Enabling Cross-Jurisdiction Digital Asset Transfer

Rafael Belchior[1], André Vasconcelos[1], Miguel Correia[1], Thomas Hardjono[2]
[1]*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Portugal*
[2]*MIT Connection Science & Engineering, Massachusetts Institute of Technology – USA*

***Abstract*—Enabling blockchain-based digital asset exchanges requires blockchain interoperability capabilities. Although some solutions have been proposed in recent years, asset and cryptocurrency transfers across legal jurisdictions are still an unsolved problem. To realize this vision, we propose HERMES, a fault-tolerant middleware that connects blockchain networks, enabling the transfer of data and value across legal jurisdictions. Hermes is based on the Open Digital Asset Protocol (ODAP), an asset transfer protocol. Hermes utilizes a novel mechanism called ODAP-2PC and decentralized logging that can solve disputes regarding asset exchange. We find Hermes to fill an existing gap: the technical infrastructure that can constitute the basis for legislating and regulating cross-chain transfers, enabling the future of finance.**

I. INTRODUCTION

There is today a resurgent interest in the potential use of digital currencies and virtual assets as the foundation of the next generation digital economy, and several trading nations including are promoting new platforms for digital transactions [19]. Significant attention is being placed on blockchain and decentralized ledger technology (DLT) as a possible technological layer underlying these platforms.

Similar to the architecture of the Internet – which consists of multiple independent autonomous systems with many routing domains – the architecture of a global network of DLTs will require interoperability across distinct DLT systems [13]. The Internet solved the problem of inter-domain IP routing by employing gateway routers which implement cross domain protocols such as the Border Gateway Protocol (BGPv4).

We believe that similar to these Internet routing gateways, the global network of DLTs will require gateways that permit digital currencies and virtual assets to be transferred seamlessly between these systems. However, this entails designing blockchain gateways that can operate reliably and can withstand attacks. The implication here is that a crash-recovery strategy must be a core design factor of blockchain gateways, where specific recovery protocols can be designed as part of the digital asset transaction protocol between gateways.

In the current work we propose HERMES, a middleware for blockchain interoperability that focuses on reliability. The main component of Hermes is an extension of the ODAP protocol [15], called ODAP-2PC, a novel protocol that supports ACID properties in cross chain transactions.

The contributions of this paper are two-fold: first, we present the HERMES fault-tolerant middleware, instantiated with the ODAP protocol and ODAP-2PC. Secondly, we provide a comprehensive discussion on Hermes as a solution for blockchain interoperability, focusing on consistency, performance, and decentralization. We also briefly explore a use case for cross-jurisdiction asset transfers, illustrating how one can leverage Hermes to achieve blockchain interoperability compliant with legal and regulatory frameworks.

The rest of this paper is structured as follows: Section II presents the background. After that, we introduce the gateway concept and Hermes, in Section III. After, in Section IV, we present ODAP-2PC. Section V, presents a use case that benefits from Hermes. Section VI presents our discussion on gateways, ODAP, and ODAP-2PC in the light of the presented research questions. The related work follows, in Section VII. Finally, we conclude the paper in Section VIII.

II. BACKGROUND

This section presents the background on fault tolerance models and nomenclature both on logging and blockchain interoperability.

Atomic Commit Protocols: An atomic commit protocol (ACP) is a protocol that guarantees a set of operations being applied as a single operation. An atomic transaction is indivisible and irreducible: either all operations occur, or none does. ACPs consider two roles: a *Coordinator* that manages the execution of the protocol, and *Participants* that manage the resources that must be kept consistent. ACPs assume stable storage with a write-ahead log (a history of operations are persisted before actions are executed). Example of ACPs are the two-phase commit protocol, 2PC, the three-phase commit protocol, 3PC, and non-blocking atomic commit protocols [7].

2PC achieves atomicity even in case of temporary system failure, accounting for a wide adoption both in the academia and in the industry. It has two phases: the voting phase and the commit phase. In the voting phase, the coordinator prepares all participants to take place in a distributed transaction by inspecting each participant's local status. Each participant executes eventual local transactions required to complete the distributed transaction. If those are successful, participants send a *YES* response to the coordinator, and the protocol continues. Else, if the *NO* response is sent, it means that the participant chose to abort; this happens when there are problems at the local partition. Next, in the commit phase, when the coordinator obtains *YES* from all participants, a *COMMIT* message is sent to the participants that voted *YES*. This message triggers the execution of local transactions that implement the distributed transaction. Otherwise, the coordinator sends an *ABORT* message, triggering a rollback on each local partition.

