Hephaestus: Modelling, Analysis, and Performance Evaluation of Cross-Chain Transactions

Rafael Belchior [1,1,1], Peter Somogyvari [2], Jonas Pfannschmid [2], André Vasconcelos [2], and Miguel Correia [2]

[1]Blockdaemon; INESC-ID
[2]Affiliation not available

December 7, 2023

Abstract

Ecosystems of multiple blockchains are now a reality. Multi-chain applications and protocols are perceived as necessary to enable scalability, privacy, and composability. Despite being a promising emerging area, we have been witnessing devastating attacks on cross-chain bridges that have caused billions of dollars in losses, and no apparent solution seems to emerge from the ongoing chaos. In this paper, we present our contribution to minimizing bridge attacks, by monitoring a *cross-chain model*. In particular, we aggregate *cross-chain events* into *cross-chain transactions*, and verify if they follow a set of *cross-chain rules*, which then generate a model.

We propose **Hephaestus**, the first cross-chain model generator that captures the operational complexity of cross-chain applications. **Hephaestus** can generate cross-chain models from local transactions in different ledgers, realizing arbitrary cross-chain use cases and allowing operators to monitor their applications. Monitoring helps identify outliers and malicious behavior, which can enable programmatically stopping attacks ("a circuit breaker"), including bridge hacks.Â

We conduct a detailed evaluation of our system, where we implement a cross-chain bridge use case. Our experimental results show that **Hephaestus** can process 600 cross-chain transactions in less than 5.5 seconds in an environment with two blockchains using sublinear storage, paving the way for more resilient bridge designs.

Hephaestus:
Modelling, Analysis, and Performance Evaluation of Cross-Chain Transactions

Rafael Belchior*†, Peter Somogyvari ‡, Jonas Pfannschmidt †, André Vasconcelos*, Miguel Correia*

*INESC-ID, Instituto Superior Técnico, Universidade de Lisboa   †Blockdaemon   ‡Accenture

Abstract—**Ecosystems of multiple blockchains are now a reality. Multi-chain applications and protocols are perceived as necessary to enable scalability, privacy, and composability. Despite being a promising emerging area, we have been witnessing devastating attacks on cross-chain bridges that have caused billions of dollars in losses, and no apparent solution seems to emerge from the ongoing chaos. In this paper, we present our contribution to minimizing bridge attacks, by monitoring a *cross-chain model*. In particular, we aggregate *cross-chain events* into *cross-chain transactions*, and verify if they follow a set of *cross-chain rules*, which then generate a model.**

We propose `Hephaestus`, the first cross-chain model generator that captures the operational complexity of cross-chain applications. `Hephaestus` can generate cross-chain models from local transactions in different ledgers, realizing arbitrary cross-chain use cases and allowing operators to monitor their applications. Monitoring helps identify outliers and malicious behavior, which can enable programmatically stopping attacks ("a circuit breaker"), including bridge hacks. We conduct a detailed evaluation of our system, where we implement a cross-chain bridge use case. Our experimental results show that `Hephaestus` can process 600 cross-chain transactions in less than 5.5 seconds in an environment with two blockchains using sublinear storage, paving the way for more resilient bridge designs.

I. INTRODUCTION

Recently, many initiatives and projects have appeared around the concept of blockchain interoperability (BI), where a multi-chain ecosystem is perceived as the enabler for a scalable and adaptable platform for various use cases [1]–[4]. To enable such an ecosystem, bespoke distributed ledger technology (DLT) interoperability solutions, such as cross-chain bridges (or simply bridges), are used to connect heterogeneous DLTs, i.e., DLTs with different privacy, security, decentralization, and scalability properties [5]. The total value locked (TVL) in bridges peaked in March 2022, at around $30 billion worth of assets locked just in Ethereum (as the chain receiving the transferred assets) [6], [7], effectively reflecting the synergistic effects of free flow of capital, as now users can use their capital on multiple blockchains. As of September 2023, the TVL is still significant, collecting around 9B USD, as Figure 1 shows. With more than 40 bridging projects [8], the trend is for projects to either mature by improving their security and usability or disappear.

Some examples of recent mediatic attacks include the Wormhole bridge, where the attacker stole around $325M [9], [10], and the most significant on-chain attack in the cryptocurrencies history, the Axie Infinity's Ronin Bridge [11], which caused around $625M in losses. In February 2022, the Wormhole bridge was attacked and resulted in $320M in damage [12]. In June 2022, the Harmony bridge was hacked, resulting in $100 million in losses [13]. Although hackers were offered $1 million to return the funds to the community, it seems that they have not complied [14]. In August 2022, the Nomad bridge collateral was stolen, resulting in the loss of $200M [15], despite the bridge being developed by an expert team and audited multiple times. More recently, in July 2023, Multichain was hacked and lost around $120 million [16].

Looking at the facts, many of the largest decentralized finance hacks in blockchain history were performed in bridges [17], [18], in a grand total of more than $2.5B in damages [19], [20]. The facts show that the community still has a long way to implement secure bridges. The trend for attackers to exploit bridges will likely not disappear soon, as the more value bridges they hold, the more incentive criminals will have to attack those systems [21].

To mitigate the presented issues, we start by formalizing the interactions between different systems (which we refer to as *domains*). Cross-chain transactions (*cctx*) occur across domains and consist of a set of transactions abstracted into a logical unit of work [22], or a single atomic transaction [23]. We refer to single atomic transactions by cross-chain events (*ccevent*)). These can take place in both off-chain and on-chain systems. A *ccmodel* is the set of *cross-chain rules* that define the conditions for *cctx*s to occur - originating a state (cross-chain state). If transactions do not follow the specified rules the *ccmodel* defines, the model is *incorrect*, and thus there are several options for the analyst to proceed. Either the model is under-specified, or there is "suspicious" behavior that caused the violation of rules (e.g., malicious, such as an attack, or non-malicious, such as a software bug). Effectively, a *ccmodel* allows one to have a baseline of expected behavior to compare ongoing *cctx*s with the baseline model, following a specification-based approach to security [24].

Capturing cross-chain logic for bridges would be helpful to formalize the protocols (and help identify bugs and bottlenecks), monitor them, and act upon specific triggers. For instance, if an attack on a bridge is detected, a monitoring smart contract may pause the withdrawals, limiting the scope and impact of the attack. However, defining cross-chain logic is difficult because the base systems to be dealt with are heterogeneous and decentralized, and the systems

built on top of them (e.g., decentralized applications) may have arbitrarily complex business logic. They can comprise multiple other systems (e.g., smart contracts). In a cross-chain setting, automating the discovery of *ccmodel*s and enabling its monitoring becomes very challenging, as there need to exist more tools to secure and monitor cross-chain applications. This is where our work fills the gap in current knowledge. In summary, we present the following contributions:

- We propose `Hephaestus`, a system that creates *cc-model*s for fine-grain monitoring and auditing multiple blockchain use cases. Our system uses and extends a state-of-the-art BI solution, Hyperledger Cacti [25].
- To assess `Hephaestus`' capabilities, we provide a comprehensive evaluation of the system. In particular, we validate our contributions by generating a *ccmodel* of a bridge system that transfers tokens between heterogeneous blockchains. We tested cross-chain model generation and monitoring capabilities of `Hephaestus` according to a set of metrics (including scalability, latency, and cost) on different scenarios and workloads. After that, we present a qualitative evaluation and a discussion of the evaluation and the proposed system.
- As a technical contribution of independent interest that directly supports our contributions, we have developed and improved various Hyperledger Cacti components over several months, including blockchain connectors, several test ledgers, the RabbitMQ test server, and several Python notebooks that are available under an open-source license for the community to use.

