Enabling Semantic Blockchain Interoperability Across Heterogeneous Infrastructures

Carlos Amaro Rafael Belchior André Vasconcelos Miguel Correia
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
{carlosrscamaro, rafael.belchior, andre.vasconcelos, miguel.p.correia}@tecnico.ulisboa.pt

***Abstract*—With the advancement of blockchain technology, numerous systems and protocols are emerging, resulting in isolated environments that constrain the technology's full potential. Consequently, one of the key trends that has emerged in blockchain technology is interoperability. Our work proposes a gateway-based architecture that supports cross-chain transactions, i.e., transactions involving different blockchains. Each network is linked with gateways that are specialized nodes that are in charge of making transactions of assets using the Secure Asset Transfer Protocol (SATP). This paper proposes a specification and implementation for the SATP's Stage 0, where sessions between gateways are established and assets are prepared through wrapping on both source and destination networks, necessary conditions for a cross-network transfer. Its core is a wrapper contract that makes use of ontologies that formalize token behavior and interfaces, described in OWL; these ontologies are then used by the gateways to understand how to interact with the corresponding token's smart contracts. These ontologies improve the support for multiple forms of fungible tokens within the SATP. The evaluation of the adoption of a wrapper contract with ontology support indicates that the additional transaction latency is minimal, around 1%. Meanwhile, the implementation of Stage 0 of SATP increases the time required to complete transfers between Hyperledger Fabric and Besu networks by approximately 30%. Our work enables cross-chain asset transactions without the need for system and architectural modifications or modifications to the token's smart contracts, compared to existing solutions.**

***Index Terms*—Blockchain, Interoperability, Bridge, Ontology, SATP**

I. INTRODUCTION

A blockchain is a distributed, immutable ledger that underpins global value transfer and decentralized computing, with applications in sectors like DeFi, cybersecurity, and healthcare. The global blockchain market is projected to reach $1 trillion by 2032, growing at 85.7% annually [1]–[3].

Despite this progress, many blockchains still keep data and value isolated, limiting broader use [4]. This results in the creation of silos and fragmentation of these systems, eroding decentralization, making data and asset transfer more difficult, and locking users into a platform, in opposition to the core principles of Web3: decentralization, data ownership, and interoperability. Applications built on these fragmented networks often experience interoperability problems, creating barriers to migration and integration between systems. The challenge of connecting Distributed Ledger Technologies (DLTs) remains unresolved [4]. Cross-chain communication is the cornerstone of blockchain interoperability, as this allows transactions across different blockchains, allowing transactions to be initiated on the source blockchain and executed on the target blockchain, facilitating seamless interactions between networks [4].

There are many approaches to blockchain interoperability proposed, with only a few focusing on the semantic layer, where ontologies play a crucial role. This layer ensures that data is standardized, structured, and homogeneous across blockchains, which is crucial given the sensitive and critical nature of many blockchain records. Ontologies [5] allow data (e.g., assets, information) to be reusable and standardized. Although there is no universally accepted methodology for the development of ontologies [6], frameworks aim to establish semantic consistency between systems [7]. An ontology is a tool that is used to clarify and organize concepts within a specific area of interest. It helps reduce confusion and inconsistencies while identifying valuable opportunities in that domain [8]. An approach to addressing the semantic layer is through the ERC standards [9], which establish common interfaces, data structures, and event patterns. While these standards are excellent tools for deploying new contracts, they may not address the needs of existing non-standard contracts or those requiring alternative interfaces. Developing a unified semantic model or ontology remains challenging, as it requires balancing diverse frameworks, sometimes complementary, others conflicting. Ontologies improve the semantic layer by enabling extensible, reusable, and standardized data representation.

We propose an implementation for Stage 0 of the Secure Asset Transfer Protocol (SATP) [10]. Central to this approach is a wrapper contract that uses ontologies to provide improved support for various forms of fungible tokens within the SATP framework. Using an approach focused on the semantic layer, these ontologies facilitate direct mapping of token methods to the necessary SATP primitives, enabling the acceptance of a broader range of tokens, already or not deployed, with no modification to their implementation.

This document is organized as follows. Section II introduces the background knowledge on Blockchain Interoperability, Gateways and Bridges, and the Secure Asset Transfer Protocol (SATP). Then, Section III presents the design, architecture, and implementation details of our solution. In Section IV, we evaluate and demonstrate the performance of our solution. Section V gives a summary of existing solutions and a comparison with ours. In Section VI, we make our conclusion and some future work proposals.

II. Background

A. Blockchain Interoperability

In day-to-day use, interoperability refers to the characteristic of a product or system to work with other products or systems [11]. In the blockchain world, we can consider interoperability to be the ability of a source blockchain to change the state of a target blockchain (and vice versa), enabled by cross-chain or cross-blockchain transactions that span a composition of homogeneous and heterogeneous blockchain systems [4].

The primary goal of blockchain interoperability is to enable smooth connections between different blockchain networks, regardless of their underlying technical implementations. Blockchain interoperability can be understood through various layers, each addressing different concerns [12]. The technical layer encompasses protocols and standards to enable different blockchain networks to interact, focusing on protocol compatibility and data standardization. The semantic layer ensures that systems interpret data consistently, preserving context and meaning across platforms. Organizational interoperability deals with human and procedural aspects, facilitating collaboration between organizations with differing structures and processes within the distributed ledger technology ecosystem. Finally, governance and legal interoperability [4], which combines legal and governance aspects, addresses regulatory and compliance issues, ensuring that cross-chain interactions adhere to legal frameworks and requirements across jurisdictions.

