (ins : inscription)
52     op ← ins[operation], value ← ins[values] ;
53     c_addr ← ins[c_addr], others ← ins[< others >];
54     return op, c_addr, value
55 end
```
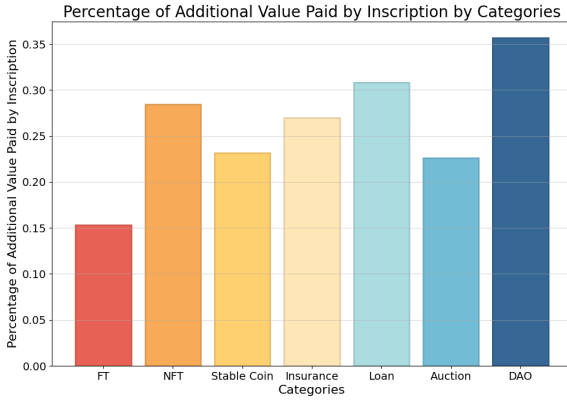


Fig. 3: Evaluation on gas consumption for finalization

After this point, the speed decreases non-linearly with an increase in committee size due to the quadratic time complexity ($O(n^2)$) of the PBFT, where $n$ represents the number of nodes. This illustrates that the choice of committee size presents a balancing act between decentralization and performance. While larger committees yield increased decentralization, they compromise on operational speed.

**Gas consumption.** We evaluate the additional amount of gas that any inscription needs to pay for validators in terms of the different functionalities of smart contracts running on the Ethereum network. Note that the overhead of sending an inscription is small and can be negligible when compared with the execution of smart contracts due to which an incentive is required to be applied. We consider the gas consumption in typical contracts regarding each functionality:

- FT (fungible tokens [10]). The simplest type of smart contract typically involves just the transfer of tokens from one address to another.
- NFT (non-fungible tokens [11]): NFT contracts can be complex due to the involvement of metadata handling, uniqueness verification, or royalty payment mechanisms.
- Stablecoin: Generally simple as well, but some additional complexity for pegging the value to an asset.
- Insurance: It can get complicated depending on the terms of the insurance policy and the type of risks it covers.
- Loan: Loan contracts can be complicated. They usually require mechanisms to handle interest calculation, risk assessment, and loan recovery.
- Auction: Auction contracts need to manage bids from multiple participants, which adds complexity.
- DAO: These are the most complex types of contracts involving governance, voting mechanisms, fund management, or interacting with many other types of contracts.

Specifically, the percentage of additional value paid by inscription across various categories of smart contracts is presented in Fig.3, using representative types as examples. FT requires the least additional value, reflecting their relatively straightforward functionality of merely transferring tokens. In contrast, DAOs [42], with their intricacies involving governance, voting mechanisms, and fund management, demand the highest percentage. NFTs, Loans, and Auctions lie in the middle ground. NFT contracts' complexity arises from handling metadata and verifying uniqueness, while Loan contracts necessitate mechanisms for calculating interest, assessing risk, and recovering loans. Auction contracts, owing to their need to manage multiple participants' bids, also require a substantial additional value. It is noteworthy that the additional value percentages are influenced by the inherent complexity and
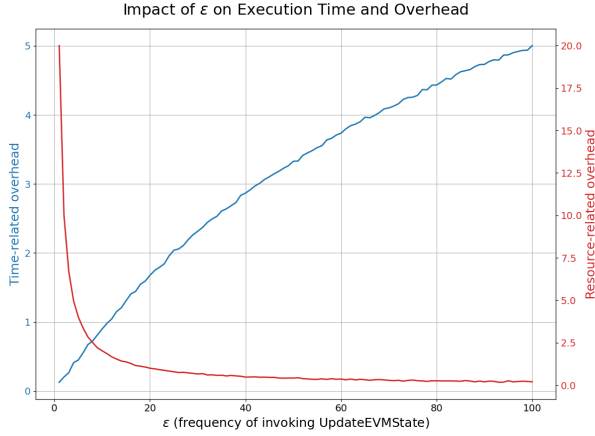
Fig. 4: Evaluation on different numbers of inter blocks

functionality of the respective smart contracts. This insight underscores the necessity for efficient management of gas consumption in order to maximize the overall system efficiency.