This paper is organized as follows. Section II presents the background. In Section III we present the concept of cross-chain transaction, and then cross-chain model, in Section IV. Section V presents Hephaestus. After that, we present implementation details, in Section VI, and the experimental evaluation, in Section VII. Section VIII presents the related work and Section IX concludes the paper.

II. BACKGROUND

This section presents the background necessary to understand the paper, that is, processes, BI, and *cctx*s.

A. Process Mining Background and Applications

Understanding core concepts around processes is important to construct a system that can analyze *cctx*s and thus create *ccmodel*s. A process is a set of activities (or tasks) to fulfill a specific goal [27]. For example, behind running a proof-of-stake blockchain, we have different processes a validator needs to run to achieve the end goal, the network's maintenance process, the consensus process, and others.

The techniques for creating, analyzing, and optimizing processes are called process mining techniques [28]. Process mining has two sub-areas that help us in our endeavors: process discovery and process conformance. Process discovery aims to infer a process from an event log, that is, from a sequence of related entries, typically represented in a table. The entries in this table are *events*. An event is an occurrence that targets an activity and a point in time, related to each other

using a *case id*. Events point to an *activity* at a certain time, i.e., they have a *timestamp*. Activities are the operations that are executed within a process. Formally, an event e is a tuple $(act, caseId, timestamp, store)$, where act is the activity name, $caseId$ is the unique reference to the event, $timestamp$ refers to when the event was created, and a key-value $store$. The key-value is in the form $\{(a_1, v_1), \dots (a_n, v_m)\}$, where each a is an attribute of the event and v its value. The set of all events is $\mathcal{E}$.

The execution of a process produces what is called a *trace*, an ordered list of events with the same case id. Formally, a trace is a non-empty sequence $[e_1, \dots, e_n] \in [1, ..., n], e_i \in \mathcal{E} \wedge \forall i, j, [1, ..., n] : e_i.caseID = e_j.caseID$. An event log is a collection of traces that refer to one or more cases. Discovering a process model can be done in various ways (for a detailed overview of how to generate process models, see [29]). Process conformance checks if the incoming transactions, including their ordering (or event entries), conform (are expected) to an existing model, helping evaluate a property called replay fitness. Conformance is part of process monitoring, helping identify errors or deviations from expected behavior. Processes have different representations. Graphical representations include BPMN diagrams [30], a helpful notation for complex process semantics. In BPMN, events are denoted as circles, activities as rounded squares, and gateways as diamond squares.

B. Blockchain and Interoperability

A blockchain is a distributed protocol in which a group of nodes collectively maintains a ledger $\mathcal{L}$ of ordered transactions, possibly grouped into blocks [31]. Blockchains support two basic operations: reads and writes. Keys index information on databases; blockchains can be seen as key-value stores. A read operation obtains the value for a certain key, while a write operation on a key updates the value and returns true if successful; otherwise, it returns false. The history of each key's values is conserved by the blockchain data structure, which aggregates transactions (write requests) into cryptographically signed blocks. Reads are used to capture the part of the state relevant to interoperability processes. Reads and writes are often mediated by smart contracts, stateful, user-defined programs run by the nodes composing the blockchain network.

The blockchain properties that need to be satisfied for interoperability are *consistency* and *liveness*, widely documented in the literature [32]. Informally, consistency means that for a pair of honest nodes, at every round, the global state of one node (list of ordered transactions) is a prefix of the other node, or vice-versa. Liveness means that if an honest node receives a transaction in a certain round, it will be included in the ledger and available for all nodes.

BI is the problem of coordinating *local reads* and *local writes* such that they satisfy some cross-chain logic. That is, reads from ledger $\mathcal{L}_1$ can be composed with a write-on $\mathcal{L}_2$, realizing multiple use cases, such as data transfers, asset transfers, or asset exchanges [33]. Extensive work has been done in this area, including using two-phase commit to provide *cctx*s ACID [34] properties, where each local transaction executes successfully, or none at all [23]. We assume that there
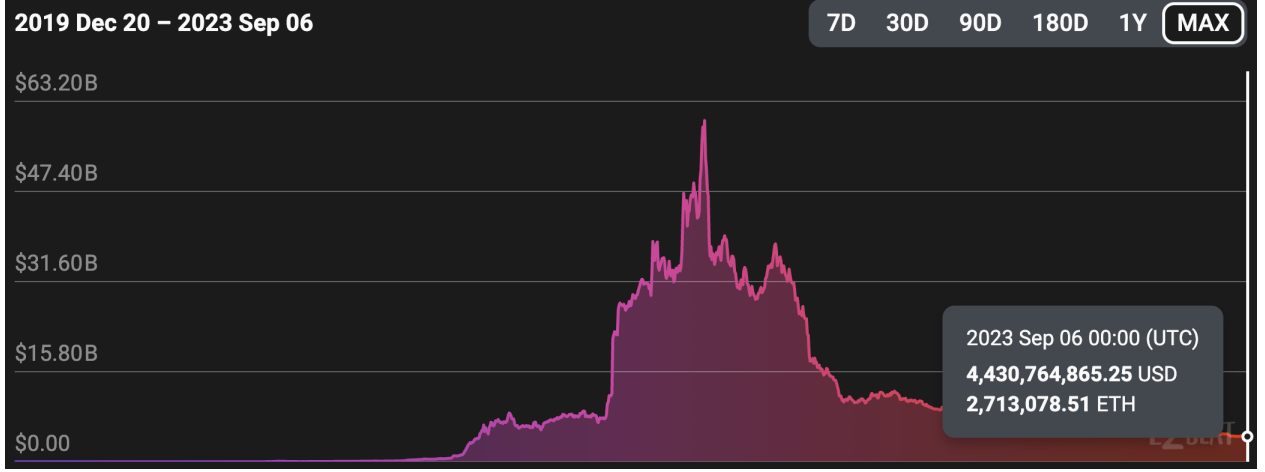
Fig. 1. Total value locked (TVL) in USD, on bridges, between 2019 Nov 15 – 2023 Sep 06. The green squares showcase relevant events in the bridging ecosystem, e.g., on 24 March 2023 (last green dot), the first ZK rollup with universal solidity support was launched. Source: [26].

is a cross-chain protocol deployed that orchestrates *cctx*s and defines the cross-chain rules that operate the use case. A *cctx* is an abstraction rooted in a set of local transactions from different systems (e.g., enterprise legacy system, centralized databases, blockchains), respecting a set of rules. Further ahead in the paper, we formally explain what these concepts are. We will map the concept of a *cctx* as a set of events (which represent local transactions) that constitute a trace over a process model (such as a ledger write). A local transaction is a transaction native to a given technological environment, called *domain*. Examples of domains are blockchains such as the Bitcoin network, the Ethereum main net, and centralized databases. Transactions trigger state changes and each state change is an event belonging to a trace. Transactions have different life cycles, data formats, and properties as a function of their domain.

Parameter	Type	Native
case ID	string	✗
receipt ID	string	✓
timestamp	Date	✓
blockchain ID	string	✗
invocation type	string	✓
method name	string	✓
parameters	string	✓
identity	string	✓
cost	number	✓
latency	number	✓

TABLE I
ccevent PARAMETERS, THEIR TYPE, AND NATURE (NATIVE ✓ OR NOT ✗).