B. Gateways & Bridges

Gateways serve as interfaces connecting blockchain networks to external systems (payment systems, fiat currencies). They enable asset deposits/withdrawals with blockchain representations and handle compliance functions, bridging the decentralized and centralized worlds [13]. Bridges function as protocols that enable cross-chain interoperability for transferring tokens, data, or assets between different blockchain networks. They overcome network isolation, typically by locking assets in the source chain while minting equivalents on the destination chain [4].

C. Secure Asset Transfer Protocol

The Secure Asset Transfer Protocol (SATP) is being developed by the Internet Engineering Task Force (IETF) [10]. SATP aims to standardize the secure and interoperable transfer of digital assets across various blockchain networks, regardless of their underlying technologies. By supporting atomic unidirectional asset transfers between gateways, it ensures transaction integrity and eliminates trust issues, paving the way for more complex N-to-N transactions.

SATP employs a gateway-based architecture to enable seamless asset transfers between different blockchains. SATP is still in development, but secure production-ready bridges remain a viable alternative solution for the time being. However, the main counterarguments are the lack of privacy in public bridges and the non-existent enterprise-grade support [14].

SATP transfer involves four key steps: *Lock*, sender's asset is secured on the origin ledger; *Mint*, a matching asset is created on the destination ledger; *Burn*, the original asset is destroyed in the source ledger; Assign, the minted asset is transferred to the recipient.

The SATP consists of four stages. In Stage 1, the two gateways authenticate each other, negotiate security parameters, exchange the asset information and operational parameters, and confirm agreement. During Stage 2, the first gateway locks the asset on its ledger, creates and signs a lock claim, which it sends to the second gateway for verification and logging, while both maintain detailed records of the process. In Stage 3, the first gateway signals the second to prepare for transfer, the second temporarily creates the equivalent asset, the first removes the original asset and notifies the second, the second then assigns the new asset to the recipient, and finally both gateways close their connection and clear any temporary information, completing the transfer.

Stage 0 involves pre-transfer verification and context establishment. This stage was considered beyond the scope of the current specifications. During Stage 0, applications create a unique transfer ID, gateways validate asset ownership and compatibility across networks, obtain necessary authorizations, exchange compliance information, authenticate parties, and agree on technical parameters for the transfer. Stage 0 is essential because it establishes trust, validates legal and technical prerequisites, secures required authorizations, and ensures seamless interoperability, thus mitigating potential disputes, operational errors, and security vulnerabilities before the beginning of the actual asset transfer process. And because of this, one of our contributions is to Stage 0.

III. SATP Gateway Architecture

The core problem we address is the challenge of performing atomic cross-chain transactions efficiently and directly between different DLTs. Current solutions often require significant modifications to existing systems and do not facilitate direct asset transfers between chains.

Figure 1 depicts the overall architecture of our solution, referred to as the Interoperability Mechanism (IM). The IM is fundamentally composed of five new main modules (Business Logic Orchestrator, Gateway Orchestration Layer, SATP Manager, Crash Recovery, and Bridge Modules) that are part of our contribution. These modules are explained in the following subsections.

Our approach takes advantage of Hyperledger Cacti[1]. We chose it because of its modular design properties, which make it a suitable method of interoperability in designing a

[1]https://github.com/hyperledger-cacti/cacti

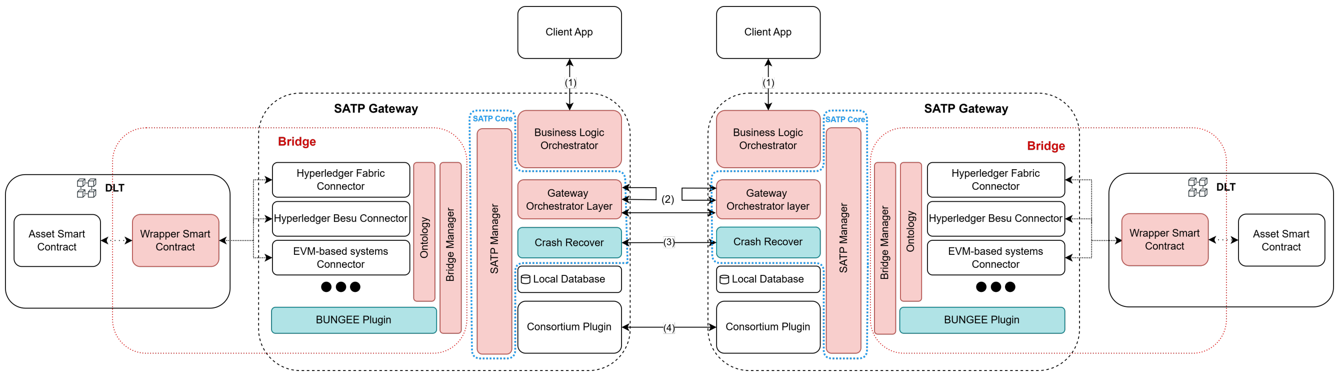


Fig. 1. Architecture of the Interoperability Mechanism, featuring a bridge module that is composed of ledger connectors (Hyperledger Fabric, Besu, and EVM-based systems), BUNGEE plugin, Ontology layer, Bridge Manager, Consortium plugin, crash recovery module, SATP manager module, gateway orchestrator module, and business logic orchestrator module. Modules colored in red are our contributions; modules colored in blue also had some contributions, but they already existed.

system that can be customized to support a wide range of distributed ledger technologies using a set of software plugins, which enable flexible customization and integration across different DLTs. This gateway is implemented using the new SATP protocol, ensuring compliance with the latest standards for secure asset transfer. This design allows stakeholders to continue using their preferred DLTs while cooperating and exchanging information with others, without the need to alter their underlying systems. The SATP Gateway[2] performs asset transfers between heterogeneous DLTs.

