Resilient Gateway-Based N-N Cross-Chain Asset Transfers

1st André Augusto
Departamento de Engenharia Informática
Instituto Superior Técnico
Portugal
andre.augusto@tecnico.ulisboa.pt

2nd Rafael Belchior
Blockdaemon Ltd.
Ireland
Departamento de Engenharia Informática
Instituto Superior Técnico & INESC-ID
Portugal
rafael.belchior@tecnico.ulisboa.pt

3rd Thomas Hardjono
MIT Connection Science & Engineering
Massachusetts Institute of Technology
Cambridge, USA
hardjono@mit.edu

4th André Vasconcelos
Departamento de Engenharia Informática
Instituto Superior Técnico & INESC-ID
Portugal
andre.vasconcelos@tecnico.ulisboa.pt

***Abstract*—New applications and solutions are emerging as blockchain technology continues to prosper in different industries. However, blockchain systems are considered isolated silos, especially when it comes to interoperability on systems putting restrictions on handling private data. We propose ODAP-AS, a resilient N-N cross-chain asset transfer protocol that enables the execution of N transfers of assets in permissioned environments, leveraging the concept of gateways. Gateways act as the devices through which a blockchain network can be accessed. We build our protocol on top of the Open Digital Asset Protocol (ODAP), and its crash recovery mechanism, ODAP-2PC, a crash fault-tolerant protocol. ODAP-AS also defines how one gateway is replaced by a backup in case of a crash. We implement a cross-chain asset transfer across Hyperledger Fabric and Hyperledger Besu using Hyperledger Cactus, which takes approximately 20 seconds. Additionally, we can conduct a sequential execution of ODAP-AS achieving 0.15 transactions/second throughput.**

***Index Terms*—Interoperability, Cross-chain, Gateway, ODAP**

I. Introduction

The concept of blockchain was popularized in 2008 as the technology behind Bitcoin, the world's largest decentralized digital currency by market capitalization [25]. Since then, blockchains have grown in popularity, opening up a new universe of possibilities, due to their unique features such as security, anonymity, decentralization, and immutability. Additionally, there have been a lot of research efforts in industries like Energy, Finance, Healthcare, and Government which makes the technology more and more adopted [2], [6], [10]. Today, there is an immense variety of public and private blockchains that have their ideas and technology stacks according to the use cases and goals they were designed to accomplish. As several studies [9], [13] point out, developers must choose between originality and stability considering blockchain implementations, which might lead to a vast amount of solutions causing the fragmentation of the blockchain industry.

Existing interoperability solutions range from the necessity of having intermediary entities that perform exchanges/transfers, such as notary schemes [13] (such as Binance), to more decentralized protocols, such as HTLCs [14], that don't require a trusted third party to ensure atomicity in a cross-chain transfer. Recently, attention has shifted to permissioned blockchains, managed by consortiums of organizations, where nodes are identified, like Hyperledger Fabric [4]. Although nodes need to be authorized and authenticated to join the network, they are not obligated to trust each other, having a governance model that relents a certain degree of trust. This permissioned concept is particularly useful to businesses, where the application logic, internal transactions, and their data might be confidential [3]. With these improvements, the connectivity of private systems might be enhanced and pave the way for new applications and use cases.

Under this scope, we find the Open Digital Asset Protocol (ODAP) [19] (currently under development in the scope of the Internet Engineering Task Force, the IETF), and its crash recovering draft, ODAP-2PC [5], which leverages the concept of gateways and describes a new protocol focused on making atomic unidirectional asset transfers between two of them (the client gateway and the server gateway). This gateway-based architecture is built upon a comparison with the concept of *Autonomous Systems* when the internet was born. At the time, the solution proposed to scale up and interconnect these networks was to implement *border gateway routers* which

would provide an entry point to each one. We can think about blockchains as networks and gateways as routers. Gateways can therefore be thought of as (semi) trusted relays between these blockchains and subsequently the parties exchanging data [13].

As it is crucial of being capable of transferring assets from party A to B, we should also focus on developing new protocols that make possible the transfer of assets to parties C and D, facilitating N-N transfers of assets. Nevertheless, solely developing new protocols is not sufficient. We, therefore, need to focus our attention on improving the existing protocols, making them reliable enough to cope with failures and still provide the required guarantees, making it possible to build our new protocols and features on top of the existing ones.

As a result, the focus of this paper is two-fold. We 1) propose an enhancement to the primary-backup mode in ODAP-2PC (increasing the resiliency of gateways); and 2) propose ODAP-AS, a new protocol on top of ODAP and ODAP-2PC, which enables the realization of atomic N-N transfers of assets in permissioned environments (versus vanilla ODAP that only allows 1-1 transfers). More specifically, ODAP-AS achieves multi-party transfers by decoupling a list of transfers into a set of 1-1 ODAP sessions.

The problem we are addressing can be translated into *"How to make N-N reliable and atomic transfers of assets in permissioned environments?"*.

Regarding the structure of this paper, section II introduces the background knowledge on Blockchain Interoperability, the Open Digital Asset Protocol (ODAP), and the respective crash recovery mechanism (ODAP-2PC). Then, in section III, we present the solution that enables us to perform N-N atomic cross-chain transfers using gateways. Following, we lay out the contribution to ODAP-2PC in section IV. Section VI and VII presents the implementation and evaluation details, respectively. The related work encompassing some protocols that perform cross-chain transfers is then specified in section VIII. We wrap up in Sections IX and X that include a plan for the near future and a conclusion of our work.