**Frequency of checking.** We further explore the influence of the parameter $\varepsilon$, which dictates the frequency of invoking the UpdateEVMState operation in terms of Bitcoin block heights, on the efficiency of the MIDASTOUCH protocol. Fig.4 demonstrates two distinctive aspects of system overhead: time-related overhead, and resource-related overhead, which are associated with the execution time and the computational resources required, respectively.

When $\varepsilon = 1$, the validator committee is obligated to scrutinize every Bitcoin block, extract the inscriptions from transactions, assemble them into a bundle, arrange them by timestamp, and finally update the corresponding Ethereum smart contracts. As an alternative, the system can postpone the update of Ethereum's smart contract state until every $\varepsilon$ Bitcoin block heights have been processed, amassing a substantial number of sorted inscriptions in the bundle during this interval.

In addition, both the time-related and resource-related overheads are affected by the choice of $\varepsilon$. Specifically, as $\varepsilon$ increases, the time-related overhead decreases gradually, demonstrating that accumulating more transactions before updating the EVM state can save execution time. However, this comes at the cost of an increase in resource-related overhead, likely due to the need for storing and sorting a larger number of inscriptions. The ideal value of $\varepsilon$ would therefore be a trade-off between these two factors, balancing the need for quick execution with the capacity of the system's available resources.

### B. Primary Security Analysis

**Safety (settlement).** In our context, the appropriate definition of safety extends beyond conventional distributed systems definitions that primarily focus on state consistency. Here, safety implies that a request is fully executed on both sides and returns the correct value. To achieve this, we primarily ensure settlement: a request should be treated as an indivisible unit that can reach the final state on both sides. This guarantees that the protocol remains in a consistent state and prevents

the fraudulent creation of additional values or unwarranted destruction of legitimately existing values within the Ethereum network. The complete lifecycle in our protocol is marked by two Bitcoin transactions: the first transaction (where an inscription request is included) serves as a trigger, initiating a series of events; and the second transaction acts as a receipt, indicating the successful completion of all associated events.

Firstly, the unidirectional invoking nature of the MIDAS-TOUCH protocol guarantees that Bitcoin users can successfully complete token transfers from the origin address to the designated address. This indicates that the operations recorded in the inscription can be invoked. Given our assumption that the majority of validators are honest, these operations will faithfully transit through our channel. Upon reaching the smart contract, the operations are processed on the Ethereum chain based on the endorsement of the majority of validators through multi-signature verification. Secondly, after the execution of operations, receipts are issued and broadcasted on the Bitcoin network to guarantee the finalization of the originated inscription requests included in the executed bundle, providing an additional layer of security and transparency.

Furthermore, the MIDASTOUCH protocol does not possess the ability to externally deduct digital assets from either side. Transactions on Bitcoin are initiated by users, while the legal invocation of a contract requires the majority of validators. Any misbehavior will be rejected through internal consensus procedures. Thus, the addition of the receipt operation ensures the protocol's safety and settlement, offering conclusive evidence of successful transaction executions.

**Liveness.** The liveness property of the system ensures that it remains continuously available without the risk of sudden shutdown. In the context of our protocol, this property signifies that any actions performed within the Bitcoin network, such as transactions or inscriptions, will eventually be reflected in the Ethereum network. This property relies on the correct functionality of the majority of validators, which we assume to be guaranteed (our assumptions in Sec.II-C).

**Fairness.** The property ensures equality among valid transactions, prohibiting any discrimination. This indicates any Bitcoin transaction conforming to protocol rules and having paid the necessary gas fee will be finally reflected in Ethereum without bias. Fairness can be achieved through the settlement of operation processing on both sides.