In the following subsections, we present a comprehensive description of the gateway, its functionalities, and its various layers. Our contributions include both the conception and implementation of key components such as Stage 0 and the ontology framework. In addition, we implemented other modules based on the existing architecture and specifications.

1) BLO Module: The Business Logic Orchestrator (BLO) acts as an intermediary between client applications and the SATP protocol. BLO translates various API requests, such as initiating transactions or retrieving session information, into SATP-compatible actions, ensuring business requirements are accurately mapped and then forwarding them for processing in the system. BLO, after receiving a transaction request sent by a client seen in (1) of Figure 1, processes the request, extracts relevant details, and then forwards the action to GOL to be processed.

2) GOL Module: The Gateway Orchestration Layer (GOL) is responsible for managing all gateway-to-gateway processes. It is in charge of gateway discovery, taking advantage of the Consortium Plugin, existing counterpart gateways, gateways' communication channels, core SATP service, and different stages. Importantly, while it provides the infrastructure to support SATP services, the actual execution of the SATP protocol and its stages is managed by the SATP Manager. A gateway can operate as both a client and a server, as seen in

[2]https://github.com/hyperledger-cacti/cacti/tree/satp-dev/packages/cactus-plugin-satp-hermes

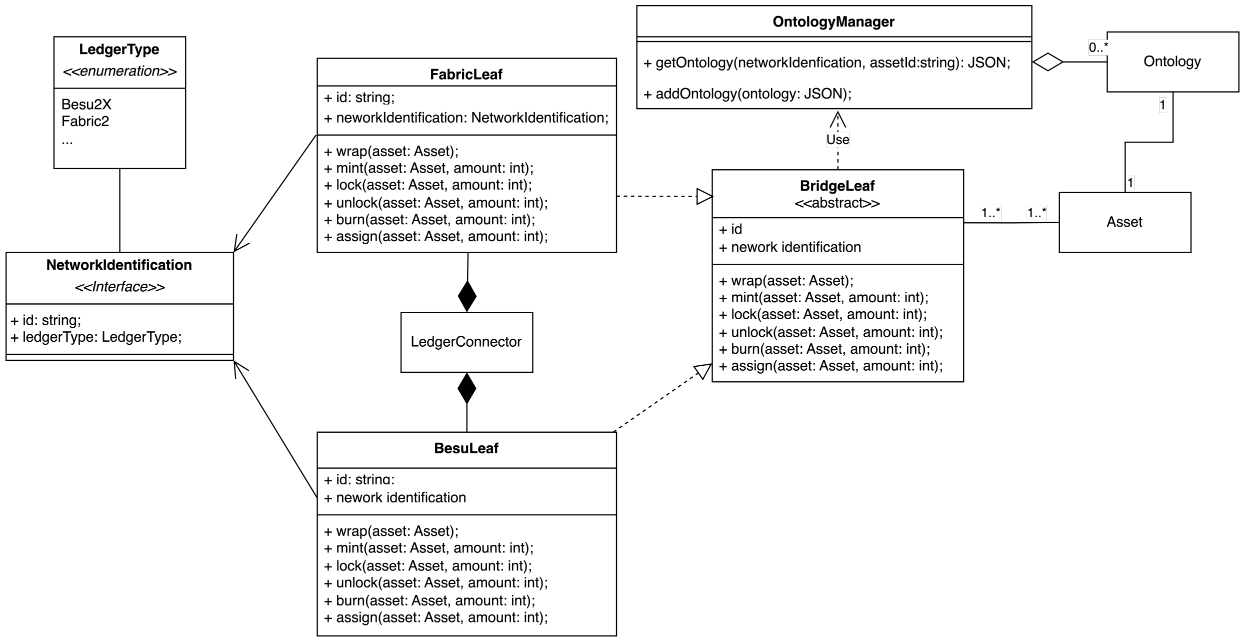
(2) in the figure 1, allowing independent asset transfers when connected to multiple ledgers.

3) SATP Manager: The SATP Manager is where the SATP protocol is implemented. For each transfer, it creates a separate session. The session is tasked with observing relevant data and operations, hence ensuring that transfers are secure and compliant with audit requirements. In addition, the module manages different phases of the protocol, such as validation, initiation of the transfer, and their completion afterward, in addition to protecting transaction states to allow rollback or recovery in cases of failure. It is integrated with other modules, such as the BLO, the Crash Recovery Module, and the Bridge Module.

4) Crash Recovery Module: The Crash Recovery module employs the SATP Gateway Crash Recovery Mechanism to safeguard the integrity of asset transfers during system failures. The module retrieves information from the local log database to restore any in-progress transfers. If recovery is not feasible, the module rolls back the transfer to its last state (returns the assets to their origin) [15]. This ensures that no assets are lost or incorrectly transferred.

5) Bridge Module: The bridge module is an interface between the gateway and the DLT. It comprises connectors, wrapper contracts, and a manager component. Connectors interact with specific DLT networks, handling their unique protocols and data structures. Wrapper contracts facilitate asset transfers and proof verification in source and target chains. The core of the bridge orchestrates the entire process, ensuring seamless communication and data consistency. The bridge acts as a wrapper around these connectors by abstracting the underlying DLT details.

The bridge interfaces with wrapper contracts and leverages ontology-specific methods to ensure seamless communication and data consistency across different DLTs. Using ontologies, the bridge can map data structures and concepts between different blockchains, facilitating accurate and efficient cross-chain interactions. Figure 2 presents the UML diagram that illustrates the relationship between the bridge module and the ontologies.