II. Background

A. Blockchain Interoperability

We can consider blockchain interoperability to be "*the ability of a source blockchain to change the state of a target blockchain (or vice-versa), enabled by cross-chain or cross-blockchain transactions, spanning across a composition of homogeneous and heterogeneous blockchain systems*" [13]. In this definition, the authors distinguish cross-chain and cross-blockchain transactions, but we use them interchangeably in this paper. Ultimately, the main goal of this area is to work on the interconnection of different blockchain systems, regardless of their inner implementation. There can be different modes of operation which is fundamental to support different use case scenarios: *asset transfers*, *asset exchanges*, and *data transfers* [17]. Asset transfers are achieved by deleting one asset on the source blockchain and its representation is created in the target blockchain. Asset exchanges focus on the transfer of ownership of one asset in each network so that the asset (or the value it represents) never leaves the relevant network. Lastly, data transfers are related to wider concepts, intending to manage and exchange data between networks rather than only assets. ODAP concerns transfers of assets, thus, in this paper we will only focus on this scenario.

B. ODAP

In the context of gateway-based blockchain interoperability, the Open Digital Asset Protocol (ODAP) [19], appears as the *"first cross-chain communication protocol handling multiple digital asset cross-border transactions by leveraging asset profiles (the schema of an asset) and the notion of gateways."* [12].

ODAP considers three access modes for clients to interact with blockchains and their resources. In this paper, our focus is on the *Relay Mode*, where a client instantiates a gateway to gateway interaction. Considering A as the client gateway, and B as the server gateway, then one ODAP session is represented by $A \overset{odap}{\rightarrow} B$. It is not the scope of the protocol the way client applications interact with each other beforehand. All communication is done through a trusted communication channel using, for example, TLS [24].

The protocol is divided into three phases/flows 1) *Transfer Initiation Flow*, where gateways exchange the communication terms and rules, making verifications regarding their identities and the asset that will be exchanged; 2) *Lock-Evidence Verification flow*, where the asset in question is locked, and a piece of evidence is presented to the other party; 3) *Commitment Establishment Flow*, in which the involved gateways effectively commit the changes and terminate the asset transfer. The commitment corresponds to the deletion of the asset in the source blockchain, and the creation of a representation in the target blockchain.

C. ODAP-2PC

ODAP lays the groundwork for communication between gateways, but it does not provide fault tolerance on its own. To address this problem, HERMES [12] proposed ODAP-2PC, a crash recovery mechanism that allows any party running ODAP to recover from a crash when exchanging messages, guaranteeing consistency across both blockchains. This mechanism is based on the logs generated before and after each sent and received message. Currently, these protocols are focused only on failures by crashing and are not concerned with Byzantine behavior, and it is proven the atomicity, consistency, durability, isolation, auditability, and termination of this protocol [12].

According to the protocol, there are two possible procedures when a crash occurs, the *self-healing* mode and

the *primary-backup* mode. In the first one, we assume the crashed gateway can recover and re-establish the communication with the other party's gateway. If the crashed gateway is not able to recover within a bounded time, a backup gateway takes control of the asset transfer. The existing specification only mentions the necessity of such procedures, not proposing any actual solution. We, therefore, explore in section IV a solution to the problem.

There are two main procedures defined by the protocol: 1) *Recovery Procedure:* where the crashed gateway sends a RECOVER message to the counterparty and retrieves the latest logs so that it is possible to resume the execution; or 2) *Rollback Procedure:* where one gateway rolls-back after a timeout from the other party. The Rollback is equivalent to issuing transactions with a contrary effect to the ones already issued [12].

III. Atomic N-N Cross-Chain Transfers using ODAP

We now present a solution to perform the execution of N-N atomic cross-chain transfers in permissioned environments. Even though there are, in academia, some solutions to the N-N problem, as of the date of writing there is no protocol that leverages gateways to achieve the same goal, with emphasis on permissioned environments. As such, the objective is making transfers of assets between N different blockchains, atomically, even in the presence of crash failures during the execution of the protocol.

A. System Model

In the current context, an N-N atomic cross-chain transfer is a set of asset transfers between N entities (represented by gateways) where atomicity is required even in the presence of crash faults. For now, we assume only one gateway per blockchain/entity and abstract the way how they were chosen within its organizations to perform the transfer with the other N-1 gateways. We assume gateways fail by crash, i.e., don't have arbitrary behavior, and $\mathcal{G}_1$ is Gateway 1, $\mathcal{G}_2$ is Gateway 2, and so on. We also leverage the concept of Virtual Asset Service Provider (VASP), which is defined as the legal entity that owns one gateway (taking advantage that in a permissioned environment nodes have a known identity) [18].

For the sake of an example, let us consider 3 VASPs: VASP1, VASP2, and VASP3, such that $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ belong to VASP1; $\mathcal{G}_5$ and $\mathcal{G}_6$ belong to VASP2; $\mathcal{G}_{10}$ belongs to VASP3 as depicted in Fig. 2).