## VI. CONCLUSION

Current Bitcoin and Ethereum are isolated due to their heterogeneous chain structure. In this work, we propose a lightweight one-way middleware, named MIDASTOUCH, to bridge the Bitcoin and Ethereum networks. We employ the notion of the newly proposed BRC-20 standard to incorporate a range of operations into each satoshi and associate them with specific events within Ethereum smart contracts. We implement a prototype of MIDASTOUCH and evaluate the performance from the Ethereum side. Evaluation results demonstrate practicability and efficiency. To our knowledge, this is the first attempt to expand the capabilities of BRC-20.

## REFERENCES

[1] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022, pp. 3003–3017.

[2] S. Wadhwa, J. Stöter *et al.*, "He-HTLC: Revisiting incentives in HTLC," *Network and Distributed System Security (NDSS)*, 2023.

[3] D. Balazs, "Herdius whitepaper," *https://herdius.com/whitepaper/Herdius_Technical_Paper.pdf*, 2017.

[4] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 1521–1538.

[5] Y. Lan, J. Gao, Y. Li, K. Wang, Y. Zhu, and Z. Chen, "Trustcross: Enabling confidential interoperability across blockchains using trusted hardware," in *International Conference on Blockchain Technology and Applications (ICBTA)*, 2021, pp. 17–23.

[6] G. Wood, "Polkadot: Vision for a heterogeneous multi-chain framework," *White paper*, vol. 21, no. 2327, p. 4662, 2016.

[7] J. Kwon and E. Buchman, "Cosmos whitepaper," *A Netw. Distrib. Ledgers*, vol. 27, 2019.

[8] Anonymous, "BRC-20 experiment," *https://domo-2.gitbook.io/brc-20-experiment/*, 2023.

[9] B. Research, "BRC-20 tokens: A primer," *https://research.binance.com/static/pdf/BRC-20%20Tokens%20-%20A%20Primer.pdf*, 2023.

[10] V. Fabian and B. Vitalik, "ERC-20: Token standard," *https://eips.ethereum.org/EIPS/eip-20*, 2021.

[11] Q. Wang, R. Li *et al.*, "Non-fungible token (NFT): Overview, evaluation, opportunities and challenges," *arXiv preprint arXiv:2105.07447*, 2021.

[12] Q. Wang and G. Yu, "Understanding BRC-20: Hope or hype," 2023.

[13] C. Rodarmor, "Ordinal numbers," *https://github.com/casey/ord/blob/master/bip.mediawiki*, 2022.

[14] "Ordinal theory handbbok," *https://docs.ordinals.com/*, 2023.

[15] B. Research, "A new era for Bitcoin," *https://research.binance.com/static/pdf/a-new-era-for-bitcoin.pdf*, 2023.

[16] ——, "Monthly market insights," *https://research.binance.com/static/pdf/monthly-market-insights-2023-06-.pdf*, 2023.

[17] L. Bertucci, "Bitcoin ordinals: Determinants and impact on total transaction fees," *Available at SSRN 4486127*, 2023.

[18] R. Li *et al.*, "How do smart contracts benefit security protocols?" *arXiv preprint arXiv:2202.08699*, 2022.

[19] S. M. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. J. Knottenbelt, "SoK: Decentralized finance (DeFi)," *ACM Advances in Financial Technologies (AFT)*, 2021.

[20] A. Kiayias and P. Lazos, "SoK: Blockchain governance," *ACM Advances in Financial Technologies (AFT)*, 2022.

[21] G. Yu, Q. Wang, T. Bi, S. Chen, and X. Xu, "Leveraging architectural approaches in web3 applications-a DAO perspective focused," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2023, pp. 1–6.

[22] Q. Wang, R. Li *et al.*, "Exploring Web3 from the view of blockchain," *arXiv preprint arXiv:2206.08821*, 2022.