OntologyManager
+ getOntology(networkIdenfication, assetId:string): JSON;
+ addOntology(ontology: JSON);
0..*
Ontology
1
1
LedgerType
<<enumeration>>
Besu2X
Fabric2
...
FabricLeaf
+ id: string;
+ neworkIdentification: NetworkIdentification;
+ wrap(asset: Asset);
+ mint(asset: Asset, amount: int);
+ lock(asset: Asset, amount: int);
+ unlock(asset: Asset, amount: int);
+ burn(asset: Asset, amount: int);
+ assign(asset: Asset, amount: int);
Use
BridgeLeaf
<<abstract>>
+ id
+ nework identification
+ wrap(asset: Asset);
+ mint(asset: Asset, amount: int);
+ lock(asset: Asset, amount: int);
+ unlock(asset: Asset, amount: int);
+ burn(asset: Asset, amount: int);
+ assign(asset: Asset, amount: int);
1..*
1..*
Asset
NetworkIdentification
<<Interface>>
+ id: string;
+ ledgerType: LedgerType;
LedgerConnector
BesuLeaf
+ id: string;
+ nework identification
+ wrap(asset: Asset);
+ mint(asset: Asset, amount: int);
+ lock(asset: Asset, amount: int);
+ unlock(asset: Asset, amount: int);
+ burn(asset: Asset, amount: int);
+ assign(asset: Asset, amount: int);

Fig. 2. UML Diagram Illustrating the Relationship Between Bridges and Ontology

A bridge consists of two leaves (this design was inspired by bascule bridges[3]. In the blockchain context, a leaf refers to the component of a blockchain bridge that interacts with a specific DLT. At least two leaves are required to successfully perform a transfer between different DLTs. A leaf is composed of a connector and a wrapper smart contract, where each leaf is an implementation specifically designed for its corresponding DLT. Each leaf interacts exclusively with its wrapper smart contract to handle token operations, providing a standardized way to manage token transfers across different blockchain networks.

6) Wrapper Smart Contract: The Wrapper Contract simplifies cross-network interoperability by encapsulating assets within a unified framework. By defining a universal interface, both the source and destination networks can interpret and work with the asset in the same way. Operating a token through the Wrapper not only provides an abstraction from every token type (e.g., ERC-20, Fabric token, etc.) but also leverages the inherent logging features of DLTs.

Additionally, the Wrapper Contract receives an ontology each time the wrap method is called. This ontology describes how the wrapped token should be interpreted and interacted with on the underlying ledger.

Rather than interacting directly with the assets, the SATP protocol works with the Wrapper Contract, abstracting the complexities of different asset implementations. This intermediary layer serves two primary purposes: creating consistent bridging processes for interoperability regardless of asset implementation and managing structured metadata about locked assets on the source chain to ensure transparency and facilitate easier tracking of bridging transactions.

[3]https://www.collinsdictionary.com/dictionary/english/bascule-bridge

When the Wrapper Contract receives an asset from the bridge, it converts the incoming metadata and instructions into appropriate function calls for the corresponding fungible asset smart contracts or chain code, aided by the shared ontology.

Every Wrapped Asset has several key attributes. A unique ID, known as the Identifier, is used to initiate bridging operations on the escrowed asset. The Amount Approved represents the user's approved quantity for bridging, while the Amount Locked indicates the amount currently held in escrow during an ongoing bridging operation. The Owner is the entity that owns the Wrapped Asset and serves as both the sender and recipient in bridging. The Token Type refers to the specific token classification of a fungible token. Lastly, the Bridge ID is a unique identifier assigned to the bridge instance that manages the transaction, including the holding account of locked assets. Additional attributes can be added, depending on the DLTs utilized. For example, EVM-based networks might require a Contract Address, while Hyperledger Fabric [16] can have MSP ID, Channel Name, or Contract Name to act as organizational identification or describe chaincode specifications.

7) Other Plugins Used: During the development of the gateway, several pre-existing modules were utilized to enhance system integration and provide additional functionalities, although they fall outside the scope of this paper. The BUNGEE [17] plugin[4], a view generator, standardizes blockchain proofs of state across different ledger technologies by creating snapshots of states and transactions, ensuring integrity and confidentiality through Merkle tree roots and par-

[4]https://github.com/hyperledger-cacti/cacti/tree/main/packages/cactus-plugin-bungee-hermes

ticipant signatures. The Consortium Plugin[5] enables dynamic membership management in blockchain networks, allowing nodes to join by proving identity and awareness of shared policies, with a policy model for organizing rules. Hyperledger Cacti[6] Connectors, including Fabric, Besu, and Ethereum Connectors, provide seamless integration between different blockchain ecosystems and external applications, simplifying smart contract deployment and transaction management while ensuring security and scalability. The database is integrated with the gateway using *Knex*, which allows the usage of various SQL databases and distributed databases like *IPFS*.

A. SATP Implementation

The SATP implementation is designed around a series of stateless services that are logically separated into four stages of the SATP.

Each stage is then divided into two components – a client gateway side and a server gateway side – to maintain clear boundaries for responsibilities and operations, allowing a gateway to work as a client and server in the same transfer. The client gateway serves as the orchestrator of the SATP process.

Each request is made by the client gateway to the server gateway, and the request passes through assigned handlers on the server gateway that deploy each SATP service and pass them on via gRPC. This layered approach comprises separate stages, distinct client and server components, and clearly defined interfaces. It creates a unique session for handling the value and details of each transfer. Each session carries its own ID and contains two sets of data: one for the client side and one for the server side.

Each SATP message includes the following 14 common fields; however, some messages of the protocol may define additional fields. Because the protocol may evolve over time, each message specifies a *Version* to ensure that all participating parties run the same SATP specification. The *Message Type* indicates the function of the message, while the *Session ID* identifies the specific multi-party session to which the message applies. In sessions where multiple concurrent transfers occur, a *Transfer Context ID* pinpoints which transfer context a given message addresses.