B. Protocol Overview

The example we are studying consists of 3 gateways ($\mathcal{G}_1$, $\mathcal{G}_5$ and $\mathcal{G}_{10}$) and 3 asset transfers between them: $\mathcal{G}_1 \rightarrow \mathcal{G}_5$, $\mathcal{G}_5 \rightarrow \mathcal{G}_{10}$, and $\mathcal{G}_{10} \rightarrow \mathcal{G}_1$.

A possible first approach to the problem consists of starting 3 ODAP sessions for each transfer that will take place. In the above example we would have $\mathcal{G}_1 \xrightarrow{\text{odap}} \mathcal{G}_5$, $\mathcal{G}_5 \xrightarrow{\text{odap}} \mathcal{G}_{10}$, and $\mathcal{G}_{10} \xrightarrow{\text{odap}} \mathcal{G}_1$. These different sessions

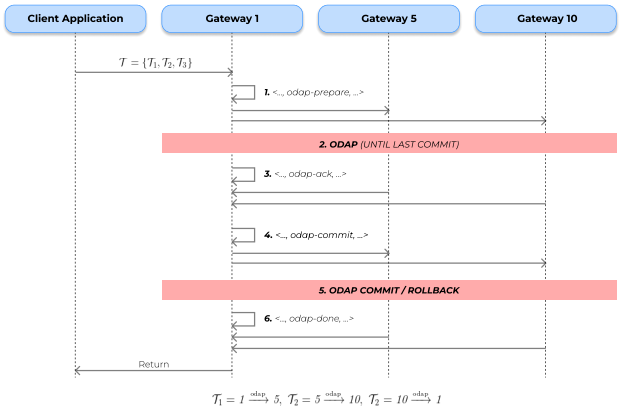

Fig. 1. ODAP-AS execution. Gateway 1 is the coordinator for the execution of multiple ODAP sessions between Gateways 1, 5, and 10.

are not coordinated with each other, thus, if there is a rollback in one of them, the others will not have knowledge of that, which would break atomicity. This forces the different processes to have some kind of coordination, like reporting back to a coordinator. In the aforementioned example, if there is a rollback, then the coordinator would be responsible for letting the other parties know that they must rollback as well. In essence, what we are trying to achieve is a coordination of the processes that assimilate to a 2PC or 3PC. The solution proposed is ODAP-AS, where the different sessions are synchronized by a 2PC initiated by a coordinator, which is one of these gateways. For now, we assume there is one gateway selected, but in the future, we can have a leader election protocol between all to decide the one with this role.

Figure 1 depicts one execution of ODAP-AS, where $\mathcal{G}_1$ acts as the coordinator of three transfers of assets between $\mathcal{G}_1$, $\mathcal{G}_5$, and $\mathcal{G}_{10}$. The problem can be formulated as $\mathcal{G}_1 \xrightarrow{\text{odap-as}} \mathcal{G}_1, \mathcal{G}_5, \mathcal{G}_{10}$.

C. Protocol Description

The coordinator gateway (Gateway 1 in Figure 1) receives from a client application a set $\mathcal{T}$ of $\mathcal{N}$ transfers $\{\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \ldots, \mathcal{T}_n\}$, where $\mathcal{T}_n$, $\forall n \in \mathcal{N}$, has the information about the client gateway, the server gateway, and the profile of the asset being exchanged.

After having all the necessary information to run the transfers, the coordinator can communicate with the client gateway for every ODAP session. This corresponds to sending a message to every A, such that $A \xrightarrow{\text{odap}} B$. This communication is done through the exchange of ODAP-AS messages, which structure is specified in III-C.

Algorithm 1 depicts ODAP-AS. The coordinator starts by sending an *odap-prepare* message to every gateway: $\mathcal{G}_1 \xrightarrow{\text{odap-prepare}} \mathcal{G}_1$, $\mathcal{G}_1 \xrightarrow{\text{odap-prepare}} \mathcal{G}_5$, $\mathcal{G}_1 \xrightarrow{\text{odap-prepare}} \mathcal{G}_{10}$ (step 1 in Figure 1). This first message includes the necessary data for each gateway to start its corresponding ODAP session with the corresponding server gateway (step 2), just as a client does in a normal ODAP 1-1 in

the *Relay Mode*. Before committing, each client gateway acknowledges the coordinator, which will gather a set of *odap-ack* messages (step 3). If every gateway responds positively, an *odap-commit* message is sent (step 4) and each ODAP session can run the last phase, where the "commits" occur — deletion and creation of the representation of the asset (step 5). If one gateway that does not respond to the first *odap-prepare* message in a defined timeout, or responds negatively, an *odap-rollback* message is sent (in detriment of the *odap-commit* message in step 4), aborting all transfers. In either condition, the protocol is finalized when each client gateway sends an *odap-done* message to the coordinator, which subsequently returns to the client application. This last step can be omitted depending on the consistency level required by the client applications, since we ensure that eventually blockchains will end up in a consistent state.