III. CROSS-CHAIN TRANSACTIONS

In this section, we define *cctx*s and their atomic units, the *cross-chain events* (*ccevent*s).

A *ccevent* extends a local transaction with metadata. We consider this metadata to be a set of non-native attributes (or parameters) $\{a_1, a_2, ..., a_n\}$ and their values $\{v_1, v_2, ..., v_n\}$. A *ccevent* e has native attributes (e.g., an *id*, a *timestamp*, the state key to which the transaction points (*target*), a *payload* (smart contract call) that will yield a state change, and a signature from the originator), and non-native attributes, obtained via a function add, i.e., $e = \text{add}_{t^d}(a, data)$, where add is a function that adds data item $data$ to an attribute a of a local transaction t from ledger l. Each $data$ item is a non-native parameter (marked with ✗ in Table I). The native parameters can be obtained from the underlying domains or systems, i.e., retrieved from the nodes supporting the blockchains without post-processing. Non-native parameters are externally obtained and are used to enrich local transactions. Native parameters may be used to calculate non-native parameters. For example, the carbon footprint depends on native parameters (e.g., on the native parameter cost (gas), in Ethereum).

A *cctx* is a set of n ordered events $\mathcal{E}$ from a subset of domains (e.g., ledgers) $\{d^1, ..., d^n\} \in \mathcal{D}$, i.e., $\mathcal{E} = \{e_1^{d^1 \in \mathcal{D}}, ..., e_k^{d^k \in \mathcal{D}}\}$, where k represents the number of events contained in a *cctx*. The events may follow a set of rules $\mathcal{R}$, the entity that logically connects events. Rules define conditions that must be verified to each event within a *cctx*; they depict the dependencies of each event on, for example, global time, local state, and third-party domain state. A rule is a datalog rule [35], [36]. A datalog rule contains a head $R_\mathcal{E}$ and a body, and is defined recursively. Given a set of predicates $\zeta = \{\zeta_1, \zeta_2, ..., \zeta_n\}$ over a set of events $\mathcal{E}$, we have that, for a certain time interval t_δ a rule is given in the form $R_{\mathcal{E}, t_\delta} \longleftarrow \zeta(\mathcal{E})$ (we omit t_δ for simplicity of representation). The event set satisfying $R_\mathcal{E}$ are the intersection $\{\mathcal{E}|\zeta_1(\mathcal{E}) \wedge \zeta_2(\mathcal{E}) \wedge ... \wedge \zeta_n(\mathcal{E})\}$, this is, for an event set to satisfy a rule, it needs to satisfy all predicates. Each predicate ζ can define the conditions over transactions, i.e., temporal dependencies, the domain of a transaction, or a target function.

For example, consider the following rule (predicate set):

$$\zeta(\mathcal{E}) = \begin{cases} \zeta_1(e) = e^x \prec e^y & \text{order dependency} \\ \zeta_2(e) = \forall e : e^x \vee e^y & \text{included domains} \\ \zeta_3(e) = \exists e : e.cost < z & \text{event attributes} \\ \zeta_4(e) = e_w.target = e_{w+v}.target & \text{event payload} \end{cases} \quad (1)$$

In this predicate set, ζ_1 defines any event that occurs in the domain d_x precedes ($\prec$) any event happening in the domain d_y. The predicate ζ_2 defines events as part of domains x or y. Predicate ζ_3 states that there is at least one event in the event set, so its cost is less than z. Predicate ζ_4 states that the target of a transaction repeats every v transactions. Other predicates can be set for any of the attributes of a *ccevent*, in Table I. While we require each event to satisfy each sub-predicate of ζ, we can also set the validity of rules as the union $\{\mathcal{E}|\zeta_1(\mathcal{E}) \vee \zeta_2(\mathcal{E}) \vee ... \vee \zeta_n(\mathcal{E})\}$, or any other combination of predicates. We assume that there is an efficient way to transform a set of conjunction predicates into disjunctions or other formats. To capture this variability, we define a function verifySatisfability that takes a predicate and an event and outputs true if the event satisfies the given predicate and false otherwise, i.e., verifySatisfability(e, ζ) $\longrightarrow \{0, 1\}$. We can then use this predicate for each event to assert a rule's validity.

To understand how this concept applies in practice, consider the following (simplified) rule that dictates the necessary events for a valid cross-chain asset transfer:

$$\zeta'(\mathcal{E}) = \begin{cases} \zeta_1'(e) = \underbrace{(\forall_e(e^x \vee e^y)}_{\text{events happen in x or y}} \wedge \underbrace{\forall_{e \in (x,y)}(e^x \prec e^y)}_{\text{events on x happen before y}} \\ \zeta_2'(e) = \underbrace{e^x.target = exists(a) \wedge e^x.target = lock(a)}_{\text{asset can only be locked if exists}} \\ \zeta_3'(e) = \underbrace{\zeta_2'}_{\zeta_2 \text{' is satisfied}} \wedge \underbrace{e^y.target = mint(a)}_{\text{a mint can occur in domain y}} \end{cases}$$

$$(2)$$

Let us define rule ζ, the disjunction of the ζ' predicate set. The predicate set ζ' defines a set of conditions for a cross-chain asset transfer to be valid. First, as determined by ζ_1', events in domain x must happen before events in the domain y. This paves the way for a lock on a source blockchain to be done before a mint on a target blockchain. Predicate ζ_2' states that an asset from the source blockchain must exist before it is locked. Predicate ζ_3' states that before an asset is minted on the target blockchain, predicate ζ_2' must be satisfied. One could add more rules, such as the time for a mint transaction has to be done before block b, i.e., $e.target = mint \wedge e.timestamp \prec b$. We illustrate a cross-chain use case that allows asset transfers, in finer detail, in Section VII-B.

IV. CROSS-CHAIN MODEL

This section defines *ccmodel* and its artifact, *ccstate*. A *ccmodel* $\mathcal{M}$ is a tuple $(\mathcal{R}, cctx)$, where $\mathcal{R}$ is a set of cross-chain rules, $cctx$ is a set of *cctx*s. The *cctxs* originate a cross-chain state $\mathcal{S}$.

A. Properties

Cross-chain models have a set of properties:

- Verifiable correctness (safety property): a model is *valid* if all *ccevent* e in each *cctx* respects the set of rules $\mathcal{R}$, i.e., $\forall cctx \in \mathcal{M} : \forall(e \in cctx) :$ verifySatisfability($e, \mathcal{R}$) $\longrightarrow 1$.
- Liveness: the current cross-chain state $\mathcal{S}$ is updated no later than every t timesteps. Updating the *ccstate* implies checking the existing *cctx*s against the model rules.
- Probabilistic completeness: the larger the event log (i.e., the number of observed events and consequently *cctx*s), the higher the model completeness probability.
- Replay fitness: given an observation of the real-world use case, the matching between the events and the *ccmodel* is higher than a threshold probability p.

Cross-chain models are correct if each *ccevent* follows each rule, as suggested by [33]. Note that the verifySatisfability predicate can be defined in several ways (e.g., a conjunction of rules, disjunction of rules, or a bespoke combination). We say a model is *incorrect* if there are events that do not satisfy the rules of the model. For example, the execution of *ccevent*s in an incorrect order as specified in the rules causes a model to be incorrect. In this case, the model is not secure. As *ccmodel*s capture security by evaluating a set of predicates on events and rules, we can capture different safety properties, such as atomicity, double-spend protection, and others commonly debated in the interoperability literature [1], [33], [37], [38].