Logging: A log $\mathcal{L}$ is a list of log entries $\{l_1, l_2, ..., l_n\}$ such that entries have a total order, given by the time of its creation. A log is considered *shared* when a set of nodes can read and write from the log. On the other hand, a log is *private* (or *local*) when only one node can read and write it. Logs are associated to a process p running *operations* on a certain node. To manipulate the log, we define a set of *log primitives*, that translate *log entry requests* from a process p into log entries. The log primitives are *writeLogEntry* (writes a log entry), *getLogLength* (obtains the number of log entries), and *getLogEntry(i)* (retrieves a log entry l_i). A log entry request typically comes from a single event in a given protocol.

A *log storage API* provides access to the primitives. Log entry requests have the format `<phase, step, operation, nodes>`, where the field *operation* corresponds to an arbitrary command, and the field *nodes* to the parties involved in the process p. We define four operations types to provide context to the protocol being executed. Operation type *init-* states the intention of a node to execute a particular operation, and operation *exec-* expresses that the node is executing the operation. The operation type *done-* states when a node successfully executed a step of the protocol, while *ack-* refers to when a node acknowledges a message received from another. Conversely, we use the type *fail-* to refer to when an agent fails to execute a specific step. The field *nodes* contains a tuple with a node A issuing a command, or a node A commanding a node B the execution of a command c, if the form is A or $A \rightarrow B$ (c may be ommitted), respectively.

Blockchain Interoperability: A recent survey classifies blockchain interoperability studies in three categories: Cryptocurrency-directed interoperability approaches, Blockchain Engines, and Blockchain Connectors [6]. Cryptocurrency-directed approaches are directed to enabling the transfer of digital assets (e.g., cryptocurrencies) across homogeneous and heterogeneous blockchains. The cryptocurrency-directed approaches typically rely on protocols leveraging public blockchains, as they assume that gateways are not trusted. As a result, these approaches are difficult to integrate with permissioned blockchains, that support with arbitrary assets and smart contracts. The second category are the blockchain engines, which enable creating application-specific blockchain that can communicate with its other instances. These solutions can benefit from implementing gateways, providing each application-specific blockchain (e.g., applications running on a parachain) self-sovereignty, regarding communications with outter blockchains. The third category, blockchain connectors include trusted relays, blockchain agnostic protocols, blockchain of blockchains solutions, and blockchain migrators. Trusted relays are software components, typically centralized, where escrows route cross-blockchain transactions.

III. The Architecture of Hermes

This section introduces the gateway concept and Hermes.

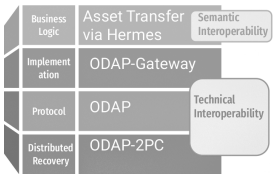

Fig. 1: The architecture of Hermes

A. Blockchain Gateways

A *gateway* is a DLT system node based on an underlying DLT-based system and functionally capable of performing CC-Tx, including asset transfers [13]. A *primary gateway* is the DLT system node acting as a gateway in a CC-Tx. Primary gateways may be supported by *backup gateways* for fault tolerance. For gateways to be crash fault-tolerant, they keep track of each operation they do in a log (of operations). The log is a sequence of log entries, each entry representing a step of the gateway protocol. A gateway protocol specifies the set of messages and procedures between two gateways for their correct functioning. The gateway protocol considered in this paper is ODAP [15], [5].

B. Blockchain Interoperability with Hermes

Hermes is a gateway system that enables DLT interoperability based on gateways. This system has four layers, allowing for end-to-end communication. The gateway protocol layer implements any standards that a specific gateway implementation needs to comply with (e.g., travel rule [12]). ODAP, a gateway-based CCCP that realizes asset transfers, allows realizing technical interoperability for asset transfers. It is built on top of a distributed recovery protocol, providing reliability in the presence of crashes. On top of the gateway protocol stands a concrete implementation of a gateway. Jointly with the gateway protocol, it provides support for semantic interoperability [6], unlocking the value level. More specifically, in the value level, the business logic is defined for clients using gateways, allowing them to attribute value to the assets exchanged with ODAP. The whole stack provides atomicity, consistency, isolation, and durability of CC-Tx. Figure 1 represents Hermes' architecture.