[23] D. A. Zetzsche, R. P. Buckley, D. W. Arner, and L. Föhr, "The ICO gold rush: It's a scam, it's a bubble, it's a super challenge for regulators," *University of Luxembourg Law Working Paper*, no. 11, pp. 17–83, 2017.

[24] A. Tapscott and D. Tapscott, "How blockchain is changing finance," *Harvard Business Review*, vol. 1, no. 9, pp. 2–5, 2017.

[25] "Inter-blockchain communication protocol," *https://tutorials.cosmos.network/academy/3-ibc/1-what-is-ibc.html*, 2023.

[26] "Hyperledger cacti," *https://github.com/hyperledger/cacti*, 2023.

[27] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono, "Hermes: Fault-tolerant middleware for blockchain interoperability," *Future Generation Computer Systems (FGCS)*, vol. 129, pp. 236–251, 2022.

[28] H. Martin, H. Thomas, and B. Rafael, "Secure asset transfer protocol (SATP)," *https://datatracker.ietf.org/group/satp/about/*, 2023.

[29] P. A. Bernstein, V. Hadzilacos *et al.*, *Concurrency control and recovery in database systems*. Addison-wesley Reading, 1987, vol. 370.

[30] Y. Zhang *et al.*, "Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks," in *IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[31] Z. Yin, B. Zhang, J. Xu *et al.*, "Bool network: An open, distributed, secure cross-chain notary platform," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 17, pp. 3465–3478, 2022.

[32] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–41, 2021.

[33] G. Wang *et al.*, "Exploring blockchains interoperability: A systematic survey," *ACM Computing Surveys (CSUR)*, 2023.

[34] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the ACM symposium on Principles of Distributed Computing (PODC)*, 2018, pp. 245–254.

[35] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 2015, pp. 281–310.

[36] K. Itakura, "A public-key cryptosystem suitable for digital multisignature," *NEC research and development*, vol. 71, pp. 1–8, 1983.

[37] I. B. Damgård, "A design principle for hash functions," in *International Conference on the Theory and Application of Cryptology (ASIACRYPT)*. Springer, 1989, pp. 416–427.

[38] C. Miguel and L. Barbara, "Practical byzantine fault tolerance," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1999, pp. 173–186.

[39] S. H. Dehshali, S. M. Hosseini, S. Z. Shabgahi, and B. Bahrak, "Throughput limitation of the off-chain payment networks," Cryptology ePrint Archive, Paper 2022/1614, 2022, https://eprint.iacr.org/2022/1614. [Online]. Available: https://eprint.iacr.org/2022/1614

[40] V. Buterin and V. Griffith, "Casper the friendly finality gadget," 2019.

[41] V. Buterin, D. Hernandez *et al.*, "Combining GHOST and Casper," *arXiv preprint arXiv:2003.03052*, 2020.

[42] Q. Wang, G. Yu, Y. Sai, C. Sun, L. D. Nguyen *et al.*, "An empirical study on snapshot DAOs," *arXiv preprint arXiv:2211.15993*, 2022.

[43] "How to stake in Ethereum 2.0," *https://ethereum.org/en/staking/*, 2023.

[44] R. Pass and E. Shi, "Rethinking large-scale consensus," in *IEEE Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 115–129.

[45] Z. Yuzhe *et al.*, "How to rationally select your delegatee in PoS," *arXiv preprint arXiv:2310.08895*, 2023.

# APPENDIX

## APPENDIX A. LIMITATION DISCUSSION

**Utility in "one-direction".** Our bridge is designed to facilitate unidirectional interactions, enabling users to invoke actions and trigger state transitions from the Bitcoin network to the Ethereum network. It is important to note that our solution does not support bi-directional functionality, meaning that users must initiate the workflow from the Bitcoin side and follow the established pathway to trigger events within the Ethereum smart contract. While the unidirectional nature of our bridge may impose certain constraints on its scope of application and usage, it is an inherent limitation resulting from the distinct data formats utilized by Bitcoin and Ethereum. The use of UTXOs in Bitcoin ensures a reliable transaction ordering mechanism but, by its nature, restricts the support for other features such as general state transitions in contracts. However, despite this limitation, we have successfully established a lightweight directional channel at a minimal cost, offering valuable assistance to Bitcoin users seeking to interact with the Ethereum network.

