Year: 2019

Bifröst: a Modular Blockchain Interoperability API

Scheid, Eder John ; Hegnauer, Timo ; Rodrigues, Bruno ; Stiller, Burkhard

DOI: https://doi.org/10.1109/LCN44214.2019.8990860

Originally published at:
Scheid, Eder John; Hegnauer, Timo; Rodrigues, Bruno; Stiller, Burkhard (2019). Bifröst: a Modular Blockchain Interoperability API. In: 44th Conference on Local Computer Networks (LCN 2019), Osnabrück, 1 October 2019, IEEE.
DOI: https://doi.org/10.1109/LCN44214.2019.8990860

Bifröst: a Modular Blockchain Interoperability API

Eder J. Scheid, Timo Hegnauer, Bruno Rodrigues, Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
[scheid,rodrigues,stiller]@ifi.uzh.ch, t.hegnauer@gmail.com

***Abstract*—The blockchain (BC) world is rapidly becoming a universe of several ledgers designed for a specific purpose, holding data previously stored (*i.e.,* siloed) in centralized databases. The use of different BCs for the same purpose could hamper the frictionless exchange of data or value. On one hand, it is natural that there are competing implementations exploring the benefits of BC. On the other hand, the problem of siloed data re-emerges, with respect to isolated chains. In this regard, BC interoperability is necessary to connect different BCs, exchanging information and assets. Moreover, to foster BC employment, developers must be able to interact with such different BCs without knowing the details of each implementation. This paper presents a novel solution, called `Bifröst`, to store and retrieve data on different BCs. `Bifröst` employs a notary scheme, which allows for connectivity to different BCs. The presented prototype is highly modular and currently implements seven adapters to popular BC implementations, including Bitcoin, Ethereum, and Stellar. The developed prototype was evaluated concerning performance, security, and data size to verify the feasibility of such an implementation and assess design decisions taken during its development.**

***Index Terms*—Interoperability; Transparency; Blockchain.**

I. INTRODUCTION

The blockchain (BC) concept was introduced in 2009 with the release of the Bitcoin white paper [26]. Bitcoin was the first cryptocurrency achieving a market capitalization of more than 69 billion US dollars [8] in 2019. After its release, several cryptocurrencies were created taking advantage of the BC technology and the success of Bitcoin. As of 2019, there exist more than 2000 different cryptocurrencies [9]. These cryptocurrencies either follow the Bitcoin BC protocol, or design their own BC protocol and implementation focusing on special features, such as smart contract support [6], privacy [23], digital identities [28], or tools to create private BCs for enterprises [13].

Due to their different protocols and technologies, native cryptocurrencies cannot be exchanged between two BCs. Furthermore, it is not possible to exchange information about their state or events directly. Thus, the research on BC interoperability is a promising field to address such a problem and is also referred to as the "holy grail" of BCs [12]. In a BC context, interoperability means connecting multiple BCs to access information and act on it by changing its state or the state of another BC. Optimally, this would be achieved without compromising the premise of trustlessness. In this sense, the main goal of BC interoperability is to link BCs, which are specialized for a specific use case. For example, when a car is bought using a specialized payment chain (*e.g., PaymentChain*) and should be automatically registered in a private chain issued by an insurance (*e.g., InsuranceChain*). Additionally, BC interoperability can be used to increase the performance of BCs by linking multiple smaller instances or decrease the risk of unavailability or attacks by distributing and duplicating data on multiple BCs [22].

From a developer-focused standpoint, interoperability also implies that a BC application is able to interact with multiple BCs without restrictions. However, as each new BC presents different library implementations and Application Programming Interfaces (API), developers currently need to acquire detailed knowledge about the technical implementation of each BC to be able to interact with it programmatically. This is a significant restriction for interoperability with BCs.Thus, simpler and intuitive interfaces need to be provided to users in order to enhance interoperability not only with other BCs but with legacy applications, as proposed earlier in the overlay networks context [21]. Ultimately, fostering the creation of innovative BC-based applications.

This paper introduces an interoperability API implementation, called `Bifröst`, that abstracts development complexities of each BC, allowing users to easily interact with multiple BCs without requiring specific knowledge about their implementations and languages. The design of `Bifröst` focused on providing a *(i)* flexible, *(ii)* modular, and *(iii)* easy to use API. Moreover, the prototype implementation of `Bifröst` is presented, including a modular interface to seven different BCs in the form of a Python API, allowing users to store and retrieve arbitrary data on available BCs. In summary, the contributions of this work are:

- The design of a BC Interoperability API;
- Removal of technical BC knowledge from applications;
- Automated BC transaction creation; and
- Transparent BC interaction.

The remainder of this paper is organized as follows. Section II classifies interoperability techniques and related work. Section III, presents the architecture design and the implementation of `Bifröst`. Section IV evaluates and discusses the prototype. Section V concludes the paper and present an outlook on future work.

II. BACKGROUND AND RELATED WORK

This section classifies existing interoperability techniques presenting an overview of the state-of-the-art. Finally, a discussion summarizes the current state and main challenges of BC interoperability.

A. Interoperability Techniques Classification

According to [7], there are currently three techniques to achieve interoperability between BCs: **Notary schemes**, **Sidechains**, and **Hash-Locking**.