ODAP-AS Message Format

1) **Version**: ODAP-AS protocol version;
2) **Message Type**: each message has a specified format (e.g., urn:ietf:odap-as:msgtype:odap-prepare);
3) **SessionID**: unique identifier (UUIDv2) representing a multi-party session;
4) **ODAP-AS Phase**: ODAP-AS phase (prepare, ack, commit, rollback, done);
5) **Sequence Number**: increasing counter that uniquely represents a message from a session;
6) **Coordinator Gateway ID**: the public key of the coordinator;
7) **Recipient Gateway ID**: the public key of the gateway interacting with the coordinator;
8) **Asset Profile**: Profile of the asset subject to the transfer;
9) **Payload**: any necessary payload related to the Message Type;
10) **Message Hash**: the cryptographic hash of this message;
11) **Signature**: signature of this message;

Notice that this protocol is built upon ODAP and ODAP-2PC, which means that the crash recovery mechanisms are still in place. If there is a crash in one execution of ODAP before committing, then the coordinator will not send the *odap-commit* message until there is a recovery from the crashed gateway, and after sending an acknowledgment to the coordinator. In addition, in case of a permanent crash of one gateway, the others will rollback if a certain timeout is exceeded (agreed by all gateways upon starting the execution).

D. Sequential vs Parallel transfers

Dealing with N transfers of assets can be done either in parallel or sequentially. Ideally, one would run all transfers in parallel, which would represent a similar latency to executing only one (assuming there is no hardware bottleneck). However, this might not be possible, especially in cases where the transfer of an asset depends on the previous one. One could think about multiple use cases where transfers have dependencies on each other and are required to be made sequentially, while the whole set of transfers is intended to be atomic. Naturally, in this situation, the total latency would increase, as analyzed in Section VII.

Algorithm 1: ODAP-AS algorithm

```
Input: transfers_list
Result: Successfully transfer of the assets between
        the N parties
prepareResponses ← [0..N];
foreach transfer tf ∈transfers_list do
    const prepareResponse =
     tf.sender.makeODAPTransfer(tf);
    prepareResponses[tf.sender.id] ← true
wait() ;              // wait for every response
for i ← 0 to transfers_list.length() − 1 do
    if prepareResponses[i] ≠ true then
        foreach transfer tf ∈transfers_list do
            const rollbackResponse =
             tf.sender.rollbackODAPTransfer(tf);
        wait() ;      // wait for every response
        return False;
foreach transfer tf ∈transfers_list do
    const commitResponse =
     tf.sender.commitODAPTransfer(tf);
wait() ;              // wait for every response
return True;
```

IV. Gateway Replacement Procedure

HERMES leverages the concept of gateways from ODAP and proposes ODAP-2PC, a crash fault-tolerant protocol. In this protocol, it is assumed any gateway recovers from crashes within a defined bound of time, but what if it doesn't? As proposed by the authors, we need to have a primary-backup mode, where any gateway can be replaced by a backup that can resume the protocol on behalf of the first one. We propose an extension to the existing protocol, where the question we are trying to answer is *"How can a backup gateway build trust with the other party's gateway and resume the execution of the protocol?"*. We add a new assumption on top of HERMES [12] system model, where each backup gateway is up-to-date with the logs of the primary, either through the shared log storage or, for example, through a publish-subscribe system.

As in the previous section, we consider VASP1, VASP2, and VASP3, such that $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ belong to VASP1; $\mathcal{G}_5$ and $\mathcal{G}_6$ belong to VASP2; $\mathcal{G}_{10}$ belongs to VASP3 (as depicted in Fig. 2). Additionally, consider every other

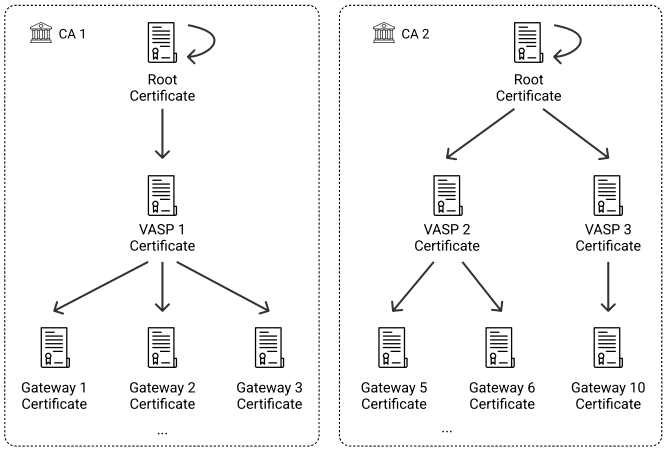

Fig. 2. Certificate Hierarchy where 2 CAs issue 3 VASP certificates, which in turn, issue gateways certificates

gateway in each VASP a backup gateway for the others. As an example, $\mathcal{G}_2$ and $\mathcal{G}_3$ are backup gateways for $\mathcal{G}_1$ open sessions, or $\mathcal{G}_1$ and $\mathcal{G}_2$ are backup gateways for $\mathcal{G}_3$ open sessions. We also assume each VASP has a certificate issued by a globally trusted Certificate Authority, CA1 or CA2.