Our architecture is flexible and modular, as its components are pluggable. Modularity allows building a system that can be adapted to specific needs. In this paper, we instantiate Hermes with the ODAP-Gateway, the ODAP CCCP, and its crash fault-tolerant distributed recovery protocol, ODAP-2PC. The whole stack allows a business case, gateway-to-gateway asset transfers, providing the basis for unidirectional asset transfers, expressed in detail in Section V. The Hermes Client allows to implement the business logic, realizing semantic interoperability.

IV. Hermes

In this section, we present the main building blocks of Hermes: ODAP and Hermes' distributed recovery mechanism, ODAP-2PC.

A. ODAP and Properties

The ODAP protocol is a gateway-to-gateway unidirectional asset transfer protocol that uses gateways as the systems conducting the transfer [15]. An asset transfer is represented in the form $T : G_1 \overset{a,x}{\rightarrow} G_2$, where a source gateway G_1 transfers x asset units from type a from a source ledger $\mathcal{B}_S$ to a recipient ledger $\mathcal{B}_R$, via a gateway G_2.

The source gateway issues a transfer such that x asset units will be unavailable at the source DLT and become available at the target DLT. A recipient gateway is the target of an asset transfer, i.e., follows instructions from the source gateway. Hermes provides as strong durability guarantees as to the underlying durability guarantees of the chosen data store. If the datastore is a blockchain, Hermes can be considered to achieve transaction durability, if transactions are immutable and permanently stored in a secure decentralized ledger.

In ODAP, a client application interacts with its local gateway (source gateway GS) over a Type-1 API. The existence of this API allows the client to provide instructions to GS (corresponding to the source gateway) concerning the assets stored in the source DLT and the target DLT (via the recipient gateway, GR). It is possible that the client has complex business logic code that triggers behavior on the gateways. Hence, ODAP allows three flows: the *transfer initiation flow*, where the process is bootstrap, and several identification procedures take place; the *lock-evidence flow*, where gateways exchange proofs regarding the status of the asset to be transferred; and the *commitment establishment flow*, where the gateways commit on the asset transfer.

B. ODAP-2PC

One of the key deployment requirements of gateways for asset transfers is a high degree of gateways availability. A distributed recovery procedure then increases the resiliency of a HERMES gateway by tolerating faults. Next, we present an overview of ODAP-2PC.

The protocol is crash fault-tolerant, so the gateways are trusted to operate the ODAP protocol as specified unless they stop. We envisage ODAP-2PC to support two strategies to increase the availability of gateways [5]: (1) *self-healing mode:* after a crash, a gateway eventually recovers, informs other parties of its recovery, and continues executing the protocol; (2) *primary-backup mode:* after a crash, a gateway may never recover, but that timeout can detect this failure; if a node is crashed indefinitely, a backup is spun off, using the log storage API to retrieve the log's most recent version.

In both modes, logs are written before operations (write-ahead) to provide atomicity and consistency to the protocol used for asset exchange. The log-data is considered as resources that may be internal to the DLT system, accessible to the backup gateway and possible other gateway nodes.

There are several situations when a crash may occur. Figure 2 represents the crash of $\mathcal{G}_S$ before it issues a validation operation to $\mathcal{G}_R$ (steps 1 and 2). Both gateways keep their log storage APIs, with γ_{local}. For simplicity, we only represent one log storage API. In the self-healing mode, the gateway

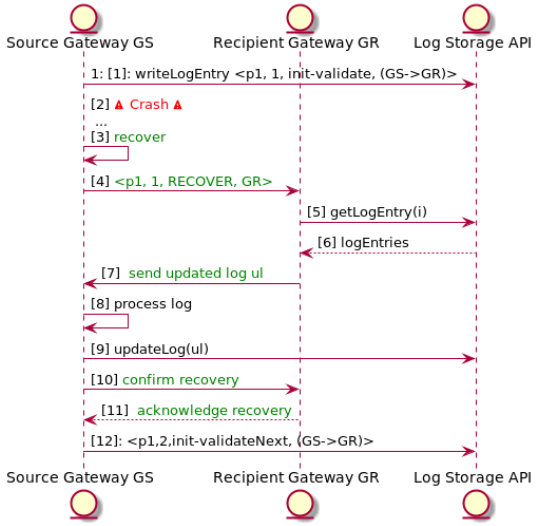

Fig. 2: $\mathcal{G}_S$ crashing before issuing init-validation to $\mathcal{G}_R$