To counter replay attacks and preserve message ordering, each message is tagged with a *Sequence Number* that increments with every communication. In addition, the *Resource URL* references the resource or asset involved in the transfer, and any outcome of an action invoked by the message is conveyed through the *Action Response* field. The *Credential Block* ensures that only authorized participants can operate or witness the transfer, embedding the necessary proofs or tokens.

For message content regarding assets or transaction details, the *Payload Profile* describes how to interpret the supplementary data in the Payload field. To verify conversation continuity, messages include the *Previous Message Hash*, a cryptographic link to the previous message in the chain. Any problems during processing are flagged by the *Error* and *Error Code* fields.

Finally, every SATP message is accompanied by a *Signature* to ensure authenticity and provide non-repudiation. This mechanism verifies that the content was not tampered with and that the message originated from the claimed sender.

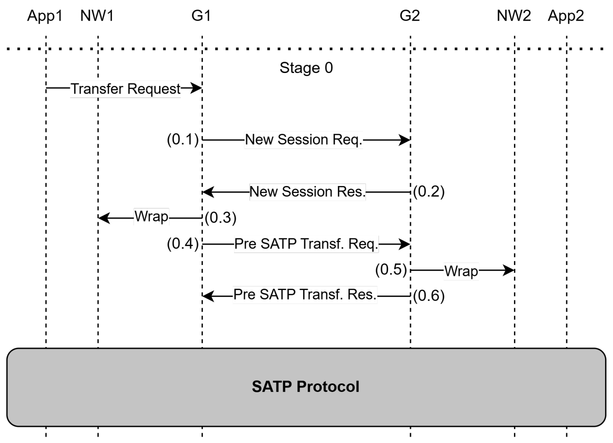


Fig. 3. Illustration of Stage 0 processes

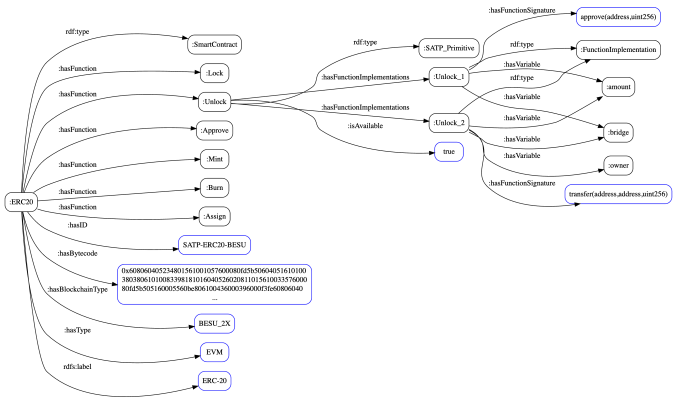


Fig. 4. Knowledge graph of ERC-20 EVM Ontology

B. Stage 0

The SATP white paper [18] outlines what should be done, but it does not yet define a protocol for Stage 0. Our implementation helps guide the specification for Stage 0. There are preliminary steps that must occur before the standard protocol stages can proceed. These steps include the creation of a session (session data is a record used by SATP to manage and track each transfer session) for each transfer between the involved Gateways and the wrapping of assets on both the source and destination networks. To address this, we have introduced Stage 0 to encompass these critical initial actions.

Stage 0 messages have the same format as the other SATP messages, ensuring consistency across protocol stages.

In Stage 0 (see Figure 3), G1 initiates a session by requesting it from G2, who acknowledges with acceptance or rejection (possibly suggesting an alternative session ID if

[5]https://github.com/hyperledger-cacti/cacti/tree/main/packages/cacti-plugin-consortium-static

[6]https://github.com/hyperledger-cacti/cacti

conflicts exist); once accepted, G1 wraps the asset within the source network (NW1) and sends proof to G2 as a pre-SATP transfer request containing wrap assertion claims and asset information for both networks, after which the asset is wrapped on the destination network (NW2) and G2 confirms this by sending proof back to G1.

Our stage 0 proposal for the SATP protocol addresses the gap in the original white paper [18] by defining the preliminary steps that are essential for seamless asset transfers. This stage ensures that a session is established between gateways and that assets are properly wrapped (mapped in the Wrapper Contract) on both the source and destination networks before proceeding with the standard SATP stages.

C. Token Ontology

This module is a critical component that improves interoperability among various blockchain networks by providing a bridge that contains a wide range of interfaces to connect to a certain smart contract. Every interaction is controlled by an ontology, defined in OWL, a known language to represent ontologies, which dictates the exact format, permitted actions, and expected behavior applicable to a specific token category. This 1:1 mapping between the ontology and the token type ensures that each operation is validated against a single authoritative source, maintaining both the token's functional consistency and overall integrity.

Beyond supporting a single token, this ontology-driven approach is designed to be flexible, accommodating multiple heterogeneous tokens across diverse platforms. Figure 4 displays a knowledge graph of the ontology of an ERC-20 token within an EVM environment, showing how basic actions such as unlocking, locking, minting, burning, assigning and approving are assigned to the corresponding names and parameters of the Solidity function. This mapping also specifies any availability constraints, reflecting whether these actions are permitted in a given deployment. In addition, the ontology includes several key attributes that define the structure and behavior of a token. A unique identifier distinguishes the token definition; each ontology should have a unique ID. The contract specification indicates the smart contract language used for implementation, confirming its execution environment. The compiled bytecode provides the low-level machine-readable representation of the contract, enabling it to be deployed on the blockchain. A cryptographic signature ensures the authenticity of the contract's deployment or interactions. In addition, a cryptographic hash serves as a unique fingerprint of the contract.

This structured approach ensures consistency in how tokens are handled across different environments while allowing for extensibility to new token standards and blockchain ecosystems. In addition to listing these fundamental operations, the ontology includes important metadata such as the contract's bytecode, digital signature, and cryptographic hash. These components are important in maintaining integrity by allowing unauthorized changes to be prevented. By retrieving the ontology from a controlled local repository, administrators can tightly govern revisions, preventing unauthorized or accidental changes that might compromise security or functionality.