A. Protocol Description

Let us assume $\mathcal{G}_1$ and $\mathcal{G}_5$, from VASP1 and VASP2 respectively, are running an ODAP session when $\mathcal{G}_1$ crashes. After a conservative period of time, both $\mathcal{G}_2$ and $\mathcal{G}_3$ assume the crash of $\mathcal{G}_1$, and elect one to establish a connection with $\mathcal{G}_5$ and resume the execution of the open session. If, for example, $\mathcal{G}_2$ is the chosen backup gateway, how does $\mathcal{G}_5$ know that $\mathcal{G}_2$ has the authorization to replace the first one ($\mathcal{G}_1$)? The solution proposed is based on Fig. 2 and three validations conducted by $\mathcal{G}_5$ concerning the certificates of the gateways:

1) Validate $\mathcal{G}_2$ certificate validity by running a certification path validation algorithm [15], which includes validating all the intermediate certificates up to a trusted root. In this case, $\mathcal{G}_5$ needs to recognize CA1 as a trusted certificate authority.
2) To ensure $\mathcal{G}_2$'s ability and permission to replace $\mathcal{G}_1$, $\mathcal{G}_5$ needs to verify if the parent certificate of both Gateway 1 and 2 certificates is the same. In other words, if both certificates were issued by VASP1, which proves they belong to the same entity.
3) Verify if $\mathcal{G}_2$'s certificate hash belongs to the list specified in $\mathcal{G}_1$'s certificate extensions [15], which indicates a set of gateways that are eligible to be the backup gateway in the case of a crash. This is set by each VASP when issuing a certificate.

V. Use Case using Promissory Notes

The applicability of our solution can be explained through an example. We present a simple supply chain use case that could benefit from the implementation of the protocol.

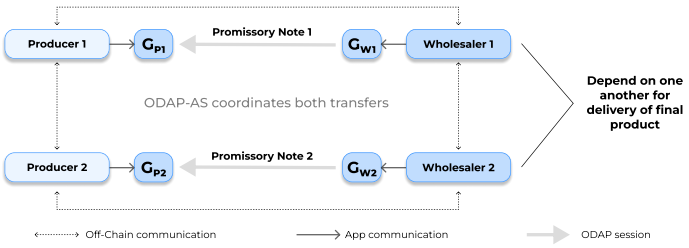

Fig. 3. Alice, Bob, and Charlie engaging in a multiparty asset transfer using ODAP-AS.

A promissory note can be defined as a promise *"made by one or more persons to another, engaging to pay a certain sum of money subject to certain requirements as to the promise"* [21]. Replacement bills or notes issued by central banks can be substituted and must be signed by the promisor [27]. Recent advances in the financial industry have focused on the digitalization of promissory notes and their integration into blockchains, given that paper promissory notes are hard to track and require hand signatures [1], [11]. Furthermore, the concept of promissory notes in the interoperability of the blockchain-based on gateways was already proposed by [11], [12].

As we have been remarking through the document, the gateway-based interoperability solutions suit the permission needs of every (or almost every) enterprise solution. Gateways are identified entities within an organization and comply with the existing regulations/legal frameworks imposed by the organization's home jurisdiction, making them suitable for this use case. As [12] points out, using these gateway-to-gateway protocols provides the building blocks for inter-jurisdiction asset transfers.

We leverage the example provided by [11] and extend it to realize an N-N atomic cross-jurisdiction asset transfer, using ODAP-AS. The base example consists of two entities, a Producer (P) that sells goods to a Wholesaler (W). P issues an invoice for value V to W, which should be paid in a maximum of 90 days. Since P might not want to wait 90 days for the payment, it can request a promissory note stating that W will pay V to P in 90 days. This promissory note can now be sold by P to a third party so as to buy any good.

Since promissory notes are valid for a given jurisdiction, if a transfer needs to take place, gateways can facilitate it while abiding by the regulations in place.

Given this base illustration retrieved from [11], we extend it to demonstrate ODAP-AS in a similar supply chain example as depicted in Figure 3. Two wholesalers – W_1 and W_2 – form a consortium that, among other products sold individually, sells products in partnership. W_1 and W_2 depend on the products sold by two producers – P_1 and P_2 – respectively.

When P_1 sells goods to W_1, P_1 issues an invoice for

value V_1, and requests a promissory note PN_1 stating the debt. The same happens between P_2 and W_2, with respect to a value V_2.

Given that W_1 and W_2 depend on one another to sell their final products, W_1 might not want to go in debt (buying and issuing a PN_1 to P_1) unless P_2 also sells the necessary amount of goods to W_2. We can therefore represent this problem as two independent asset transfers that need to be performed atomically. The problem can be formulated as a set of transfers – $W_1 \xrightarrow{PN_1} P_1$, and $W_2 \xrightarrow{PN_2} P_2$ – that must be atomic, either are both successful, or both failed.

If we consider $\mathcal{G}_{P1}$ as P_1's client gateway, $\mathcal{G}_{P2}$ as P_2's client gateway, $\mathcal{G}_{W1}$ as W_1's client gateway, and $\mathcal{G}_{W2}$ as W_2's client gateway we could leverage gateways and ODAP-AS to perform the atomic asset transfers. The problem can, therefore, be formulated as $\mathcal{G}_{P1} \xrightarrow{\text{odap-as}} \mathcal{G}_{P1}, \mathcal{G}_{P2}, \mathcal{G}_{W1}, \mathcal{G}_{W2}$, if $\mathcal{G}_{P1}$ is the coordinator.

We can conclude that ODAP-AS makes possible the execution of multi-party cross-jurisdiction asset transfers.

VI. Implementation

Besides our theoretical work, we also contribute to the open-source community by implementing ODAP, ODAP-2PC, and ODAP-AS, as business logic plugins in Hyperledger Cactus [22]. The implementation of the ODAP plugin in Cactus reaches approximately 30k lines of code and was merged into the main branch of the project.