eventually recovers (step 3), building a recovered message in the form <phase, step, RECOVER, nodes> (step 4). The non-crashed gateway queries the log entries that the crashed gateway needs (steps 5, 6). In particular, $\mathcal{G}_S$ obtains the necessary log entries at step 7 and compares them to its current log. After that, $\mathcal{G}_S$ attempts to reconcile the changes with its current state (step 8). Upon processing, if both log versions match, then the log is updated, and the process can continue. If the logs differ, then $\mathcal{G}_S$ calls the primitive updateLog, updating its log (step 9) and thus allowing the crashed gateway to reconstruct the current state. In this particular example, step 9 would not occur because operations exec-validate, done-validate, and ack-validate were not executed by $\mathcal{G}_R$. If the log storage API is on the shared mode, no extra steps for synchronizations are needed. After that, it confirms a successful recovery (steps 10, 11). Finally, the protocol proceeds (step 12).

Figure 3 represents a recovery scenario requiring further synchronization. At the retrieval of the latest log entry, $\mathcal{G}_S$ notices its log is outdated. It updates it, upon necessary validation, and then communicates its recovery to $\mathcal{G}_R$. The process then continues as normal. (for instance, corresponding to exec-validate, done-validate, and ack-validate)

ODAP-2PC is a 2PC protocol able to detect and recover from crashes, delivering the effort to execute an asset transfer starting at ODAP's phase 3: the commitment establishment flow. Crashes at other phases of the ODAP are handled by the self-healing mechanism, supported by the messaging and logging mechanism, as depicted by Figures 2 and 3. ODAP-2PC considers two parties: the coordinator $\mathcal{G}_S$, and the participant $\mathcal{G}_R$. The coordinator manages the protocol execution while the participant follows the coordinator's instructions. In phase 3, these two parties exchange sensitive messages that include the lock and unlocking of assets. Those messages may not arrive due to failures (e.g., communication failures, gateway crash due to power outage). To detect crashes, we use a timeout

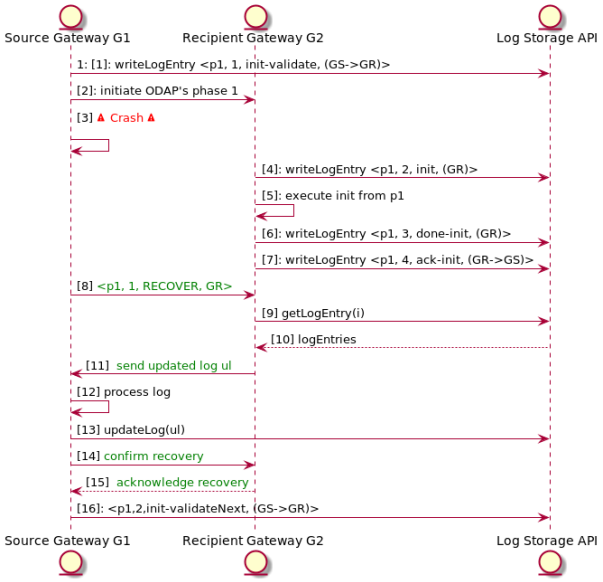

Fig. 3: $\mathcal{G}_S$ crashing after issuing the init command to $\mathcal{G}_R$

δ_C. However, processes may wait for the crashed gateway to recover for an unbounded timespan, wasting resources (e.g., locked assets). To avoid this, we introduce an additional timeout $\delta_{rollback}$. When a gateway does not recover before this timeout, a *timeout action* is triggered, corresponding to the *rollback protocol*. A possible rollback protocol cancels the current transactions by issuing transactions with the contrary effect, guaranteeing the consistency of the DLT whose gateway is not crashed. Upon recovery, the crashed gateway is informed of the rollback, performing a rollback too. This process guarantees the consistency of both underlying DLTs.