Notary schemes use a trusted entity as an intermediary between two BCs. The role of the notary is to verify that an event took place in one BC and feed this information to another BC. The main advantage of the notary scheme is its simplicity, as no change is required in the underlying implementation of BCs. The downside is that it is necessary to trust in the notary [7].

Sidechains provide the ability to validate and process information about the state of other BCs. Technically, this is achieved by using Simplified Payment Verification (SPV). SPV uses block headers and Merkle trees to verify if a transaction occurred on another BC without having to download the whole ledger. Although the data needs to be externally fed from one BC to the other, this process does not require trust. Due to the cryptographic properties of BCs it is simple to prove if the data has been tampered with [26], [27]. Furthermore, exchanges between two BCs can be enabled by so-called "Pegged sidechains". This scheme generates a proof that the assets are locked in one BC, so that a transaction of the same amount can be made on a second BC. However, smart contract capabilities are required to create a sidechain. Furthermore, to achieve full interoperability, every BC would need a sidechain, which in turn needs to support every other BC. The maintenance of this growing system becomes a major challenge.

Hash-locking is a technique that allows trades between two or more parties without an intermediary. Hash-locking triggers an action on two BC simultaneously in an atomic manner, *i.e.,* with both actions or none of them occurring. For example, two parties A and B want to make a trade but have their assets on different BCs. In order to perform the trade, they can use the following scheme:

1) A generates a secret s and computes the hash $h(s)$, which is sent to B.
2) Both A and B lock their assets into a smart contract, which can verify if an inputted s belongs to $h(s)$.
3) A has to provide s to the contract holding B's funds within $2{*}X$ seconds to trigger a transfer to A; otherwise, the asset goes back to B.
4) B has to provide s to the contract holding A's funds within X seconds to trigger a transfer to B; otherwise, the asset goes back to A.
5) A reveals the secret within X seconds. Thus B learns the secret and can claim the asset from A.

As hash-locking schemes can be chained after each other, it is possible to enable trades even if there is no direct connection between the trading parties. This technique is used by most decentralized exchanges today. This scheme works and is atomic as long as both parties act according to their financial interest. One drawback of such a scheme is that the BC needs to support this particular type of smart contract, called Hash-TimeLock Contract (HTLC). Also, the waiting period X could be exploited by speculating on falling or increasing prices between the trades.

B. Notary Scheme Projects

Herdius [16] is a decentralized exchange platform which focuses on the common linking point among all BCs, the private keys. Thus, Herdius enables exchanges between different BCs through sharing them. To decentralize this process, Herdius encrypts the users' private keys (*e.g.,* Ethereum private key) with a private key generated within Herdius. This key is then sliced and distributed, using a threshold multisignature mechanism, to notaries called "assembler nodes". Multiple assembler nodes are able to sign a transaction on behalf of the user by combining their parts of the private key. To include another layer of security, no assembler node can completely decrypt the native private key. Instead, the transaction is signed by making use of homomorphic cryptography computations. By using this structure, Herdius is able to decentralize the notary scheme [10].

C. Sidechain Projects

BTC-Relay [5] enables one-way interoperability between Bitcoin and Ethereum. It is a sidechain for Bitcoin in Ethereum, which uses SPV to verify and process transactions from Bitcoin. This work relies on so-called "Relayers" that submit Bitcoin headers to an Ethereum smart contract to earn a micro-fee every time an application processes a Bitcoin payment in the BTC-Relay smart contract.

Aion [25] tackles the problem of the Sidechain scheme that every BC would need a sidechain for each other BC by creating a central hub. Every BC would be able to route transactions through the Aion hub to every other BC. This hub itself is a BC and connects to the different BCs using their protocol powered by sidechains. Thus, every BC which supports this protocol can become a part of the Aion network, but has to fulfill the following requirements:

- Be decentralized in some fashion and support procedures commonly found in BCs such as atomic broadcast and transactions.
- Be able to recognize transactions between BCs as distinct from regular transactions.
- Be aware of the consensus protocol used by the bridge and store a transaction deemed valid.
- Implement lock-time or a similar feature that allows tokens to be held by the network for a period of time.

Although these are standard requirements for the sidechain approach, most of today's major BCs do not natively fulfil them. To still be able to integrate them into Aion's system, they are working on "token bridges", which are tailored solutions for individual BCs. One of Aion's concrete accomplishments is a pegged sidechain in Ethereum [4].

Cosmos [24] provides a hub for interoperability and their Proof-of-Stake (PoS) consensus mechanism called "Tendermint", which is built with interoperability as main principle. Hence, apart from connecting today's BCs to the hub using sidechains, Cosmos is providing tools for users to create their BC with *Tendermint* as the consensus mechanism. Interoperability between the *Terndermint*-based BCs is enabled more straightforwardly and directly than SPV.

Polkadot [36] presents a similar approach to Cosmos and uses a slightly adapted version of their consensus mechanism "Tendermint". The main difference between Polkadot and Cosmos lies in their governance approach. Cosmos allows any BC to join its hub as long as they support their protocol. In contrast, to attach a BC to the Polkadot network, a large number of their cryptocurrency ("Dots") must be staked. Therefore, Polkadot takes more control over their partner chains by being able to punish malicious behaviour.