A. Hyperledger Cactus

Cactus is a project under the Hyperledger ecosystem. It allows users to make an adaptable and secure integration of different blockchains and provides a pluggable architecture that enables the execution of ledger operations across as many blockchains as needed. One major advantage of using Cactus is that it is capable of handling the integration of both private and public blockchains, which integrates very well with the solutions proposed previously in Sections III and IV. At the date of writing, the project has nearly 1.5 million lines of code, 230 stars, and 172 forks on GitHub.

B. Architecture

We present the architecture of a single ODAP session in Figure 4 (without the green flow, and directly connecting the client app with the ODAP plugin). The client applications are the first entities involved in any transfer of assets. This represents off-chain communication (not in the actual blockchains), where both parties agree on the parameters of the transfer, which is afterward communicated to the respective gateways. In this implementation, each gateway is represented by a business logic plugin (ODAP plugin) that has multiple connections to different parties. The first one is the local database. Each gateway has in its server a local database to store the logs generated by the execution of the ODAP protocol. Each ODAP plugin

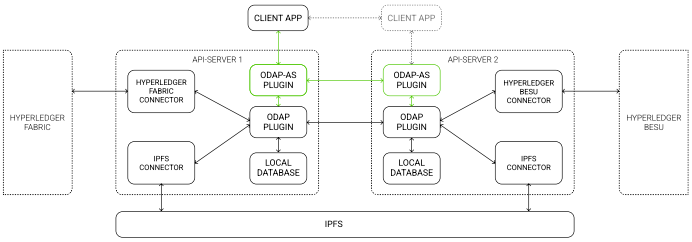

Fig. 4. Architecture of the ODAP plugin in Hyperledger Cactus with ledger connectors (Hyperledger Fabric and Hyperledger Besu), IPFS connectors, and local databases.

also has an IPFS connector to the same network, which will be used to verify the integrity of the logs. Lastly, there are the ledger connectors, which make possible the interaction with the underlying blockchains in the form of transactions (and subsequently, smart contract calls). In this implementation, each gateway is connected to a different blockchain connector, the source gateway to Hyperledger Fabric and the target gateway to Hyperledger Besu.

Now, moving forward to the N-N environment, we consider the ODAP-AS plugin as the new entry point for client application requests and the coordinator for the multi-party transfer. This new plugin communicates with every client gateway's ODAP-AS plugin. In total, N ODAP sessions will be initiated by making a direct request from ODAP-AS to the internal ODAP plugin III.

VII. Evaluation

This section evaluates the performance of ODAP-AS, but also of ODAP and ODAP-2PC, following their implementation in Hyperledger Cactus.

A. Goals

The main question we are trying to answer is whether the aforementioned protocols are indicated and provide the necessary reliability to perform atomic cross-chain transfers between N parties in a permissioned environment. Additionally, we summarize the goals and objectives of this evaluation in the form of questions:

1) **Q1:** What is the impact of having a gateway crash during one ODAP session?
2) **Q2:** What is the impact of having a rollback during the one ODAP session?
3) **Q3:** How does the performance of N-N cross-chain transfers in parallel vary with the number of transfers?
4) **Q4:** How does the performance of N-N cross-chain transfers in sequence vary with the number of transfers?

B. Testing Environment

All tests were run in a Google Cloud Compute Engine VM instance composed of 4 vCPUs, and 20 GB of memory, having a Boot Disk mounted using an Ubuntu 18.04

image, and a 30 GB SSD. As previously mentioned in Section VI, we took advantage of Hyperledger Fabric and Hyperledger Besu connectors in Hyperledger Cactus, to be the source and target blockchains, respectively. Hence, for testing purposes, we utilized the respective all-in-one Docker images available in Docker Hub.

C. Experimental Results

Q1: The first query we are trying to address is the impact of a crash in one ODAP session. To answer this question, we ran ODAP until the end of phase 2 and simulate the crash of the client gateway. We then bring the client gateway back online and resume all open sessions (only one in this case), before the defined rollback timeout. The recovery procedure is triggered and eventually, the state of both gateways ends up synchronized. The crash recovery procedure, after the crashed gateway recovers, takes only 0.50% of the total time of the protocol (not counting the time the gateway was disconnected).

Q2: Now, let us take a look into the influence of rolling back one ODAP session. In this execution, we simulate the crash of the client gateway just after the creation of the asset on the target blockchain, and we do not recover the gateway until the timeout on the server-side is exceeded (set for 5 seconds). By noticing this timeout, the server gateway initiates the rollback procedure. When the client gateway is back online and initiates the recovery procedure, it learns the rollback done by the counterparty, which will trigger its rollback. The rollback procedures from both the client and server side, in addition to the recovery procedure triggered by the client gateway, took 37.69% of the total time (this is the worst-case scenario). We can ultimately conclude that the rollback procedure is to be avoided as much as possible to the detriment of the recovery procedure. In fact, if we guarantee a backup gateway for each gateway running ODAP (as proposed in Section IV), no rollback procedure shall ever be triggered. This demonstrates the importance of having such a proposal.