Updating a model is, in practice, polling for new cross-chain events and matching them with the rules defined by the model. To this end, and as each domain has its own clock, measuring and tracking time across systems is important. Liveness states that models need to be updated every t timesteps - the time between updates is an attacker's time window. If a model cannot guarantee liveness within t timesteps, we say that the model is outdated (and thus security is not guaranteed). We will show that liveness is particularly important to detect cross-chain attacks.

Completeness is related to precision. A precise model avoids underfitting, a degree of measurement of how complete is the *ccmodel*. Replay fitness expresses the ability to explain on-chain behavior, i.e., how close the *ccmodel* is to reality. Other properties that are interesting in our context matter, but will be explored in future work. These generalizations measure whether the model is too tied to specific execution instances of a cross-chain use case. Simplicity measures whether the model is understandable by humans. Other aspects are omitted from the model generation, such as the task of minimizing noise, that is, minimizing behavior that is infrequent and does not represent the typical behavior of the process.

B. Cross-Chain State

The cross-chain state $\mathcal{S}$ is a key-value store that holds attributes relevant to the cross-chain use case, i.e., they are defined on a case-by-case basis. It is generated from the *cctxs* from the *ccmodel* and it is similar to the world state concept in Hyperledger Fabric. Essentially, the state contains the result

of executing all the *cctx*s, possibly enriched with metadata. For example, if the use case is a bridge, the state will record the assets locked in the source chain and the corresponding representations minted on the target chain, for each user, and some key metrics (see Section VI-C). Metrics are performance attributes of a set of *cctx*s [39] and provide meta-information about a cross-chain use case. These metrics indicate points of interest in a cross-chain use case. Metrics realize a meta state, where metrics about the formation and execution events that lead to that state are created.

Latency: We define latency as the time between a local transaction (via extended clients) and the creation of an *ccevent*. The total latency of a *cctx* ($\delta(cctx)$) is given by the latency of each event $\delta(e)$ from each local transaction, summed to the operational latency of the *ccmodel* generator ($\delta(op)$):

$$\delta(cctx) = \sum_{\substack{i=1,\dots,n \\ j=1,\dots,k}} \delta(e_i^{d_j}) + \delta(op)$$
$$\forall d\, e \in cctx \quad (3)$$

The operational latency is the time the model generator takes to retrieve and process the local transactions.

The latency of a *ccmodel* is the sum of the latency of each *cctx*:

$$\delta(\mathcal{M}) = \sum_{i=1,\dots,n} \delta(cctx_i) \quad (4)$$

Throughput: The throughput of a *cctx* is defined as $\frac{1}{\delta(cctx)}$, and it counts the number of sets of events processed per unit of time. Effectively, the latency for each event compresses the issuance and processing of each local transaction (which can take a long time depending on the blockchain), plus operational costs. The slowest finalization time δ_{fmax} can be a valuable metric to complement throughput (and help identify bottlenecks in a *cctx*).

$$\delta_{fmax} = \max_{e_i \in \mathcal{E}, d \in \mathcal{D}}(\delta(e_i^d)) \quad (5)$$

Cost and Revenue: Each local transaction might have a cost of transaction fees plus operation fees (in case a relayer or entity is transporting the local transaction payload across chains). Inspired by [39], we define the cost c of a *cctx* events as the sum of variable costs (c_δ) plus operational costs (c_{op}):

$$c(cctx) = \sum_{\substack{i=1,\dots,n \\ j=1,\dots,k}} cost_\delta(e_i^{d_j}) + c_{op}$$
$$\forall d\, e \in cctx \quad (6)$$

The environment (e.g., via our system) typically gives information about these costs. The revenue is calculated in a way similar to the above formula. We can then calculate the profit of each *cctx* by subtracting the costs from the revenue. The concept of revenue can be modeled as positive utility and cost as negative utility, which sometimes maps better to real-world applications.

V. HEPHAESTUS: A CROSS-CHAIN MODEL GENERATOR

Hephaestus[1] is a software system that generates *cc-model*s. It first captures local transactions (*ccevents*) and then generates *cctx*s. Those *cctx*s assist in generating the *ccmodel*, along with a process discovery algorithm. Derived from the *cctx*s we have the *ccstate*, which holds metrics of interest, and a key-value store that represents the system variables with regard to bespoke business logic.

The *ccmodel* holds a set of rules (as defined in Section III), and dictates which *cctx*s should be issued. In particular, rules define the order and dependencies of each *ccevent* that later form a *cctx*. Such models can be specified or learned from the environment. After that, such a model will continuously update its cross-chain state in real-time during the monitoring phase (Section V-D). This framework allows us to answer several questions: *What is the state of our cross-chain use case/protocol? Are there unexpected behaviors, i.e., deviations from the model? What are the current bottlenecks of a given cross-chain use case?* and *Is there suspicious behavior concerning my use case, for example, an attack?*

A. System Model

Cross-chain applications are structured as multi-step protocols with different types of agents. Agents are users (e.g., end-users, relayers [2], or protocol administrators) and smart contracts. Users take turns doing off-chain processing and interacting with one or more domains (e.g., submit transactions against smart contracts). Agents are considered Byzantine, i.e., they can attempt to deviate from a protocol. Smart contracts enforce cross-chain logic. Specifically, smart contracts running on different blockchains are trusted replicas in the state-machine replication literature (similarly, each domain is considered trusted, even if centralized). This assumption implies that if a domain cannot be trusted, then safety on cross-chain rule execution cannot be guaranteed. Domains can only learn the state of other domains and their changes using an agent. Of course, each domain must decide whether an agent is telling the truth. This can be achieved with a cross-chain protocol, where trust assumptions vary substantially [1]. Furthermore, we assume a partial synchrony model, i.e., there is some finite unknown upper bound t on the responses (e.g., event creation) from the underlying domains. Hephaestus runs a global clock, i.e., it can measure time against different domains, despite their clocks being different.

Our model assumes a cross-chain protocol that can provide a set of rules such that the set of rules is enforced by on-chain (e.g., smart contracts) and off-chain components (e.g., relayers). In particular, for us to generate a correct *ccmodel*, the execution of cross-chain transactions needs to result in a consistent cross-chain state for a certain set of rules. In our bridge example, the rules define that no double spending occurs. This is, it is impossible to mint an asset on a target blockchain without first locking it on the source blockchain; similarly, it is not possible to unlock such an asset on the source blockchain without first burning it on the target