Ark [32] describes itself as an open-source framework to create BCs with one click. The ARK Platform provides a so-called "Smart Bridge". The Smart Bridge is a sidechain in the Ark BC. Instead of supporting different BCs by implementing sidechains in other BCs, Ark provides a generic protocol; thus, leaving it up to the BC to support it. However, this means that either a BC needs to be created by using the tools from Ark or it has to change its implementation to be able to send data to the Ark BC. A second interoperability feature is their planned integration of cryptocurrency exchanges (*e.g.,* Shapeshift) to exchange between different cryptocurrencies automatically. Thus, additionally, implementing a notary scheme.

MIT's Tradecoin interoperability model [15] proposes a design to achieve BC interoperability following the principles of the Internet architecture, with BCs being Autonomous Systems (AS) and a set of gateways interconnecting them. In the design, the gateways act as inter-domain (*i.e.,* inter-BCs) entities and are responsible for implementations of Tradecoin, validating transactions in supported BCs, and mediating transactions between private BCs.

D. Hash Locking Projects

Wanchain [34] has developed a hash lock-based solution to enable token transfers between their BC and Ethereum. Furthermore, they are working on supporting other BCs. Instead of the more common Hashed Time Lock contract, they rely on so-called "Locked Account" scheme. This scheme is used to create an account which will be locked while the two transfers are occurring. It manages that by splitting up the private key of the account and distributing it to multiple nodes in the Wanchain. The nodes can only jointly perform the transfer from the locked accounts and will make sure that the transactions are atomic. Another interoperability feature is the ability to run Ethereums Solidity contracts on the Wanchain.

Interledger claims to be "the protocol for connecting ledgers", and their InterLedger Protocol (ILP) is a reference to their inspiration, which is the IP protocol [18]. Different from the projects above, Interledger is not a BC, token, or central service. Instead, Interledger is a network of connectors, connecting hashed-timelock contracts in different BCs to enable decentralized exchanges [19].

E. Discussion

Although differing in the architectures and schemes, most of the projects presented follow the sidechain approach. This approach is the most promising one regarding the spectrum of use cases it can cover. However, it requires changes in the implementation of target BCs, as most of them do not support locking or destroying tokens. To overcome this restriction, some projects combine different techniques. For example, Interledger combines sidechains with notary scheme [33] to achieve interoperability. Moreover, most of the projects focus on the interoperability by connecting BCs via a hub in a separated ecosystem instead of enabling interoperability between existing BCs. Thus, leading to spread islands of interoperability projects.

Moreover, the projects described in this section do not provide the same characteristics as `Bifröst`. For example, [16] requires multiple assembler nodes to sign a transaction, which hinders *easy of use*, as these nodes must be configured and always available. Sidechains projects, such as [5], [15], [24], [25], [32], [36] are not *flexible*, because they require changes in the underlying implementation of the supported BCs. Even though [33] provides a *modular* design, it depends on the BC support of hashed-timelock contracts. Further, [34] enables the exchange of its currency to other BCs. However, it maintains another BC and the underlying BC must support hash locking mechanisms. Thus, both projects require mechanisms that hamper the system's *flexibility*. In this sense, `Bifröst` differs from these interoperability projects as it aims to provide all these characteristics simultaneously.

III. MODULAR BLOCKCHAIN INTEROPERABILITY API

This section describes `Bifröst`, being organized into three subsections. Firstly, it is defined the API goal and the requirements taken into consideration during the development of the API. Secondly, it is described the details of the components and workflow of the API. Lastly, it is presented the implementation of the API, and listed technical details of libraries and technologies. It should be mentioned that `Bifröst` is part of the *Policy-based Blockchain Agnostic Framework* [29], and is called OpenAPI in such a framework.

A. Objective and Requirements

The main objective of `Bifröst` is to provide a simple interface to interact with different BCs, *i.e.,* allows users to store, retrieve, and migrate data from BCs. In this paper, a "user" is defined as a developer of a BC application. Thus, this interface allows the developer to create programs that support a variety of BCs without knowing the underlying BC and library implementation. Additionally, this allows further abstractions by potentially letting algorithms decide on which BC to use depending on certain parameters, *e.g.,* transaction costs or BC performance. Figure 1 depicts the interaction with a BC application and `Bifröst`.

Three requirements were taken into consideration for the development of `Bifröst`: *(i)* flexibility, *(ii)* modularity, and *(iii)* ease of use.

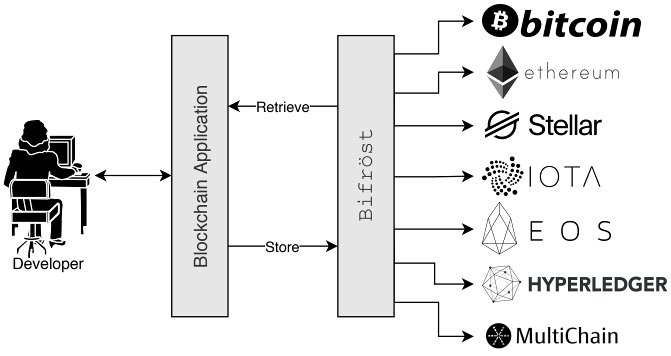

Fig. 1. User Interaction with Blockchain Application and `Bifröst`

The first, **flexibility**, is given by allowing the user to store a string in the BC, which could represent any arbitrary data, *e.g.,* an SHA-256 hash. The second, **modularity**, is provided by implementing the adapter of each BC with a standard interface, simplifying adding adapters to new BC.