IV. Evaluation

In this section, we evaluate the IM, the SATP Gateway with the Stage 0 Implementation, and the implications of the ontology definition implementation. We go through the evaluation process to assess the performance of the SATP Gateway under different conditions and assumptions.

A. Goals

Before proceeding with the evaluation of the proposed solutions, it is essential to outline the goals and objectives of this evaluation:

1) What is the performance, in terms of latency, of making one cross-chain transfer between two gateways?
2) What is the added latency of the Stage 0 step?
3) What is the performance, in terms of latency, of making one cross-chain transfer between one gateway using the SATP gateway implementation?
4) What is the impact, in terms of latency, of using a Wrapper Smart Contract, which utilizes ontologies to facilitate semantic interoperability, compared to directly calling the Asset contract?

B. Experimental Environment

To evaluate the solutions, we set up a testing environment. We set up an Ubuntu 24.04 LTS virtual machine with 12GB RAM, 6-core CPU, and 128GB SSD in a server running a Hypervisor OS, Proxmox 8.2.2. The server was running an 8th Gen Intel(R) Core(TM) i7-8700 @ 3.20GHz processor with 64GB 2666mhz DDR4 RAM. Our test environment incorporated two distinct blockchain networks using preconfigured test ledgers from the Cacti project repository: Hyperledger Fabric (Fabric Samples[7] config) and Besu (we used a custom genesis configuration, the genesis file specifies a lightweight Ethash consensus mechanism with a fixed mining difficulty, enables the London hard fork from block zero, and sets a very high gas limit to allow unrestricted smart contract execution)[8]. The testing was carried out in a dedicated Cacti branch[9] by running the various tests present in the *satp-hermes* package. Every test was executed 50 times.

1) Wrapper Smart Contracts With Ontology Implementation: We begin by evaluating the performance implications of calling methods through the Wrapper Contract. Table I shows the time necessary to execute the SATP methods called directly and through the Wrapper Contract in a Besu and Hyperledger Fabric environment. In Besu, using the Wrapper Contract increased the execution time for the *Assign* method by 3.07% and the *Burn* method by 0.05%, while reducing the execution time for the *Lock* and *Mint* methods by 0.88% and 7.19%, respectively. The largest improvement was observed in

[7]https://github.com/hyperledger/fabric-samples
[8]https://github.com/hyperledger-cacti/cacti/blob/main/tools/docker/besu-all-in-one/genesis.json
[9]https://github.com/LordKubaya/cacti_fork/tree/thesis-test

TABLE I
EXECUTION TIME FOR SATP METHODS IN BESU AND HYPERLEDGER FABRIC: DIRECT CALL VS. WRAPPER

Method	**Besu**					**Fabric**				
	Direct(ms)	σ(ms)	**Wrapper**(ms)	σ(ms)	**Diff**(%)	**Direct**(ms)	σ(ms)	**Wrapper**(ms)	σ(ms)	**Diff**(%)
Assign	1022.12	296	1053.46	420	3.07	4176.58	139	4179.22	165	0.63
Burn	1019.44	323	1020.00	308	0.05	4192.88	180	4244.40	213	1.23
Lock	1019.76	328	1010.74	321	−0.88	4190.78	138	4226.08	253	0.84
Mint	1076.84	439	999.40	298	−7.19	4216.14	189	4246.40	195	0.72
Unlock	2061.00	473	1081.92	358	−47.50	4210.18	229	4192.54	194	−0.42

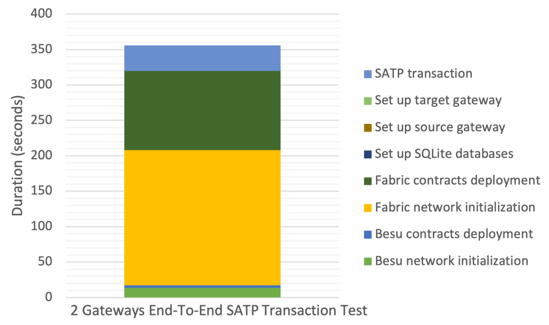


Fig. 5. SATP e2e test including ledger configuration, contract and SQLite deployment, and two gateways setup

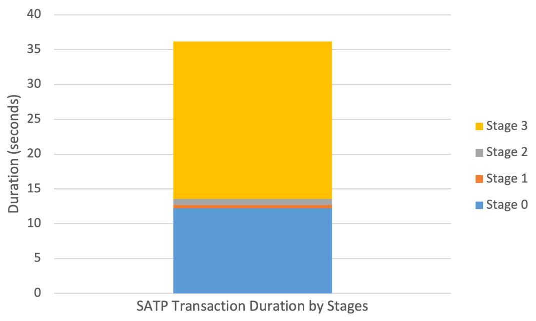


Fig. 6. SATP execution between two Gateways divided into the different stages

the *Unlock* method, with a 47.50% reduction in execution time. The *Wrap* and *Unwrap* operations averaged 1030.29 ms ($\sigma =$ 395.94 ms) and 1086.20 ms ($\sigma = 362.04$ ms), respectively, in the Besu environment.

Based on the previous analysis, methods that require only a single contract call show an average time difference of approximately 1.24%, which is largely negligible. However, when a SATP method involves multiple calls to the contract, there are significant performance improvements. This outcome can be attributed to the fact that the connector call accounts for most of the overall execution time.