---

[1]Hephaestus is the Greek god of metallurgy (that can connect different chains into a single useful artifact.)
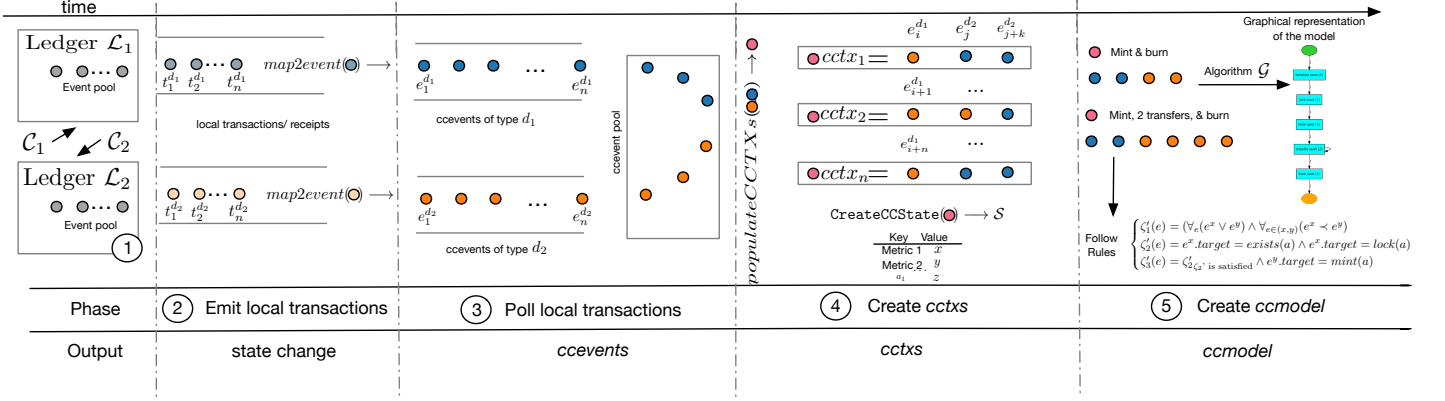
Fig. 2. Cross-chain model pipeline, spanning from phases 2 to 5.

blockchain. We will elaborate on this use case in Section VII. Liveness ensures that all cross-chain rules from a protocol are evaluated (executed) timely. In our example, liveness means that "conforming parties' assets cannot be locked up forever".

B. System Overview

After defining our domain scope (step ①), a set of modified blockchain clients, called *connectors* issue transactions against target blockchains via an application (step ②). These blockchains emit events (or transaction receipts) that our connectors capture. Hephaestus then collects the local transactions (also called transaction receipts) in step ③ and generates *ccevent*s enriched with metadata. After that, it generates a set of *cctx*s (④) and next, it generates a *ccmodel* (⑤). The next section will illustrate the steps in finer detail. Finally, the monitoring phase occurs (Section V-D), where events are constantly monitored and used to update the *ccstate*. Business logic can be defined to facilitate the integration with legacy systems or to implement audit or monitoring functionality.

C. Cross-Chain Model Generation

This section explains how to generate a model, focusing on phases ② to ⑤. Our pipeline is divided into phases (cf. Figure 2): ② *Emit Local Transactions*, ③ *Poll Local Transactions*, ④ *Create cctxs*, and ⑤ *Create ccmodel*. The monitoring phase is illustrated in the next section.

In phase ②, we start by listening to local transactions in our domain set $\mathcal{D}$. Each domain in our system has an accessible event pool from which we can fetch the events used to build the model. Without loss of generality, and to simplify our reasoning, we look for local transactions in the domains ledger $\mathcal{L}_1$ (◯), and ledger $\mathcal{L}_2$ (◯). The considered transactions are created and submitted by our connectors, $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. Transactions have a case id, meaning local transactions without this special identifier are not considered. Our clients capture the relevant transactions and send them to the *ccmodel* generator, starting the next phase. In phase ③, the raw transaction receipts enter a processor module that translates local transactions into a *ccevent* according to a function $map2event(tx_{\mathcal{L}}) \longrightarrow ccevent$.

The output of phase ③ are *ccevent*s coming from $\mathcal{L}_1$, ● $event_1$, and from ledger $\mathcal{L}_2$, ● $event_2$ which we aggregate onto a *ccevent* log, with events coming from different ledgers. Therefore, events implement a standardized data model. After constructing a set of events, we proceed to phase ④. In this phase, we receive an event log and output the cross-chain state and a set of *cctx*s (via a *createCCTXs* function).

Algorithm 1: Cross-Chain State Update.
Creation of a cross-chain state from a set of *ccevent*s

Input: Set of events $\mathcal{E}$
Input: State update algorithm createCCState
Input: Cross-chain rules $\mathcal{R}$
Input: Cross-chain state $\mathcal{S}$
Output: Upon success returns cross-chain state $\mathcal{S}$, and a *SYNC MOVE* ●

1 **require** verifySatisfability$(e, \mathcal{R}, \mathcal{S})$ // Returns tuple (event, *MOVE ON LOG* ●) if event do not conform to the rules, cross-chain state is invalid.
2 **foreach** $e \in \mathcal{E}$ **do**
3   // For each event in retrieved event set
4   **if** $\nexists \mathcal{S}[e.caseID]$ **then**
5   // each cross-chain state key is indexed by case ID.
6          $cc = $populateCCTX$(\mathcal{S}[e.caseID], e)$
7   **end if**
8   **else**
9          $cc = $updateCCTx$(\mathcal{S}[e.caseID], e)$
10   **end if**
11      $\mathcal{S} = \mathcal{S} \cup cc$
12      $\mathcal{S}' = $createCCState$(\mathcal{S}, e.caseID)$
         // Calculates the updated *ccstate*, algorithm is parametrizable
13 **end foreach**
14 **return** $(\mathcal{S}', $ SYNC MOVE ●$)$

Algorithm 1 illustrates step ④. It receives the processed *ccevent*s from step ③, a state update algorithm createCCState, a collection of rules (as specified in the use case), and the previous *ccstate* (could be empty). Note that the *ccevent*s follow the data model specified in Section III. Algorithm 1 is responsible for monitoring that the events follow the rules, and is executed from time to time. Further-

more, it updates the cross-chain state (composed of metrics and a key-value store).

First, we collect every event retrieved dynamically by the connectors. A "require" check on line 1 verifies if the set of *ccevent*s respects the rules (see the defined rules for the bridge use case in Section III). Note that the rules evaluation might consider the current *ccstate*, which is why it is provided to the `verifiySatisfiability` primitive. If the check returns false, the algorithm returns an error and is handled by the incident management component. We will describe more details in the next subsection.

Otherwise, we iterate on the events from the set and aggregate them into *cctx*s (lines 3-10). Note that *cctx*s are indexed by caseID. Here, we create the metadata fields, namely the metrics (latency, cost, throughput, in line 5) or update them (in line 8) depending on if a *cctx* with identifier "caseID" already exists. The `populateCCTXs` function assigns the different metadata from *ccevent*s and attributes (e.g., payload) to the newly created *cctx*. The `updateCCTx` updates a *cctx* based on the new information a *ccevent* carries. For example, if the *ccevent* carries a cost, the new cost of the *cctx* will have its cost incremented by *e.metadata.cost*. The new or updated *cctx* is added or updated to our current *ccstate*. Now, we need to update specific cross-chain semantics with the function `createCCState`. We provide an example for `createCCState`, Algorithm 2. This algorithm verifies if the current event locks an asset on a source blockchain so it can be minted on a target blockchain. It sets a bit to one for each locked asset. Note that this function is illustrative and does not reflect a complex lock-unlock mechanism. For instance, the algorithm should check if an unlock with a newer timestamp happened regarding a locked asset and perform user management.

Having a cross-chain state, we initiate the *ccmodel* generation phase (5). In this phase, we generate a *ccmodel* using an algorithm $\mathcal{G}$ and the *ccstate*. Several graphical representations are possible, such as a process tree or a BPMN model. We provide details on this process in Section VI-D.