Finally, the third, **ease of use**, is achieved by abstracting technical details from the underlying BC implementation, providing simple API functions (cf. Section III-C), which require only two inputs from the user, being the data to be included and the BC identification. Moreover, using *docker* to run BC Remote Procedure Call (RPC) servers improves **ease of use** as its employment minimizes compatibility problems because the nodes execute in an isolated and replicable environment.

B. `Bifröst` *Components*

`Bifröst` relies on a notary scheme to interact with multiple BCs. This scheme was selected because it is a straightforward manner to manage data stored on different BCs without changing the underlying BCs implementation or maintaining parallel chains. `Bifröst` consists of three main components: (1) the API, (2) the BC adapters, and (3) a database. Figure 2 presents an overview of these three parts and the data flow between them using the `store` function as an example.

1) The *API* is the entry point for interacting with `Bifröst`. It consists of the exposed functions `store`, `retrieve`, and `migrate`. The API is responsible for receiving the user input and communicating with the correct BC adapter.
2) The *Adapters* convert the user input into a transaction which is subsequently transmitted to the BCs nodes. The nodes forward the transactions to the BC network, where miners will process them. In the case of the `retrieve` function, the adapter requests the data from the BC instead of creating a transaction. A new adapter must be implemented to allow support for each new BC.
3) The *Databases* store the necessary credentials for the transactions, and stores the transaction hash after a successful transaction has been included in the BC. This hash can be later used to retrieve the stored data.

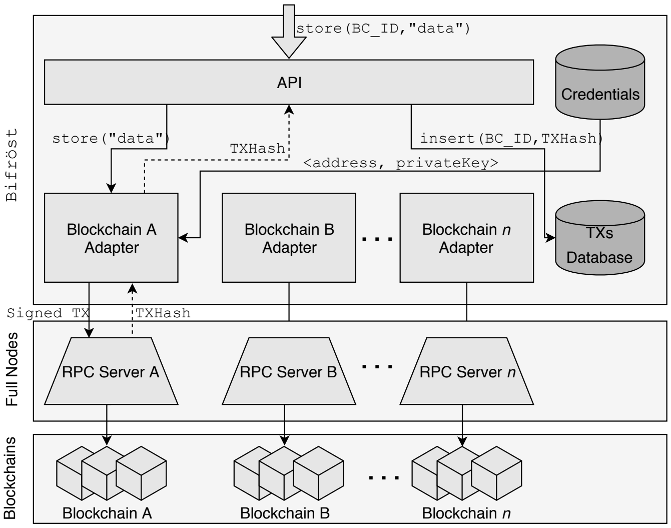

Fig. 2. General `Bifröst` Architecture and `store` Function Flow

C. `Bifröst` *Implementation*

As described in Section III-B, `Bifröst` is composed of three main components: (1) the API, (2) the BC adapters, and (3) a database. Thus, they were implemented, in a Python prototype, to evaluate the whole architecture design and workflow.

1) API: It has three exposed functions (cf. Listing 1). Their implementation is described in the following paragraphs.

The `store(text, blockchain)` function receives the data in the form of a string and the identification of the BC as input. It then stores the string on the defined BC, waits for the defined transaction confirmation time, and returns the transaction hash.

The `retrieve(transaction_hash)` function receives a transaction hash as a parameter and returns the string previously stored in the BC. The corresponding BC used to retrieve the data is automatically recognized by the API using a query in the database.

The `migrate(transaction_hash, blockchain)` function retrieves a stored string from one BC and copies it to another BC. The parameters are the transaction hash of the origin and the name of the target BC. It should be noted that it is not possible to delete data from the BC and management of which transaction hash is valid depends on the user.

```python
def store(text, blockchain):
    adapter = Adapter[blockchain]
    tx_hash = adapter.store(text)
    return tx_hash
def retrieve(transaction_hash):
    blockchain = database.find_blockchain(tx_hash)
    adapter = Adapter[blockchain]
    text = adapter.retrieve(tx_hash)
    return text
def migrate(transaction_hash, blockchain):
    value = retrieve(tx_hash)
    new_hash = store(value, blockchain)
    return new_hash
```

Listing 1. Exposed API Functions

TABLE I
BLOCKCHAIN ADAPTER IMPLEMENTATION DETAILS

Blockchain	Library Name	Node type	Protocol	Network
Bitcoin	python-bitcoinrpc	Full Node	RPC	Public Testnet
Ethereum	web3.py	Full Node	RPC	Local Testnet
Stellar	py-stellar-base	Remote Node	HTTP	Public SDF Testnet
IOTA	PyOTA	Remote Node	RPC	Public Testnet
EOS	eosjs_python	Remote Node	RPC	Public Jungle Testnet
Hyperledger	sawtooth_sdk	Full Node	HTTP	Local Testnet
Multichain	python-bitcoinrpc	Full Node	RPC	Local Testnet
PostgreSQL	psycopg2	PostgreSQL	Postgres	Local Postgres

TABLE II
OVERVIEW OF SELECTED BLOCKCHAINS

Blockchain	Type	Consensus	Finality	Blocktime [s]	Confirmation After
Bitcoin	Public	PoW	No	600	6 blocks
Ethereum	Public	PoW	No	15	7 blocks
Stellar	Public	SCP	Yes	5	1 block
IOTA	Public	IOTA	Yes	60	1 block
EOS	Public	dPoS	Yes	0.5	1 block
Hyperledger	Private	PoET	Yes	20	1 block
Multichain	Private	PoA	Yes	15	1 block