In Hyperledger Fabric, the Wrapper Contract led to execution time increases of 0.63% for the *Assign* method, 1.23% for *Burn*, 0.84% for *Lock*, and 0.72% for *Mint*, while the *Unlock* method saw a 0.42% decrease. The average execution times for *Wrap* and *Unwrap* were 4220.82 ms ($\sigma = 166.58$ ms) and 4206.02 ms ($\sigma = 180.09$ ms), respectively, in the Hyperledger Fabric environment.

According to the Fabric analysis, the average difference across all methods is under one percent, indicating that the additional overhead introduced by the Wrapper Contract is effectively negligible.

2) SATP Gateway: Figure 5 shows the execution of one test that performs a SATP transfer between two gateways. The time necessary to setup the local and remote SQLite database, the Besu network and respective contracts, the Hyperledger Fabric network and respective contracts, two gateways, and running a SATP transfer between both gateways were 0.15, 13.64, and 13.64, 190.65 and 111.87, 0.03, and 36.15 seconds, respectively. The duration of setting up the Hyperledger Fabric network is what consumes the majority of the time, followed by the deployment of the Fabric contracts, taking 53.54% and 31.41% of the total time, respectively. The SATP transfer execution only takes 36.15% of the total time. In both the Hyperledger Fabric and Besu network setups, we deploy a pair of smart contracts, the asset and wrapper contracts. Naturally, the deployment and execution of each can vary greatly in duration.

Our next goal is to examine how latency is distributed among the different stages of a single SATP execution. Figure 6 depicts both the overall runtime of SATP and the time spent in Stages 0, 1, 2, and 3. In Stage 0, the gateways receive the ontology for each ledger of the asset being transferred. After the ontology is loaded into the Wrapper Contract, and at this point, a session is also created. In Stage 1, only a few messages are exchanged, and both gateways perform various validations. In stage 2, the gateways perform the locking of the asset on the source blockchain, an action that requires an on-chain transaction. Finally, in Stage 3, the gateways fulfill their commitments from the earlier phases: the locked asset on the source blockchain is burned, and is minted on the destination blockchain. This final step involves issuing two new transactions.

The latency of the SATP stage is proportional to the complexity of the transaction and the frequency of the interaction with the ledgers. Stage 3 has the highest impact (62.61%, 22.63s), involving multiple complex transactions (burn in Besu; mint and transfer in Hyperledger Fabric) requiring cross-ledger coordination. Stage 0 comes next (32.71%, 11.82s) with wrap operations in both the Besu and Fabric ledgers. Stage 2 has minimal impact (3.27%, 1.18s), performing only a single lock operation in Besu. Stage 1 contributes the lowest impact

(1. 41%, 0.51 s) as it does not require ledger transactions. The total SATP process takes 36.14 seconds.

C. *Qualitative Assessment*

Belchior et al. [19] propose a comprehensive evaluation framework to help stakeholders choose the right solution for blockchain interoperability. This framework is particularly useful, as it takes into account a wide range of qualitative factors, allowing for a thorough assessment. It evaluates each solution across three dimensions: Potentiality, Compatibility, and Performance, picturing both the strengths and limitations of a given approach.

Our solution has a Potentiality Assessment of P4 (P1-P4). This indicates that our solution is capable of connecting with different blockchain networks, both those using the same protocol and those built on different architectures. Through testing, we perform asset transfers between Hyperledger Fabric and Besu networks, both with distinct architectures.

In terms of Compatibility Assessment, our solution is at the C2 level (C1-C3). It provides a basic level of semantic and organizational interoperability. This means that systems using our solution can understand and process data from each other using shared formats (these shared formats being the ontology framework developed) and basic agreements. However, other integrations, such as governance models or legal frameworks, are still a work in progress. This is largely due to the early stage of development of the consortium plugin, which currently allows some policy sharing across networks.

For Performance Assessment, our solution is at the PE1 level (PE1-PE3), which reflects acceptable throughput and latency. As discussed in Section IV-B1, the speed of cross-chain transactions is mainly determined by the underlying consensus protocols of the connected blockchains. We did not assess the cost of operation in detail, since we focus on private networks, where cost is not as important as it is for public networks. However, running the system on public blockchains such as Ethereum Mainnet could result in higher fees.

V. Related Work

Current blockchain interoperability solutions vary from requiring intermediary entities to facilitate exchanges or transfers. A comprehensive survey on blockchain interoperability [4] categorizes these solutions into three types: Public Connectors, Blockchain of Blockchains (BoB), and Hybrid Connectors. Public Connectors enable interoperability between public blockchains through mechanisms such as sidechains, notary schemes [4], or Hash Time Lock Contracts (HTLC) [20]. BoB solutions support the development of application-specific blockchains that can interoperate with each other [4]; the best known examples include Polkadot [21] and Cosmos [22]. Solutions that do not fit into the first two categories are classified as Hybrid Connectors, which are designed for both public and private environments, such as those providing an abstraction layer for interacting with underlying ledgers, SATP according to [4] is in the Hybrid Connectors category [23]. While Polkadot enables communication between its parachains through a relay chain architecture, and Cosmos connects independent blockchains via its Inter-Blockchain Communication protocol, both require chains to conform to their respective frameworks.

Our solution focuses on semantic interoperability through ontologies that formalize the understanding of asset contracts across blockchains. Unlike the solutions mentioned above, which require substantial architectural modifications, our method minimizes implementation changes while enabling direct asset transfers between heterogeneous blockchains.

VI. Conclusion

We propose an innovative approach to blockchain interoperability, utilizing ontologies that describe the methods of asset contracts necessary for the SATP protocol. This paper contributes to SATP gateways that have the ability to transact different fungible assets between heterogeneous blockchains using the SATP protocol. We propose an ontology model that enables smart contracts to interoperate without additional code changes, offering a unified asset definition across multiple DLTs.