Algorithm 2: State update algorithm processCCState - Verification of a lock transaction referring to asset a

Input: Cross-chain state $\mathcal{S}$
Input: Case id id referring to the lock
Output: Updated cross-chain state $\mathcal{S}$

```
1 foreach cctx ∈ S do
2     if cctx.caseID = id then
3         // current event?
4         if cctx.target == verifyLock ∧ e.target == a
           then
5             // if current event specifies a lock
6             S.lockedAssets[a] = 1  // then set asset a
                                        to locked
7         end if
8     end if
9 end foreach
10 return S
```

D. Cross-Chain Transaction Monitoring

In this section, we explain how we monitor transactions and detect non-conformance behavior, alleviating on-chain bridge hacks. Non-conformance behavior can be one of three: outliers, malicious intent (bug exploitation/attack), or non-modeled behavior. The baseline for detecting non-conformance is a *ccmodel*, which corresponds to a specification of expected behavior. We define a set of traces belonging to the set of all possible traces $\{t_1, ..., t_n\} \in \mathcal{T}$ as a current execution of a cross-chain use case. For each trace being executed, we consider a set of steps $s_1, ..., s_n$. We then take the sequence of steps and perform alignment. Alignment-based replay aims to find one of the best alignment between the trace and the model. Each alignment creates a set of pairs (trace, transition) such that for each pair, one of the following can occur: 1) *SYNC MOVE* 🟢 - both the trace and the model advance in the same way during the replay, meaning we have a match, 2) *MOVE ON LOG* 🔴 - the trace that is not mimicked in the model. This means there is a deviation between our specification and the observed behavior.

The idea is now to retrieve incoming *ccevent*s at every t timesteps and build a trace. The current trace is constructed and encoded in the *ccstate*. For every incoming event, the next step of the trace is calculated and compared against the model (i.e., compared against the set rule $\mathcal{R}$). If a trace is *SYNC MOVE* 🟢,i.e., `verifySatisfability` returns true, then it is common behaviour (expected). The state is updated and returned. Otherwise, it is non-modeled behavior, and *MOVE ON LOG* 🔴, along with the point on the trace (current event) that originated the error. The lesser t, the better liveness a model provides and, consequently, the smaller the attack window. This error triggers an incident response framework. The framework defines how the incident should be investigated as it can result from under-modeling or malicious behavior (for example, an attack). The definition of an incident response framework for briges is out of scope and is left for future work. In any case, the end-user may inspect the event leading to the trace and understand which parameter has caused such behavior. Malicious behavior can come in different forms. The most common are smart contract vulnerabilities holding the business logic that realizes the use case. Many more attack vectors exist, such as smart contract framework vulnerability, dependency vulnerability, cryptographic vulnerability, network attacks such as denial of service or network partitioning, consensus manipulation, and others [40].

Upon detecting malicious activity, an incident response plan can be put into practice and halt the bridge until a patch is deployed, according to good practices [7], [41], [42]. The most prevalent attack in bridges is a cross-chain double spend [33], [43]: where the lock-unlock mechanism of such bridges is bypassed. In this paper, we focus on this type of attack, executed as a smart contract exploitation further elaborated in Section VII-D.

VI. Implementation

In this section, we present the implementation. The code is available on Github[2]. We developed our work as a Hyperledger Cacti (Cacti) [25] plugin. Cacti is a blockchain integration project supported by enterprises such as Blockdaemon, Accenture, IBM, and Fujitsu, with more than 270 stars and 80 contributors. Next, we detail this paper's relevant technical contributions and the implementation of `Hephaestus`.

A. Connectors

We implemented two blockchain connectors to connect to multiple blockchains and retrieve transactions. Connectors are self-contained application programming interfaces that constitute the basis for interoperability functionality. The first connector binds our software to Hyperledger Fabric 2.2 — a permissioned blockchain system. Fabric is designed for enterprise-grade applications that benefit from decentralization. It supports smart contracts (called chaincode), that can be written in several general-purpose programming languages. The nodes execute proposals for transactions signed and sent to an orderer node. Orderer nodes reach consensus on the order of transactions, batch them into blocks, and link them, creating the blockchain. Then, new blocks are sent to the nodes on the network. Fabric has a key-value store that holds the most up-to-date values from the blockchain - a desirable programming model to implement a bridge; it allows chaincodes to retrieve state without reconstructing the blockchain. We implemented this connector, package name *cactus-plugin-ledger-connector-fabric* in Typescript, counting $\approx$5k lines of code. We wrote 16 integration tests, accounting for $\approx$4k lines of code. The connector supports functionality to issue transactions (*transact*), deploy smart contracts (*deployContract*), send transaction receipts to `Hephaestus`, and several administrative tasks (such as registering a new user).

The second connector connects to a Hyperledger Besu (Besu) 1.5.1 network. Besu is an open-source Ethereum client, that also has capabilities to span private EVM-runtime compatible networks. It allows for interacting with Ethereum networks, including participating in the consensus process, developing and deploying smart contracts and decentralized applications. Besu implements proof of authority algorithms such as IBFT (more suitable for private networks) and proof of work (Ethash). We implemented this connector, package name *cactus-plugin-ledger-connector-besu* in Typescript, counting $\approx$5k lines of code. We wrote 14 integration tests, accounting for $\approx$3k lines of code. The connector supports functionality to issue transactions (*transact*), deploy smart contracts (*deployContract*), send transaction receipts to `Hephaestus`, and several administrative tasks (such as obtaining a raw block from the network).

B. Test Ledgers

We implemented tools to programmatically create test networks for Fabric and Besu, allowing for reproducible tests

[2]https://rafaelapb.page.link/code

Concept	Implementation	Lines of code
Domain	Fabric ($\mathcal{L}_1$), Besu ($\mathcal{L}_2$)	–
Domain logic	Bridge smart contracts	819
Ledger client	Fabric ($\mathcal{C}_{\mathcal{L}1}$), Besu ($\mathcal{C}_{\mathcal{L}2}$)	17k
Test ledger	Fabric and Besu test ledgers	1.8k
Model Generator	`Hephaestus`	5.9k
Process Discovery	pm4py	–
Process Conformance	pm4py	–

TABLE II

IMPLEMENTATION EFFORT AS THE NUMBER OF LINES OF CODE CREATED, FOR EACH PRESENTED COMPONENT

and debugging of our application, namely the tools *besu-all-in-one* and *fabric-all-in-one*. These tools not only allow the reproducibility of our work but also ease the developers to create new applications and build on top of `Hephaestus`. The all-in-one test ledgers are divided into two parts: 1) a test ledger manager, a Typescript program that launches, administrates, stops, and destroys test ledgers by binding to a process running a Docker container, and 2) Dockerfiles defining the networks. The Fabric test ledger manager has $\approx$1k lines of code. The Besu test ledger manager has 428 lines of code. Other test ledgers such as *corda-all-in-one* and *substrate-all-in-one* are available for the research community.
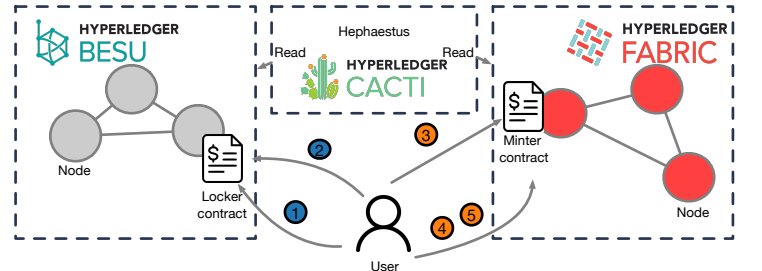
C. Bridge and Smart Contracts

Fig. 3. Cross-chain bridge across Hyperledger Fabric and Hyperledger Besu

We implement a use case of asset transfers across a Hyperledger Besu network and a Hyperledger Fabric network as a foundation for testing `Hephaestus` capabilities. A cross-chain asset transfer triggered by an end-user generates a set of events representing an asset lock on a source blockchain (Besu) and an asset unlock on a target blockchain (Fabric). This lock-unlock mechanism assures that the representation of a minted asset is pegged to a locked asset. In asset transfers, there is typically a third-party actor called a relayer, which carries the proof of a lock to the target blockchain, so the mint can occur[3]. Alternatively, the user can provide proof. Without loss of generality, we use the later representation.