Proof-of-Work (PoW), Stellar Consensus Protocol (SCP), delegated PoS (dPoS), Proof-of-Elapsed-Time (PoET), Proof-of-Authority (PoA)

2) Blockchain Adapters and Nodes: Table I provides an overview of the different adapters. Seven adapters to different BCs were implemented. Additionally, a PostgreSQL adapter was implemented to allow the data to be stored in a traditional database instead of a BC. Apart from the BC and library name, Table I also presents information about the node type, connection, and network type (*e.g.,* local or public testnet). For example, the Bitcoin implementation connects to a full node with access to the public Bitcoin testnet and uses the RPC protocol to communicate with the adapter.

3) Adapter Implementation: Internally, the `store` function constructs a raw transaction using the provided string. This transaction is then signed using the stored private key and sent using the RPC or HTTP protocol. After receiving the transaction hash, the function `confirmation_check` validates the existence of the transaction after waiting for a specified period. Thus, it is confirmed that the block was included and finality is given. If the confirmation was successful, the hash is saved to an SQLite database, referencing a BC identifier and including a timestamp.

The `retrieve` function requires the transaction hash as a parameter. The transaction is retrieved from the BC using RPC or HTTP. Afterwards, it extracts the string from the obtained transaction. Listing 2 presents an excerpt from the implementation. The parameter `cls` reflects the chosen BC in the form of an adapter class.

```
def store(cls, text):
    tx = cls.create_tx(text)
    signed_tx = cls.sign_tx(tx)
    tx_hash = cls.send_raw_tx(signed_tx)
        if(cls.confirmation_check(tx_hash)):
            cls.add_tx_to_database(tx_hash)
            return tx_hash
        else:
            raise LookupError(’Tx not confirmed!’)


def retrieve(cls, tx_hash):
    transaction = cls.get_tx(tx_hash)
    data = cls.extract_data(tx)
    return cls.to_text(data)
```

Listing 2. Adapter Implementation

4) Confirmation Times: Submitting a transaction to a BC does not guarantee that it will be persisted in the BC. For example, the transaction might be refused by the network because of a low transaction fee or due to the underlying BC design [35]. Moreover, even if it is included, block finality needs to be ensured. Skipping this finality check can lead to transactions located on a fork of the BC, which is abandoned in the future. This issue is tackled by waiting x amount of time before writing the transaction to the database. If the transaction is not found after this time, it is considered invalid and discarded. The value of x depends on the BC. For example, considering Table II, Bitcoin requires a waiting time x of 3600 s (6 blocks × 600 s), whereas Ethereum requires an x of 105 s (7 blocks × 15 s) to have a high probability of finality. Such an approach of waiting x amount of time is simplistic because, at the moment, `Bifröst` is a Proof-of-Concept (PoC). Thus, there are still open transaction-related aspects (*e.g.,* robustness and resilience) to be addressed.

Even if block finality is given, transactions do not happen instantly. Especially with a PoS consensus, there can be multiple rounds of communication between the nodes until two-thirds of them agree on one solution. As there can be delays in the communication, a minimum waiting time of 20 seconds was implemented [14]. The prototype was tested using public testnets or local private networks. Even though the consensus mechanism of those networks often differs from the mainnet, the waiting times were set to match with the confirmation times on the mainnets.

5) Database: The database is responsible for storing a list of supported BCs, credentials for the BCs, and transaction hashes. In the prototype, an SQLite database and the Python library $sqlite$3 were selected. SQLite was chosen because of the simplicity of storing the whole database without maintaining a dedicated server [30]. The next items describe the tables which compose the database.

- **Blockchains**: The BC table has two columns, being a unique id and the $name$ of the BC. This table is used to link the BCs to the transactions and credentials.
- **Credentials**: The credentials table has six columns. The unique id used to map the identity of users to their credentials. The *blockchain_id* connects the credentials to the corresponding BC. Furthermore, $address$ stores the public key and key the private key of the BC account. The $user$ and $password$ provide the credentials for the RPC or HTTP client. The necessary credentials depend on the BC. For example, Stellar does not need $user$ and $password$, and IOTA does not need key, as zero-value transactions do not require a sender; thus, no signature.

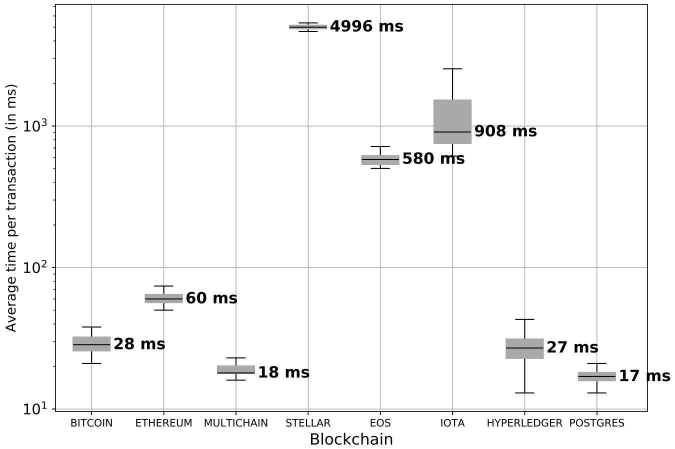

Fig. 3. Performance Measurements on Supported Blockchains

- **Transactions**: After a transaction is completed, its hash is stored in the column $hash$. The BC that the transaction is included is identified by the $blockchain\_id$ column. Furthermore, the *timestamp* of issuing is included in the column $issued\_at$.