Algorithm 1 depicts the ODAP-2PC. A coordinator $\mathcal{G}_S$ and a participant $\mathcal{G}_R$ perform a CC-Tx T, that typically is an asset transfer of x number of a assets, i.e., $T : \mathcal{G}_S \overset{a,x}{\rightarrow} \mathcal{G}_R$. Any time a party ABORTS, the protocol stops, and that transaction is considered invalid (and thus the run of the protocol fails). We define a set of gateway primitives Σ = {PRE_LOCK, UNLOCK, LOCK, COMMIT, CREATE_ASSET, COMPLETE, ROLLBACK}, such that they realize pre-locking an asset, locking an asset, unlocking an asset, committing to a CC-Tx, creating an asset, asserting for the end of the protocol, and performing a rollback, respectively. The gateway primitives are divided into two types: off-chain primitives, and on-chain primitives, represented by $\sigma^{offchain}$ and $\sigma^{onchain}$, respectively. Some off-chain primitives call their respective on-chain primitive. The protocol receives a set of gateway primitives that realize the commit, locking, rollback and other operations. Lists $PO_{\mathcal{G}_S}$ and $PO_{\mathcal{G}_R}$ track the operations to be rolledback in case of failure for $\mathcal{G}_S$or $\mathcal{G}_R$, respectively.

V. Use Case: Gateway-Supported Cross-Jurisdiction Promissory Notes

In this section, we present a use case implementing digital asset transfers, benefiting from the gateway paradigm. The digital assets to be exchanged are defined as an *asset profile*,

Algorithm 1: ODAP-2PC Protocol

Input: Coordinator $\mathcal{G}_S$, Participant $\mathcal{G}_R$, Asset a, Gateway primitives PRE_LOCK, LOCK, COMMIT,CREATE_ASSET, COMPLETE, ROLLBACK
Result: Asset a transferred from $\mathcal{G}_S$ to $\mathcal{G}_R$

1 $PO_{\mathcal{G}_S} = \perp$ ▷ rollback list for $\mathcal{G}_S$
2 $PO_{\mathcal{G}_R} = \perp$ ▷ rollback list for $\mathcal{G}_R$
3 ▷ Pre-Voting Phase
4 preLock $= \mathcal{G}_S$.PRE_LOCK(a) ▷ step 2.3
5 $PO_{\mathcal{G}_S}$.append(preLock)
6 ▷ Voting Phase
7 $\mathcal{G}_S \overset{vote-req}{\longrightarrow} \mathcal{G}_R$ ▷ step 3.1
8 **wait until** $\mathcal{G}_R \overset{\alpha(vote-req)}{\longrightarrow} \mathcal{G}_S$ ▷ step 3.2
9 ▷ Decision Phase
10 **if** $\mathcal{G}_R \overset{\alpha(vote-req)}{\longrightarrow} \mathcal{G}_S = NO$ **then**
11 $\quad \mathcal{G}_S \overset{abort()}{\longrightarrow} \mathcal{G}_R$ ▷ otherwise, $\mathcal{G}_R \overset{\alpha(vote-req)}{\longrightarrow} \mathcal{G}_S$ = YES
12 $\quad \mathcal{G}_S$.ROLLBACK($PO_{\mathcal{G}_S}$) ▷ undo $\mathcal{G}_S.preLock(a)$
13 **end if**
14 $lock = \mathcal{G}_S$.LOCK(a) ▷ step 3.3
15 $PO_{\mathcal{G}_S}$.append(lock)
16 $commit = \mathcal{G}_S$.COMMIT() ▷ step 3.4
17 **if** $commit = \perp$ **then**
18 $\quad \mathcal{G}_S \overset{abort()}{\longrightarrow} \mathcal{G}_R$
19 $\quad \mathcal{G}_S.rollback(PO_{\mathcal{G}_S})$ ▷ undo $\mathcal{G}_S$.LOCK(a)
20 **end if**
21 $\mathcal{G}_S \overset{commit}{\longrightarrow} \mathcal{G}_R$
22 $a' = \mathcal{G}_R$.CREATE_ASSET() ▷ step 3.5
23 $PO_{\mathcal{G}_R}$.append(a')
24 **wait until** $\mathcal{G}_R \overset{\alpha(commit)}{\longrightarrow} \mathcal{G}_S$ ▷ step 3.6
25 **if** $\mathcal{G}_R \overset{\alpha(commit)}{\longrightarrow} \mathcal{G}_S = COMMIT$ **then**
26 $\quad \mathcal{G}_S$.COMPLETE() ▷ step 3.8
27 **end if**
28 **else**
29 $\quad \mathcal{G}_S \overset{abort()}{\longrightarrow} \mathcal{G}_R$ ▷ otherwise, $\mathcal{G}_R$ failed the commit
30 $\quad \mathcal{G}_S$.ROLLBACK($PO_{\mathcal{G}_S}$) ▷ undo $\mathcal{G}_S$ locks
31 $\quad \mathcal{G}_R$.ROLLBACK($PO_{\mathcal{G}_R}$) ▷ undo $\mathcal{G}_R$.CREATE_ASSET()
32 **end if**
33 return ▷ asset transferred