Q3: As previously mentioned in Section III-D, we can consider two different constructs, one where the transfers occur in parallel, and the other where one happens after the other, sequentially. Considering N parallel transfers, the total latency of the protocol will be similar to the execution of only one, unless the available physical resources become the bottleneck by functioning at their maximum rate. We can thus conclude there is no value in evaluating N ODAP transfers in parallel since we would be testing the infrastructure instead of the proposed protocols.

Q4: We move to the sequential setting. In Figure 5 we can find the relation between the number of sequential transfers and the total latency of the protocol. As we can see from the graph, the latency is constantly rising, following a linear trend. In addition to the latency, we also measured the throughput of this execution. Since the number of transactions in each ODAP session is constant (lock, delete and create asset), and the latency of such execution is constant (given by the linear growth in Figure 5), we would expect the throughput to be also constant. The throughput is depicted in Figure 6.

We can observe, in both Figure 5 and 6, a small deviation from our predictions. We believe one of the reasons for this is the testing process, mainly being constantly writing results to local files (I/O operations), or even the change of configurations before each run (because the asset IDs need to be changed so that different assets are locked, deleted and re-created in the respective blockchains). We can thus conclude that the results of our evaluation are alike what we expected.

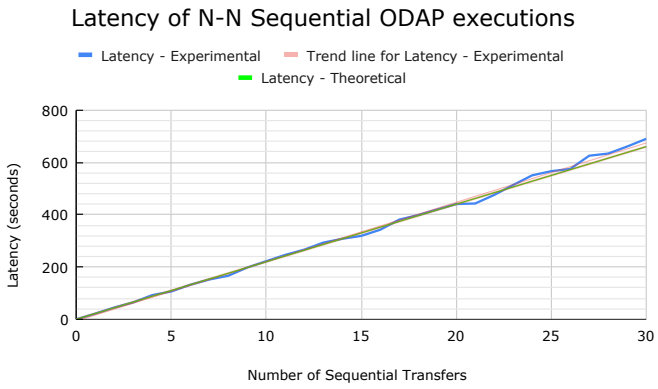

Fig. 5. ODAP-AS latency given the number of sequential transfers

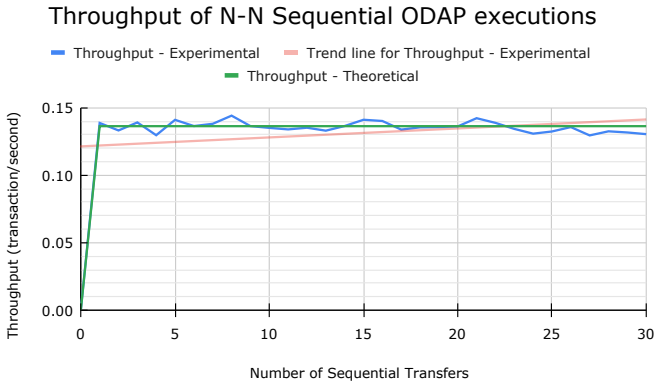

Fig. 6. ODAP-AS throughput given the number of sequential transfers

VIII. Related Work

HERMES [12] is a fault-tolerant trusted relay that connects blockchain networks, based on ODAP and ODAP-2PC. ODAP-AS, proposes enhancements to the latter, increasing its reliability, and subsequently decreasing the number of assumptions made.

Fyn et al. [16] propose Move, a protocol that enables the transfer of smart contracts between blockchains built

on top of the EVM, by leveraging 2PC, however, it is only focused on 1-1 interactions.

Luo et al. [20] suggest an inter-blockchain architecture for routing management and transfer of messages between blockchains, which is similar to our gateway architecture. However, it requires a third-party blockchain, called router blockchain. This is similar to AC^{3}WN [29], which requires a witness blockchain to enable the execution of atomic swaps between N parties.

Polkadot [28] and Cosmos, enable the interconnection of different chains (in the respective network) through the Cross-Chain Message Passing Protocol (XCMP), or the Inter-Blockchain Communication protocol (IBC), respectively. XCMP enables the interoperation with more than two heterogeneous blockchains, however, IBC can only interoperate with up to two heterogeneous blockchains. The way these systems are used as either one or the other actually does not remove the existing blockchain fragmentation. Instead of having blockchain fragmentation, we have blockchain engines (or blockchains of blockchains) fragmentation, which in fact leads to the same problem.

Wang et al. [26] also leverage 2PC to propose a new distributed commit protocol for conducting transactions across N blockchains, however, the safety and liveness properties are not yet theoretically proved. If the coordinator crashes, atomicity is only guaranteed through the assumption that eventually a new coordinator is elected. ODAP-AS guarantees atomicity through the primary-backup mode, and if needed, by triggering the rollback procedure in each ODAP session.

IX. Blockchain Views as Proofs for Blockchain Interoperability

Blockchain is being used by organizations/consortia that do not necessarily trust each other but need to share some pieces of data, including private information. As [7] points out, due to the private nature of some blockchains, there can exist different views on the state of the blockchain depending on the stakeholder (and the respective authorization).

BUNGEE [8] is a view generator that captures snapshots from ledgers to build integrated views of the global state of networks. In the gateway-based architecture, one gateway might not be allowed to look at the internal data of the other gateway's blockchain. We can, therefore, leverage the concept of blockchain views to allow one gateway to verify the state of an asset (e.g., if it is locked or not) in the other party's blockchain, removing some trust assumptions on gateways. To guarantee the truthiness of these views, they should be signed by the view generator instantiated in each gateway.