We evaluated our implementation of this architecture using the SATP protocol within Hyperledger Cacti. Our assessment focused on the performance of the SATP Gateway and the impact of incorporating Stage 0 and Wrapper Contracts that support ontologies. We found that the additional steps in Stage 0 represented 32.71% of the overall SATP process time. However, the use of the Wrapper Contract had a negligible effect on latency.

We are currently extending our solution to support non-fungible tokens, including the development of an ontology system designed to accommodate these unique digital assets. In addition, the ontologies used in our solution are not distributed or shareable. Future research could explore the use of DLT for ontology storage and management via smart contracts, allowing easier sharing and interoperability between bridges. Smart contracts could support version control and secure data sharing, allowing the ontology to evolve with system requirements while maintaining integrity through consensus-based updates. This would decentralize control, reduce manual intervention, and ensure compatibility across the network.

Acknowledgments

This work was financially supported by Project Blockchain.PT – Decentralize Portugal with Blockchain Agenda, (Project no 51), WP 7: Interoperability, Call no 02/C05-i01.01/2022, funded by the Portuguese Recovery and Resilience Program (PRR), The Portuguese Republic and The European Union (EU) under the framework of Next Generation EU Program. This work was also supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 and UID/PRR/50021/2025 We thank the anonymous reviewers for their suggestions, which helped us improve this paper.

REFERENCES

[1] J. Kolb, M. AbdelBaky, R. H. Katz, and D. E. Culler, “Core concepts, challenges, and future directions in blockchain: A centralized tutorial,” *ACM Comput. Surv.*, vol. 53, no. 1, feb 2020.

[2] F. A. Sunny, P. Hajek, M. Munk, M. Z. Abedin, M. S. Satu, M. I. A. Efat, and M. J. Islam, “A systematic review of blockchain applications,” *IEEE Access*, vol. 10, pp. 59 155–59 177, 2022.

[3] Polaris Market Research, “Blockchain technology market size, share, analysis report, 2023-2032,” 2023. [Online]. Available: https://www.polarismarketresearch.com/industry-analysis/blockchain-technology-market

[4] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, “A survey on blockchain interoperability: Past, present, and future trends,” *ACM Comput. Surv.*, vol. 54, no. 8, Oct. 2021.

[5] N. Guarino, D. Oberle, and S. Staab, “What is an ontology?” *Handbook on ontologies*, pp. 1–17, 2009.

[6] N. F. Noy, D. L. McGuinness *et al.*, “Ontology development 101: A guide to creating your first ontology,” 2001.

[7] U. Hector and C. Boris, “BLONDiE: Blockchain ontology with dynamic extensibility,” *CoRR*, vol. abs/2008.09518, 2020. [Online]. Available: https://arxiv.org/abs/2008.09518

[8] J. de Kruijff and H. Weigand, “Understanding the blockchain using enterprise ontology,” in *Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Springer, 2017, pp. 29–43.

[9] Ethereum Foundation, “Ethereum Improvement Proposals,” accessed: January 29, 2025. [Online]. Available: https://eips.ethereum.org/

[10] T. Hardjono, M. Hargreaves, N. Smith, and V. Ramakrishna, “Secure Asset Transfer (SAT) Interoperability Architecture,” Internet Engineering Task Force, Internet-Draft draft-ietf-satp-architecture-06, Dec. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-satp-architecture/06/

[11] Merriam-Webster Inc., “Merriam-Webster Dictionary,” Apr. 2008.

[12] E. Abebe, D. Behl, C. Govindarajan, Y. Hu, D. Karunamoorthy, P. Novotny, V. Pandit, V. Ramakrishna, and C. Vecchiola, “Enabling enterprise blockchain interoperability with trusted data transfer,” in *Proceedings of the 20th International Middleware Conference*, 2019, pp. 29–35.

[13] T. Hardjono, “Blockchain gateways, bridges and delegated hash-locks,” 2021. [Online]. Available: https://arxiv.org/abs/2102.03933

[14] K.-E. Marstein, A. Chiriac, L. Riley, T. Hardjono, and G. Verdian, “Implementing secure bridges: Learnings from the secure asset transfer protocol,” in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2023, pp. 1–9.

[15] R. Belchior, M. Correia, A. Augusto, and T. Hardjono, “Secure Asset Transfer Protocol (SATP) Gateway Crash Recovery Mechanism,” Internet Engineering Task Force, Internet-Draft draft-belchior-satp-gateway-recovery-03, Jan. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-belchior-satp-gateway-recovery/03/

[16] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proceedings of the 13th EuroSys Conference*, 2018, pp. 1–15.

[17] R. Belchior, L. Torres, J. Pfannschmidt, A. Vasconcelos, and M. Correia, “BUNGEE: Dependable blockchain views for interoperability,” *Distrib. Ledger Technol.*, vol. 3, no. 1, Mar. 2024.

[18] M. Hargreaves, T. Hardjono, R. Belchior, and V. Ramakrishna, “Secure Asset Transfer Protocol (SATP) Core,” Internet Engineering Task Force, Internet-Draft draft-ietf-satp-core-11, Jan. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-satp-core/11/

[19] R. Belchior, L. Riley, T. Hardjono, A. Vasconcelos, and M. Correia, “Do you need a distributed ledger technology interoperability solution?” *Distributed Ledger Technologies: Research and Practice*, vol. 2, no. 1, pp. 1–37, 2023.

[20] Bitcoin Wiki, “Hash time locked contracts,” 2021. [Online]. Available: https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts

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

[22] J. Kwon and E. Buchman, “Cosmos whitepaper: A network of distributed ledgers,” 2019.

[23] R. Belchior. The state of blockchain interoperability in 2021. [Online]. Available: https://rafaelbelchior.medium.com/the-state-of-blockchain-interoperability-in-2021-2780aa28957f