IV. Evaluations and Discussion

The evaluation of the `Bifröst` prototype was performed concerning performance, security, and data storage size. While an analysis of the prototype performance using a sample of 1000 transactions per BC is presented first, a security analysis is conducted second. Finally, data size limitations of the current implementation are discussed.

A. Performance Analysis

A performance analysis was performed to measure both the overhead and the stability of the application. The measurements were performed on a MacBook Pro 2017, Dual-Core i5 @ 2.3 GHz, 8 GB RAM, macOS Mojave 10.14.1 with Python 3.6.6. At total, 1000 measurements were gathered for each BC, except for Bitcoin. In case the of Bitcoin, 100 samples were taken because of the limit of 25 unconfirmed transactions on the Bitcoin RPC server. As some remote nodes impose restrictions on how many transactions can be sent over a specific time, the measurements were completed in batches of 25 transactions each. PostgreSQL was included in the analysis to compare BC performance with a regular database. Even though all the 1000 transactions per BC were completed without issues, a stability and scalability evaluation must be conducted to consider the prototype stable.

Figure 3 depicts the outcome of the performance measurements. The *x*-axis represents the different BC adapters, and the *y*-axis represents the average time per transaction (in milliseconds). It can be seen that there is a performance difference between using local nodes and remote nodes. Remote nodes were used by Stellar, IOTA and EOS as presented in Table I. Noticeably, the Bitcoin client using a full local node with a public testnet was faster than other BCs (*e.g.,* Ethereum) which were using a local node with a private testnet. Multichain presented a similar performance as the PostgreSQL database due to the fact that Multichain is a private BC focused on data streams. However, Multichain requires more operations (*e.g.,* raw transaction creation and cryptographic signature) that PostgreSQL does not. It has to be noted that with some adapters (*e.g.,* Hyperledger), multiple transactions could be put into one batch. Thus, only one batch would need to be transmitted for multiple transactions, which would result in higher throughput.

B. Security Analysis

In the following sections, relevant security aspects are discussed. First, it is discussed the security of storing the private keys in a central server. Then, the security of local and remote nodes is described. Finally, the centralization issue is addressed.

1) Private Key Management: A major concern is to secure the private keys from attackers. A successful attack in this vector could have two implications. First, an attacker could spend the funds linked to the account. As this solution is made for storing data on the BC, the funds would only need to cover the transaction costs. Keeping only the necessary funds for the transactions would; therefore, mitigate this risk. Second, an attacker could change the data by storing some arbitrary data which would be hard to distinguish from the genuine transactions.

The presented prototype is a PoC, implemented to assess `Bifröst`'s feasibility. Thus, the private key is stored in plain text on an SQLite database. In this sense, if an attacker would get access to the server, it would be trivial to access the private keys. One solution to circumvent this problem is to encrypt the private key with a symmetric key which is based on a password set by the user. Thus, only allowing a transaction to occur if the user temporarily decrypts the private key. Another possible solution consists of a *multisignature* transaction scheme, where more than one key is necessary to sign transactions, and they are stored in another, more secure location.

2) Local and Remote Nodes: Another risk is the exposure of the RPC or HTTP port by the BC node. If the node additionally holds the private keys in a local wallet, transactions could be initiated remotely. However, all of the BC adapters in this prototype sign the transactions locally, meaning there is no need for the node to hold the private keys. Nevertheless, the ports could be used to transmit unrelated, malicious transactions and Distributed Denial-of-Service (DDoS) attacks targeting the node could be executed. In cases where public remote nodes are used, the owner of those nodes could potentially block a transaction from being processed. Furthermore, the availability of the whole system could become an issue.

Nonetheless, these security issues could be circumvented by using full local nodes and allowing RPC connections only from the local machine (*i.e.,* localhost) or restricting connections to trusted IP addresses. DDoS attacks on `Bifröst` can be mitigated by implementing a request limiter, throttling the processing of requests by IP or request type.

TABLE III
MAXIMUM DATA SIZE IN DIFFERENT BLOCKCHAINS

Blockchain	Maximum String Size
Bitcoin	80 Byte [3]
Ethereum	46 kByte [1]
Stellar	28 Byte [31]
EOS	256 Byte [11]
IOTA	1300 Byte [20]
Hyperledger	20 Byte [17]
Multichain	80 Byte [3]

3) Centralization: One of the main benefits of the BC technology is the removal of trust by relying on decentralization and cryptographic properties. Thus, it would be advantageous if this property holds for this solution as well. However, using a notary scheme implies that a user needs to trust in the notary, *i.e.,* the host of the application. First, the notary has access to the private keys, which makes it vulnerable to the issues discussed in Section IV-B1. Further, the notary controls the application and the node. Therefore, it is able to arbitrarily alter the original transactions, *e.g.,* alter the sender or the data or censor certain transactions.