which is ongoing work at the IETF [21]. An asset profile is "the prospectus of a regulated asset that includes information and resources describing the virtual asset". A virtual asset, on its turn, is "a digital representation of value that can be digitally traded" [21]. Asset profiles can be emitted by authorized parties, having the capability to legally represent real-world assets (e.g., real estate).

A. Asset Profile

The *Asset Profile Definitions for DLT Interoperability* draft presents an unambiguous manner of representing a digital asset, independently of its concrete implementation [21]. This is notably for tokenization, as a physical asset might be represented in a multitude of ways. Thus, it is important to find a sufficiently generic schema that allows representing an arbitrary digital asset, and thus enable asset transfers. Perhaps most importantly, its definition assures that heterogenous DLTs refer to the same asset within a transfer. An asset profile contains the following fields (from [21]): issuer, asset code, asset code type, issuance date, expiration date, verification endpoint, digital signature, prospectus link, among others. We refer to this asset profile as $\mathcal{A}_p$. For generic protocols

manipulating assets (e.g., transfer, creating), this asset profile can provide the necessary attributes for trust establishment. For instance, gateways should be able to verify its counter party identity in case of an asset transfer. Moreover, the asset profile and asset code should be identifiable and retrievable, allowing different attributes to be parsed as inputs to the asset gateway primitives.

B. Using Hermes to Exchange Promissory Notes

Promissory notes are freely transferable financial instruments where issuers denote a promise to pay another party (payee) [24]. Notes are globally standardized by several legal frameworks, providing a low-risk instrument to reclaim liquidity from debt. Notes contain information regarding the debt, such as the amount, interest rate, maturity date, and issuance place. Notes are useful because they allow parties to liquidate the debts and conduct financial transactions faster, overcoming market inefficiencies. In practice, promissory notes can be both payment and credit instruments. A promissory note typically contains all the terms about the indebtedness, such as the principal amount, credit rating, interest rate, expiry date, date of issuance, and issuer's signature. Despite their benefits, paper promissory notes are hard to track, require hand signatures and not-forgery proofs, accounting for cumbersome management. To address these challenges, recent advances in promissory notes' digitalization include FQX's eNote [10]. Blockchain-supported digital promissory notes (eNotes) worth about half a million dollars were used by a "Swiss commodity trader to finance a transatlantic metal shipment" [2]. eNotes are stored in a trusted ledger covered by the legal framework, belonging to a specific jurisdiction. Consider the following supply chain scenario: a producer (P) produces a certain amount of goods that sells to a wholesaler (W). W accepted the goods, and now P issues an invoice of value V. The wholesaler could pay in, for example, 90 days. Because P does not want to wait up to 90 days for its payment, it requests a promissory note from W, stating that V will be paid in 90 days. This way, P can sell that same promissory note to a third party. The promissory note is abstract from any physical good being exchanged. Depending on the issuer, collateral might not be needed, as the accountability for liquidating the debt is tracked by the blockchain where it is stored.

Blockchain-based promissory notes belonging to a particular jurisdiction are stored in a certified blockchain that exposes a gateway. When a promissory note needs to change jurisdictions (e.g., a promissory note issued in the USA that needs to be redeemed in Europe), the gateways belonging to the source and target blockchains perform an asset transfer, where the asset is a digital promissory note. Alternatively, the gateway extends to several jurisdictions. Below is an example of an asset profile of a digital promissory note. Such digital promissory notes can be trivially exchanged between blockchains using Hermes and the ODAP-2PC protocol, where gateways belonging to different jurisdictions (e.g., representing different blockchains regulated by different entities) perform asset transfers.

VI. Discussion

Hermes can be instantiated in blockchains supporting smart contracts that implement functionality for locking and unlocking assets. The gateway paradigm allows integrating DLT-based systems to centralized legacy systems by leveraging existing legal frameworks. For extra robustness, data integrity and counterparty performance can be attested, using trusted hardware [14], [8]. Remote attestations are particularly important, since provably exposing internal state to external parties is a crucial requirement for CC-Txs [4].