T-ODAP [23] presents a protocol on top of ODAP, which enables a trustless transfer of assets between gateways through a Decentralized View Storage, which contains the views that were published in one or more networks.

X. Conclusion

At the moment, there is no solution for the multi-party transfer problem in permissioned environments, so this paper proposes a new protocol, ODAP-AS, based on a 2PC to ensure coordination between the different entities and ODAP sessions. In addition, we propose an enhancement to the primary-backup mode in the existing crash recovery procedure, ODAP-2PC. Since we base our work on a 2PC, we might have an issue if the coordinator gateway crashes shortly before transmitting the *COMMIT* messages, however, if we consider a gateway can always recover (through a primary or backup mode), we can be confident in the protocol's finality and atomicity. We, additionally, need to define the crash recovery procedure of the proposed protocol, where it is not defined how a gateway can reestablish the connection with the coordinator of the global transfer in case of a crash. Subsequently, as a plan for the near future, we intend to develop the crash recovery mechanism for the new N-N atomic cross-chain transfers protocol, and integrate the concept of blockchain views.

References

[1] Electronic promissory notes on blockchain. https://www.gov.pl/attachment/e3ff4c9d-72f0-4ae4-89ac-f952f8ea666f, 2018. [Online].

[2] J. Abou Jaoude and R. G. Saade. Blockchain applications–usage in different domains. *IEEE Access*, 7:45360–45381, 2019.

[3] M. J. Amiri, D. Agrawal, and A. E. Abbadi. Caper: A cross-application permissioned blockchain. *Proc. VLDB Endow.*, 12(11):1385–1398, jul 2019.

[4] 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 thirteenth EuroSys conference*, pages 1–15, 2018.

[5] R. Belchior, M. Correia, A. Augusto, and T. Hardjono. draft-belchior-gateway-recovery-04 – dlt gateway crash recovery mechanism, 2022.

[6] R. Belchior, M. Correia, and A. Vasconcelos. Justicechain: Using blockchain to protect justice logs. 2019.

[7] R. Belchior, S. Guerreiro, A. Vasconcelos, and M. Correia. A survey on business process view integration. *Business Process Management Journal*, 11 2021.

[8] R. Belchior, T. Limaris, J. Pfannschmidt, A. Vasconcelos, and A. Correia. My perspective is better than yours: Blockchain interoperability with views. *to appear*, 2022.

[9] R. Belchior, L. Riley, T. Hardjono, A. Vasconcelos, and M. Correia. Do you need a distributed ledger technology interoperability solution?, 2022.

[10] R. Belchior, A. Vasconcelos, and M. Correia. Towards secure, decentralized, and automatic audits with blockchain. 2020.

[11] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono. Enabling cross-jurisdiction digital asset transfer. In *2021 IEEE International Conference on Services Computing (SCC)*, pages 431–436, 2021.

[12] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono. Hermes: Fault-tolerant middleware for blockchain interoperability. *Future Generation Computer Systems*, 2021.

[13] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia. A survey on blockchain interoperability: Past, present, and future trends. *arXiv preprint arXiv:2005.14282*, 2020.

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

[15] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.

[16] E. Fynn, A. Bessani, and F. Pedone. Smart contracts on the move, 2020.
[17] T. Hardjono, M. Hargreaves, N. Smith, and V. Ramakrishna. An interoperability architecture for dlt gateways, 2021.
[18] T. Hardjono, A. Lipton, and A. Pentland. Towards an interoperability architecture for blockchain autonomous systems. *IEEE Transactions on Engineering Management*, 67(4):1298–1309, 2019.
[19] M. Hargreaves, T. Hardjono, and R. Belchior. Open digital asset protocol draft 03, 2021. [Online].
[20] L. Kan, Y. Wei, A. Hafiz Muhammad, W. Siyuan, L. C. Gao, and H. Kai. A multiple blockchains architecture on inter-blockchain communication. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 139–145, 2018.
[21] D. M. Lobl. Promissory notes, 2013.
[22] H. Montgomery, H. Borne-Pons, J. Hamilton, M. Bowman, P. Somogyvari, S. Fujimoto, T. Takeuchi, T. Kuhrt, and R. Belchior. Hyperledger cactus whitepaper. https://github.com/hyperledger/cactus/blob/main/whitepaper/whitepaper.md, 2020. [Online].
[23] C. Pedreira, R. Belchior, M. Matos, and A. Vasconcelos. Securing Cross-Chain Asset Transfers on Permissioned Blockchains. 6 2022.
[24] E. Rescorla and T. Dierks. The transport layer security (tls) protocol version 1.3. 2018.
[25] J. Taskinsoy. Bitcoin could be the first cryptocurrency to reach a market capitalization of one trillion dollars. *Available at SSRN 3693765*, 2020.
[26] X. Wang, O. T. Tawose, F. Yan, and D. Zhao. Distributed nonblocking commit protocols for many-party cross-blockchain transactions, 2020.
[27] J. S. Waterman. The promissory note as a substitute for money. *Minn. L. Rev.*, 14:313, 1929.
[28] G. Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 21, 2016.
[29] V. Zakhary, D. Agrawal, and A. E. Abbadi. Atomic commitment across blockchains. *arXiv preprint arXiv:1905.02847*, 2019.