Nevertheless, BC applications are rarely trust-free, and there are always layers which require a certain amount of trust, such as monitoring applications in Internet-of-Things (IoT) applications. Taking `Bifröst` as an example, the first layer of trust is the BC underlying code and cryptographic properties. The second layer is the RPC server and the RPC client. The notary is the third layer of trust. After that, other layers must be trusted, *e.g.,* the user's hardware and software. From this perspective, the notary scheme adds a layer which requires trust, but is still only one out of many. However, the amount of trust needed can be minimized by running the approach in a trusted computing environment, such as the Intel's Software Guard Extensions (SGX) [2].

C. Data Size Analysis

Transaction-focused BCs, *e.g.,* Bitcoin and Stellar, are not designed to store arbitrary data. Thus, there is a limit on the amount of data included in a transaction. Table III summarizes how much data is possible to store in the different BCs supported by `Bifröst`. Note that on some BCs, *e.g.,* Ethereum, this restriction could be circumvented by using Smart Contracts (SC). In an SC deployed in Ethereum, the storage of data can be divided over more than one block, bypassing the maximum data size limitation.

Due to the presented data limitation and the fact that the BC technology was not conceived as a database, but rather only a distributed ledger, holding only transaction-related data, it can be seen that it is not cost-efficient to store large amounts of data on BCs. In this sense, a (decentralized) database can be used to store the "raw" data, and a hash of this data is stored in the BC. Thus, the validity of the data can be verified at any time while the data is stored efficiently.

V. SUMMARY AND FUTURE WORK

Due to trade-offs in terms of features, privacy and efficiency, there will be multiple, coexisting blockchains (BC) in the future, each covering a subset of the possible use cases. Thus, interoperability is needed to link different BCs by exchanging information and assets. This paper addressed BC interoperability by providing a novel solution, called `Bifröst`, to store, retrieve, and migrate data on different BCs using a notary scheme. The notary scheme was chosen because it does not require any modification of the underlying implementation of BCs. `Bifröst` abstracts technical BC functions, *e.g.,* the creation, signing, and transmission of raw transactions, by providing a simple Application Program Interface (API) to developers. Thus, such an API allows developers to create innovative BC-based applications that transparently interact with several BCs, *e.g.,* Bitcoin, Ethereum, and Hyperledger.

Moreover, a Python prototype of `Bifröst` was described, and seven different BC adapters were implemented. These adapters contain BC-specialized code, including the transaction template, data translation, and RPC communication. Code listings of the prototype were presented to demonstrate these abstractions and interactions. Further, a performance and security analysis was conducted on the prototype implemented. This analysis showed that the necessary time to create and send transactions in private BCs (*e.g.,* Multichain) is lower than in public BCs (*e.g.,* Ethereum). The security analysis presented attack vectors that must be addressed, such as the private key storage and single point of failure. Nevertheless, the implementation showed that it is possible to abstract BC technical details and provide a simple interface to developers to interact with multiple BCs, enabling BC interoperability.

In conclusion, `Bifröst` provides a *flexible*, *modular*, and *easy to use* API, which allows BC-based applications to be developed without adding the overhead of understanding complex technical detail of each BC or requiring changes in the BC code. Moreover, `Bifröst` allows the developer to select and deploy BC nodes with the use of *docker*, which decreases configuration and deployment times by relying on *containers*. Thus, as `Bifröst`, different than existing BC interoperability projects, combines all these aspects in a single project, it is a promising solution to addresses BC interoperability.

Future work plans for the `Bifröst` API to include *(i)* security improvements, *(ii)* transaction-related error handling, *(iii)* communication to smart contracts, and *(iv)* research towards developing a decentralized version of `Bifröst`.

ACKNOWLEDGMENTS

This work has been supported by Komission für Technologie und Innovation (KTI), Switzerland (CTI-No. 26996.L PFES-ED). Also, the authors would like to acknowledge manifold discussions with modum.io, especially Stefan M. Weber, Sacha Uhlmann (modum.io), and Thomas Bocek from the Hochschule für Technik Rapperswil (HSR).

REFERENCES

[1] Afri, "Is There a Limit for Transaction Size?" 2018, Last access March 20, 2019. [Online]. Available: https://ethereum.stackexchange.com/questions/1106/is-there-a-limit-for-transaction-size

[2] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing," August 2013, Last access July 11, 2019. [Online]. Available: https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing

[3] M. Bartoletti and L. Pompianu, "An Analysis of Bitcoin OP_RETURN Metadata," 2017, Last access May 5, 2019. [Online]. Available: http://arxiv.org/abs/1702.01024

[4] Blockgeeks, "Most Comprehensive AION Blockchain Guide," 2018, Last access March 20, 2019. [Online]. Available: https://blockgeeks.com/guides/aion-blockchain/

[5] btcrelay.org, "BTC Relay," 2017, Last access March 20, 2019. [Online]. Available: http://btcrelay.org/

[6] V. Buterin, "A next generation smart contract & decentralized application platform," 2014, Last access March 20, 2019. [Online]. Available: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf

[7] ——, "Chain Interoperability," 2016, Last access March 20, 2019. [Online]. Available: https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/5886800ecd0f68de303349b1/1485209617040/Chain+Interoperability.pdf

[8] CoinMarketCap, "Bitcoin price, charts, marketcap, and other metrics," 2019, Last access March 26, 2019. [Online]. Available: https://coinmarketcap.com/currencies/bitcoin/