A tradeoff between reliability and performance exists. Storing logs in local storage typically has lower latency but deliver weaker integrity and availability guarantees than store them on the cloud or in a ledger. Generally, the more resilient the logging is, the higher the latency. For critical scenarios where strong accountability and traceability are needed (e.g., financial institution gateways), blockchain-based logging storage may be appropriate. Conversely, for gateways that implement interoperability between blockchains belonging to the same organization (i.e., a legal framework protects the legal entities involved), local storage might suffice.

Considering non-trusting gateways, Hermes might not be sufficiently decentralized. Besides picking the appropriate log storage support, one could choose from several techniques to decentralize gateways or to enhance the accountability level. A first option is to implement a gateway as a smart contract: this does not allow a gateway to deviate from its configured behavior but has shortcomings, such as inflexibility, lack of scalability, and operation costs. In particular, smart contracts often lack the possibility of being integrated with external resources and systems; oracles may provide some extra flexibility [6]. Smart contract-based gateways could also need to pay transaction fees in public blockchains, such as gas on Ethereum [25], raising additional costs. Additional costs imply that adding gateways on the same blockchain is not scalable. Second, to decentralize Hermes, one could implement a Byzantine fault-tolerant version of a gateway, similarly to what is planned on Cactus [17].

Regarding security, gateways should assure the integrity and non-repudiation of log entries and ensure that the protocol terminates. If an adversary performs a denial-of-service on either gateway, the asset transfer is denied but ODAP-2PC assures eventual consistency of the underlying DLTs. Accountability promoted by robust storage can diminish the impact of these attacks. The connection between gateways should always provide an authentication and authorization scheme, e.g., based on OAuth and OIDC [1], and use secure channels based on TLS/HTTPS [20].

VII. Related Work

Hardjono et al. proposed a gateway-based architecture inspired by the architecture of the Internet [13], further expanded by recent work [12]. Vo et al. propose decentralized blockchain registries that can identify and address blockchain oracles [22]. Such registries can be extended to support gateways. Hyperledger Cactus [17] is a trusted relay connecting DLTs,

whereby a consortium of Cactus Nodes endorses transactions. Cactus uses two families of software components that, on its sum, constitute a gateway: validators and connectors. Validators are components that retrieve state from blockchains, while connectors are active components that issue transactions. The consortium can run arbitrary business logic, including logic for asset transfers, making Cactus a suitable infrastructure to implement gateways. Like Cactus, HERMES is a trusted relay directed to enterprise use cases. Our system can be decentralized using one of the approaches detailed in Section VI. Other trusted relays can realize the concept of gateway (e.g., [3], [4], [16], [9]). For the sake of space, we refer readers interested in interoperability to [6].

Generally, 2PC is not used for blockchain consensus [18], but rather for communication across blockchains. Fynn et al. presented a Move operation that can migrate accounts and arbitrary computation across Ethereum virtual machine based chains [11]. An atomic Move operation can be implemented with 2PC. Wang et al. [23] presented a 2PC protocol for conducting CB-Tx. In this scheme, a blockchain is elected as the coordinator, managing the process between an arbitrary number of blockchains. This protocol includes a heartbeat monitoring mechanism to guarantee liveness. However, it is not clear how are ACID properties assured, e.g., atomicity, as the authors do not provide a rollback protocol. Our work provides ACID properties via ODAP-2PC and the rollback protocol.

VIII. CONCLUSION

This paper introduced HERMES, a middleware that enables blockchain interoperability across DLT-systems that can operate under different legal frameworks. HERMES is instantiated with ODAP, an asset transfer protocol between two gateway devices. Hermes supports ACID properties and can assure accountability by keeping an off-chain or on-chain shared log of operations. We propose and discuss ODAP-2PC, a distributed recovery mechanism, guaranteeing asset transfers between blockchains to be atomic and secure. By studying Hermes' reliability, performance, decentralization, security, and privacy, we explore the potential of gateways to respond to the current interoperability challenge. By presenting the digital promissory note use case, we show that Hermes is an appropriate trust anchor for enterprise use cases requiring cross-blockchain asset transfers. Future work will enable several gateways to be involved in an asset transfer (ODAP-3PC), paving the way for efficient multiparty atomic swaps.