The use case is implemented as follows: on the Besu network, we have a Solidity smart contract "Locker" with two methods: *create asset*, and *lock asset*. First, a user must

[3]We could model the relayers behavior in our use case, by adding two activities: *submit proof*, which contains a payload that certifies that an asset was locked from the source chain, and *proof submission*, a payload consisting of a proof that validates the minting of an asset. These events would be added by a system other than the mock blockchain (e.g., relayer).

create an asset (step 1) and then lock it (step 2). On the Fabric network, the Typescript smart contract "Minter" allows a user to call the method *mint asset* (step 3), creating the representation of the locked asset. Afterward, a user can freely transfer that token to other users in exchange for other tokens, using *transfer asset* (optional, step 4). Finally, if users want to recover the original tokens, they run *burn asset representation* (step 5). This procedure would unlock the assets on the source chain (omitted for brevity).

D. Hephaestus Plugin

We implemented `Hephaestus` as a business logic plugin for Hyperledger Cacti, written in Typescript. Its latest version is version commit *8d8567e* (*stable branch*), package name *cactus-plugin-cc-tx-visualization*. The main class is *CcTxVisualization*, which takes as input references different blockchain connectors. The plugin can be run as a web service, inspecting the event pools. After a local transaction is detected, the plugin adds them to a temporary queue. Transactions are transformed into events. From time to time, events are transformed into *cctx*s (batched for efficiency). The data model for events and metrics can be defined by the developer.

`Hephaestus` uses the *cctx*s to are used to build a cross-chain state, which is made available to the applicational layer. Models are built from *ccevent*s, given as input to a Python script (model generator) that, on its end, generates the *ccmodel*. The model generator used the open-source library pm4py version 2.2.20 [44]. We generate our model using the Inductive Miner algorithm [45], and generate the corresponding BPMN and process tree diagrams. Our plugin counts ≈5.5k lines of code. To identify misconformance, we used an alignment technique, available in the *conformance_diagnostics_alignments* function from the *pm4py* library, namely the Scipy linear solver tool.

VII. EVALUATION

Goals: The goals of the experiments are as follows. 1) evaluate `Hephaestus` performance in terms of transaction throughput, latency, and storage required. We also evaluate the scaling capabilities concerning the number of local transactions, activities, and domains. Goal 2) is to evaluate the system's capability to identify misconformance, given a baseline *ccmodel*. In this section, we first conduct an experimental evaluation, followed by a qualitative analysis and discussion.

Experimental Setup: We deployed an instance of `Hephaestus` on Google Cloud (CPU with eight cores, 32Gb of RAM, SSD). The different event providers are our Hyperledger Fabric connector, and the Hyperledger Besu connector (version 1.0.0). We initialize a RabbitMQ server serving our event collector *rabbitmq-test-server*. Event emitters on the connectors are implemented as RabbitMQ clients. Every experiment was run 50 times (where we removed the first and last 10 runs, considering a total of 30 runs), and we report the average result, along with the standard deviation, for each run. We share the scripts to generate the plots and *ccmodel*s, making the evaluation process reproducible. Furthermore, we save the output of

each evaluation scenario[4] and the generated cross-chain logs[5] and share it with the reader.

Metrics and Workloads: For each run, we capture the following metrics: throughput (*cctx*s) and their latency, storage cost, i.e., performance metrics. We test two scenarios under variable workloads, which we present later in this section. We characterize each scenario as a tuple (*interoperation mode, number of blockchains, event type, and workload*). The interoperation mode states what cross-chain feature we are testing, asset transfers, asset exchanges, or data transfers. While intuitive, for space limitations, we refer to [33] for a detailed explanation. The number of domains reflects the number of ledgers or other systems emitting events in the scenario, namely Hyperledger Fabric, Hyperledger Besu, or a mock blockchain (essentially, we only model the message transmission). Finally, each workload contains details on the number of events, activities, and domains in that scenario. We implemented a workload generator that produces events across different blockchains. Events are then captured by `Hephaestus`.

A. Baseline: Dummy Use Case with Test Receipts

In this section, we depict the evaluation of our system using a mock blockchain, interoperation mode asset transfer, within a single domain. The workload consists of 6 events, 6 activities, with *ccmodel* generation algorithm $\mathcal{G} = $ `inductive miner`.

The dummy use case represents a *cctx* composed of 6 *ccevent*s. This transaction locks an asset from a source blockchain and unlocks a representation of the same asset on a target blockchain (typically using parties called relayers[6]). Instead of using blockchains to collect receipts, receipts are emitted by a single mock blockchain, which we call the *test blockchain*. The mock blockchain processes transactions as detailed in Section VI-C, namely *create asset*, *lock asset*, *mint asset*, *transfer asset* (optional), and *burn asset representation*.

We measure the performance of the following phases (see Figure 3a): the *Infrastructure Setup* (phase 1), the emission and polling of local transactions, as events *Emit Local Transactions* (phase 2.1), and *Poll Local Transactions* (phase 2.2), the creation of *cctx*s, *Create cctx* (phase 3.1) and the creation of the *ccmodel*, *Create ccmodel* (phase 3.2). The *infrastructure setup* includes setting the event emitters (connectors, including creating blockchain networks and initializing the connectors), setting the event collector (RabbitMQ server), and setting up `Hephaestus`. The *Emit Local Transactions* phase emits test events or issues transactions against the deployed ledgers. The *Poll Local Transactions* waits for the events and sends them to `Hephaestus` for processing. The *Create cctx* generation includes mapping the local transactions to *cctx*s,

---

[4]Online: https://rafaelapb.page.link/cctx-viz-output

[5]Online: https://rafaelapb.page.link/cctx-viz-csv

[6]We could model a third-party responsible for carrying proofs of on-chain execution, the relayer [1] - yielding two more events, in addition to the modeled six. Those two extra events are modeled as activities: *generate proof*, which contains a payload that certifies an asset was locked from the source chain (the proof), and *submit proof*, an event asserting a transaction with proof was submitted to be validated.
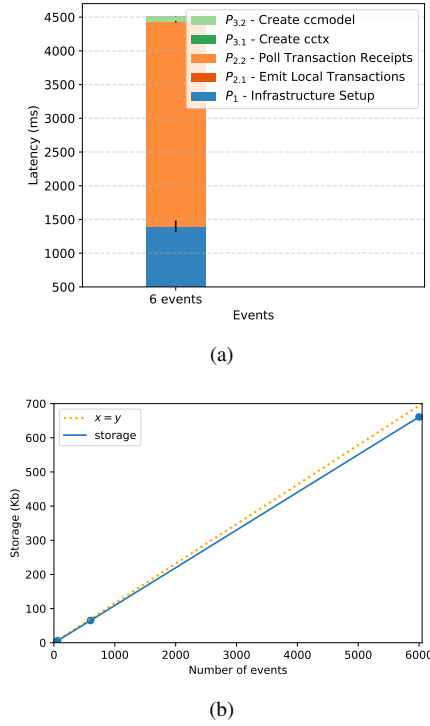
(a)