[9] ——, "Cryptocurrency Market Capitalizations," 2019, Last access March 26, 2019. [Online]. Available: https://coinmarketcap.com/

[10] B. Deme, "What is Herdius?" 2018, Last access March 20, 2019. [Online]. Available: https://medium.com/herdius/what-is-herdius-3831a47cfb6

[11] EOS.IO, "What is the Character Limit of MEMO?" 2018, Last access March 20, 2019. [Online]. Available: https://github.com/EOSIO/eos/issues/4296

[12] Grayblock, "Interoperability The Holy Grail of Blockchain," 2018, Last access March 20, 2019. [Online]. Available: https://medium.com/coinmonks/interoperability-the-holy-grail-of-blockchain-eb078e1a29cc

[13] G. Greenspan, "MultiChain-White-Paper.pdf," 2015, Last access March 20, 2019. [Online]. Available: https://www.multichain.com/download/MultiChain-White-Paper.pdf

[14] A. Grigorean, "Latency and Finality in Different Cryptocurrencies," 2018, Last access March 20, 2019. [Online]. Available: https://hackernoon.com/latency-and-finality-in-different-cryptocurrencies-a7182a06d07a

[15] T. Hardjono, A. Lipton, and A. Pentland, "Towards a Design Philosophy for Interoperable Blockchain Systems," 2018, Last access July 10, 2019. [Online]. Available: http://arxiv.org/abs/1805.05934

[16] Herdius, "Herdius Whitepaper," 2017, Last access March 20, 2019. [Online]. Available: https://herdius.com/whitepaper/Herdius_Whitepaper_1.1.pdf

[17] Intel Corporation, "IntegerKey Transaction Family," 2018, Last access March 20, 2019. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications/integerkey_transaction_family.html#state

[18] Interledger W3C Community Group, "Interledger," 2018, Last access March 20, 2019. [Online]. Available: https://interledger.org/

[19] ——, "Interledger Architecture," 2018, Last access March 20, 2019. [Online]. Available: https://interledger.org/rfcs/0001-interledger-architecture/

[20] iota.org, "The Anatomy of a Transaction," 2018, Last access March 20, 2019. [Online]. Available: https://iota.readme.io/v1.2.0/docs/the-anatomy-of-a-transaction

[21] D. Joseph, J. Kanna, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, "OCALA: An Architecture for Supporting Legacy Applications over Overlays," in *3rd Symposium on Networked Systems Design and Implementation (NSDI 2006)*, San Jose, CA, USA, May 2006, pp. 267–280.

[22] D. Kajpust, "Blockchain Interoperability: Cosmos vs. Polkadot," 2018, Last access March 20, 2019. [Online]. Available: https://medium.com/@davekaj/blockchain-interoperability-cosmos-vs-polkadot-48097d54d2e2

[23] A. M. Kurt, "Zero to Monero - First Edition," 2018, Last access March 20, 2019. [Online]. Available: https://pdfs.semanticscholar.org/16a8/c0aa45bd09830b8c5115c2c1e441f177fc82.pdf

[24] J. Kwon and E. Buchman, "Cosmos | Cosmos Documentation," 2018, Last access March 20, 2019. [Online]. Available: https://cosmos.network/docs/resources/whitepaper.html

[25] S. Matthew, "AION White Paper," 2018, Last access March 20, 2019. [Online]. Available: https://aion.network/media/en-aion-network-technical-introduction.pdf

[26] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, Last access March 20, 2019. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[27] K. Nelaturu, "Blockchain Interoperability Sidechains," 2018, Last access March 20, 2019. [Online]. Available: https://medium.com/coinmonks/blockchain-interoperability-sidechains-e8204b8c2a10

[28] NEO, "NEO Documentation," 2014, Last access March 20, 2019. [Online]. Available: http://docs.neo.org/en-us/index.html

[29] E. J. Scheid, B. B. Rodrigues, and B. Stiller, "Toward a Policy-based Blockchain Agnostic Framework," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, Washington, USA, April 2019, pp. 609–613.

[30] SQLite, "SQLite Is Serverless," 2018, Last access March 20, 2019. [Online]. Available: https://www.sqlite.org/serverless.html

[31] Stellar.org, "Transactions," 2018, Last access March 20, 2019. [Online]. Available: https://www.stellar.org/developers/guides/concepts/transactions.html

[32] The ARK Crew, "ARK Whitepaper," 2018, Last access March 20, 2019. [Online]. Available: https://ark.io/Whitepaper.pdf

[33] S. Thomas and E. Schwartz, "A Protocol for Interledger Payments," 2016, Last access March 20, 2019. [Online]. Available: https://interledger.org/interledger.pdf

[34] Wanchain, "Wanchain Whitepaper," 2018, Last access March 20, 2019. [Online]. Available: https://wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf

[35] I. Weber, V. Gramoli, A. Ponomarev, M. Staples, R. Holz, A. B. Tran, and P. Rimba, "On Availability for Blockchain-Based Systems," in *IEEE 36th Symposium on Reliable Distributed Systems (SRDS 2017)*, Hong Kong, China, September 2017, pp. 64–73.

[36] G. Wood, "Polkadot: Vision for a Heterogeneous Multi-Chain Framework," 2017, Last access March 20, 2019. [Online]. Available: https://polkadot.network/PolkaDotPaper.pdf