ACKNOWLEDGMENTS

We warmly thank Benedikt Schuppli for discussions on digital promissory notes and our colleagues in the IETF ODAP working group for discussions on ODAP-2PC. This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and 2020.06837.BD, and by the European Commission program H2020 under grant agreement 822404 (QualiChain).

REFERENCES

[1] Final: OpenID Connect Core 1.0 incorporating errata set 1.
[2] Transatlantic Shipment of Metals Financed via FQX eNote — Treasury Management International.
[3] 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 Industrial Track*, pages 29–35. Association for Computing Machinery, 2019.
[4] E. Abebe, D. Karunamoorthy, J. Yu, Y. Hu, V. Pandit, A. Irvin, and V. Ramakrishna. Verifiable Observation of Permissioned Ledgers. *arXiv 2012.07339v2*, 2021.
[5] R. Belchior, M. Correia, and T. Hardjono. DLT Gateway Crash Recovery Mechanism draft 02. Internet-Draft draft-belchior-gateway-recovery-02, Internet Engineering Task Force, 2021.
[6] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *ACM Computing Surveys*, may 2021.
[7] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley, 1987.
[8] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen. Principles of remote attestation. *International Journal of Information Security*, 10(2):63–81, jun 2011.
[9] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov. Smart Contract Invocation Protocol (SCIP): A Protocol for the Uniform Integration of Heterogeneous Blockchain Smart Contracts. In *International Conference on Advanced Information Systems Engineering*, volume 12127 LNCS, pages 134–149, 2020.
[10] FQX. eNI™ Infrastructure - fqx.ch - Electronic Negotiable Instruments - FQX, 2020.
[11] E. Fynn, F. Pedone, and B. Alysson. Smart Contracts on the Move. In *Dependable Systems and Networks*, 2020.
[12] T. Hardjono. Blockchain Gateways, Bridges and Delegated Hash-Locks. *arXiv 2102.03933*, 2021.
[13] T. Hardjono, A. Lipton, and A. Pentland. Towards an Interoperability Architecture Blockchain Autonomous Systems. *IEEE Transactions on Engineering Management*, 67(4):1298–1309, June 2019.
[14] T. Hardjono and N. Smith. Towards an Attestation Architecture for Blockchain Networks (to appear). *World Wide Web Journal – Special Issue on Emerging Blockchain Applications and Technology*, 2021.
[15] M. Hargreaves, T. Hardjono, and R. Belchior. Open Digital Asset Protocol draft 02. Internet-Draft draft-hargreaves-odap-02, Internet Engineering Task Force, 2021.
[16] L. Kan, Y. Wei, A. Hafiz Muhammad, W. Siyuan, G. Linchao, and H. Kai. A Multiple Blockchains Architecture on Inter-Blockchain Communication. *Proceedings - 2018 IEEE 18th International Conference on Software Quality, Reliability, and Security Companion, QRS-C 2018*, pages 139–145, 2018.
[17] H. Montgomery, H. Borne-Pons, J. Hamilton, M. Bowman, P. Somogyvari, S. Fujimoto, T. Takeuchi, T. Kuhrt, and R. Belchior. Hyperledger Cactus Whitepaper, 2020.
[18] J. Nijsse and A. Litchfield. A Taxonomy of Blockchain Consensus Methods. *Cryptography*, 4(4):32, 2020.
[19] A. Pentland, A. Lipton, and T. Hardjono. Time for a new, digital Bretton Woods. *Barron's*, June 2021.
[20] E. Rescorla. RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3, 2014.
[21] A. Sardon, T. Hardjono, and Benedikt Schuppli. Asset Profile Definitions for DLT Interoperability (draft-sardon-blockchain-interop-asset-profile-00). Technical report, 2021.
[22] H. Tam Vo, Z. Wang, D. Karunamoorthy, J. Wagner, E. Abebe, and M. Mohania. Internet of Blockchains: Techniques and Challenges Ahead. In *2018 IEEE iThings/GreenCom/CPSCom/SmartData*, pages 1574–1581, 2018.
[23] X. Wang, O. T. Tawose, F. Yan, and D. Zhao. Distributed Nonblocking Commit Protocols for Many-Party Cross-Blockchain Transactions, jan 2020.
[24] J. S. Waterman. The Promissory Note as a Substitute for Money. *Minnesota Law Review*, 14:313, 1929.
[25] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. Byzantium version 7e819ec. Technical report, 2019.