**Evaluation on "one-side".** Our evaluation primarily focuses on the smart contract functionality and performance within the Ethereum testnet. The limitation arises from our consideration of costs, particularly due to the unaffordability of conducting batch transactions on Bitcoin's network, which lacks a dedicated testnet. In this initial version, we have implemented all the functional events on both the Bitcoin and Ethereum sides,

enabling us to maximize our evaluation of the potential performance and associated costs. However, we acknowledge that there is ample room for further optimization. We encourage industry teams with an interest in this topic to invest more resources into conducting comprehensive evaluations.

**Extension, "NOT" comparison.** Even though we propose a middleware to bridge the Bitcoin network and Ethereum, our primary emphasis is not on cross-chain functionality, but rather on leveraging Ethereum as a tool to enhance BRC20. As a result, the protocol has been intentionally crafted to address the specific requirements of the BRC-20 scenario.

**Faithfulness of validators.** It is well recognized that even permissioned blockchain systems are not completely immune to the trustworthiness of validators, regardless of the committee size. Concerns may arise among regular users regarding the potential compromise of validators, which could pose a threat to the stability of the middleware. To mitigate such risks, we recommend that each middleware validator deposits a substantial amount of tokens (e.g., 32 ETH [43]) into the protocol. This ensures that validators have significant stakes in the network, reducing the likelihood of malicious behavior. This will provide users with a higher level of confidence when transferring larger amounts of tokens through the middleware. Additionally, increasing the committee size by enabling dynamic formation can significantly enhance the robustness and decentralization of the system, moving it closer to a *permissionless* model [44]. However, it's important to acknowledge that some degree of centralization might persist [45], but steps can be taken to mitigate this tendency.

## APPENDIX B. USE CASE

To illustrate the real-world operation of the MIDASTOUCH, we provide the following case. We have three participants named Alice, Bob, and Carol, all of whom are actively engaged in the network and aspiring to become validators for the system. To be eligible, they utilize the CommiteeRegistration function. Each participant provides their Bitcoin and Ethereum addresses and deposits a specified amount of ETH into a designated contract $c$. This transaction is recorded or inscribed onto the Bitcoin network, and subsequently, Alice, Bob, and Carol are added to the validator set $\mathbb{V}$.

Then, we introduce Dave, an end-user who intends to execute a transaction on the Bitcoin network. Dave creates an inscription in the transaction output, which is included when the new block is mined. At this point, the HandleInscription function comes into play. It scans each transaction output from the newly mined Bitcoin block, validates Dave's inscription, and appends it to the bundle $\mathbb{B}$. The corresponding contract address is also added to the contracts set $\mathbb{C}$.

On the Ethereum network, the UpdateEVMState function is triggered every $\varepsilon$ block. This function retrieves the latest state $s_i$, where $s_i \in \mathbb{S}$, for each contract $c$ within the set $\mathbb{C}$, including the contract to which Dave's inscription was added. In the case where the contract is associated with BRC-20 or similar token-managing functionalities, the function also fetches the

balance record $\Psi$ for each unique Bitcoin address linked to various tokens, including Dave's address.

During each $\varepsilon$ block, Alice, Bob, and Carol, as validators, process each inscription within the bundle $\mathbb{B}$. They distribute the gas fee $g$ among themselves based on their respective Bitcoin addresses. Through a collaborative multi-signature validation process, they update the state $s_i'$ ($s_i' \in \mathbb{S}$) and address balances $\Psi$. This completes the entire workflow of the proposed middleware protocol, ensuring a consistent state is maintained across the Bitcoin and Ethereum networks.

10