(b)

Fig. 4. Figure a) shows the latency, in milliseconds, for each phase of the baseline test scenario. Figure b) storage requirements, in kilobytes, for a variable number of events.

| | Phase 1 | | Phase 2 | | | | Phase 3 | | | |
| | | | Phase 2.1 | | Phase 2.2 | | Phase 3.1 | | Phase 3.2 | |
events	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
6	1397.53	83.06	0.47	0.51	3030.27	8.94	0.67	0.61	83.51	1.95
60	1387.93	20.09	2.20	0.66	3068.97	19.07	1.27	0.45	87.2	0.93
600	1388.33	22.26	13.03	3.02	3163.47	21.89	7.53	1.14	98.21	1.51
6000	1392.20	18.05	116.97	20.67	3459.37	18.36	26.5	3.42	265.61	4.18

TABLE III
END-TO-END PROCESS LATENCY MEAN (μ) AND STANDARD DEVIATION (σ), IN MILLISECONDS, AS A FUNCTION OF THE NUMBER OF EVENTS.

and calculating *cctx* metrics. Figure 4a) shows the latency phase breakdown for the emission of six events. We can observe that the setup phase takes around 1.5 seconds, and the most time-consuming phase takes approximately 3 seconds, despite being mock transactions. Figure 5 shows the same breakdown for a variable number of events. Table III supports this figure by reporting the mean end-to-end latency for each phase along with the standard deviation. Phases 1 and 2.2 remain practically constant. Phase 2.1 is sublinear. Phase 3.2's performance indicates that after a certain threshold (between 600 and 6000 transactions), the system starts bottlenecking.

We measure the storage required for generating, storing, and processing events into a *ccmodel*. Figure 4b) shows the required storage as a function of the number of events created. The RabbitMQ container and respective runtime data occupy 257Mb and 72.4kB, respectively. The storage requirements appear to be sublinear to the number of events - six events (one *cctx*) occupy 789 bytes, while six thousand events occupy around 6.6Mb. For a *cctx*, means each *cctx* occupies around 789 bytes + derived data (metrics, a few bytes). Since the metrics are five floats, a date, a string with 128 chars, and a list of events, each *cctx* occupies at least 937 bytes. Finally, the *cross-chain model generation phase* includes parsing the created *cctx*s and generating the *ccmodel*. The generated BPMN model for the dummy use case scenario is represented in Figure 5. Each *cctx* takes 985 milliseconds to build.

B. Use Case: Asset Transfer across Heterogeneous Networks

In this section, we depict the evaluation of our system using two blockchains. The interoperation mode is asset transfer within two domains. The workload is composed of batches of 6 transactions (2 Besu plus 4 Fabric), and 6 activities, with *ccmodel* generation algorithm $\mathcal{G} = $ `inductive miner`.

Next, we illustrate an asset transfer between a private network running Hyperledger Besu and a private network running Hyperledger Fabric, implementing a cross-chain bridge. The rule set for a valid cross-chain asset transfer is illustrated in Equation 2, from Section III. The asset transfer process is the same as in the baseline scenario, i.e., a *cctx* is composed of 6 events, where two are local blockchain transactions. An `Hephaestus` instance is connected to a Fabric connector and a Besu connector. Each connector is connected to a Fabric network version 2.2 and Besu network version 21.1, respectively. The Fabric network consists of 2 peers and 1 orderer, using Raft as the consensus protocol of orderers and LevelDB to maintain the local storage in each node. The Besu network consists of a solo node network. The use case explored in this section follows the same transaction flow as the baseline, i.e., transactions *create asset*, *lock asset*, *mint asset*, *transfer asset* (two of them) and *burn asset representation* are issued in this order. This flow implements a simplified version of a cross-chain promissory note transfer [23] between Hyperledger Besu and Hyperledger Fabric. We used the smart contracts described in Section VI. Figure 6 depicts the normal functioning of the bridge, and also a scenario where double spending happens.

When testing the variable workload of 6-6000 events, the "Infrastructure Setup" and "Poll Transactions Receipt" phases take the most time, as expected. The median latency required for these phases is 1392 and 3469 seconds, respectively, for 6000 events. The infrastructure setup phase takes 90%, 56%, and 12% of the overall execution latency, while the transaction emission takes 7%, 42%, and 88%, for 6, 60, and 600 transactions, respectively. Since these phases take most of the execution time, we illustrate the breakdown of the remaining phases, "Poll Local Transactions", "Create cctx", and "Create ccmodel", in Figure 4. The storage requirements are similar to the baseline use case. For a 60-event execution, we obtain that each *cctx* (6 events) takes 2,04 seconds to construct.

C. Baseline Vs. Use Case

The bottleneck for both scenarios is the infrastructure setup (phase 1) and transaction emission phase (2.1) or polling transactions (phase 2.2). For the use case, the bottlenecks are the infrastructure setup (phase 1) and receipt emission phases (phase 2.1). We observe that the infrastructure setup
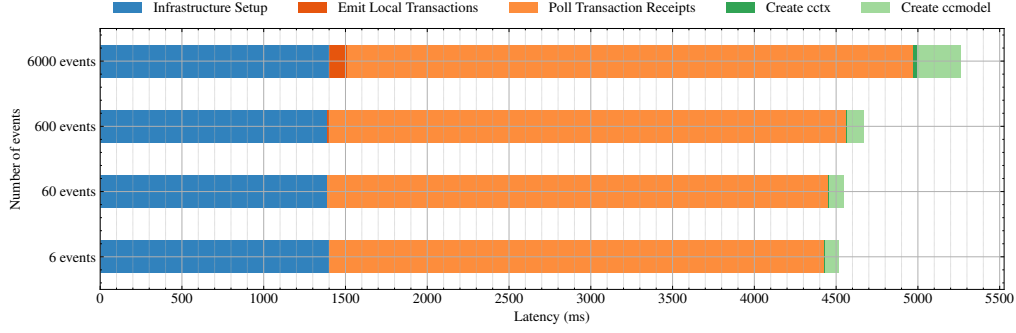
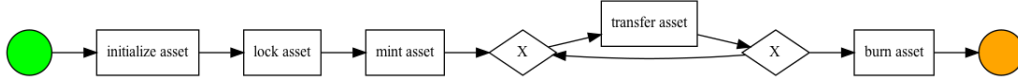Fig. 5. Latency for each phase of the baseline test scenario for a variable number of events.

Fig. 6. Generated business process modelling notation model for the events generated in the baseline phase.

and transaction emission phases occupy around 97%, 98%, and practically 100% of the execution time, depending on if we are emitting 6,60, or 600 events, respectively. This is expected, and we conclude the bottleneck is the transaction execution and commitment. In the dummy use case creating receipts is a cheaper task than retrieving them; in this use case, the inverse happens because executing transactions on blockchains is generally an expensive task. In a production environment, the cross-chain throughput is limited by the finality speed of the underlying systems: the infrastructural part of Hephaestus is efficient in issuing the transactions and retrieving the respective receipts. Varying the number of domains/blockchains should not affect the creation of *cctx* as all transaction receipts are interpreted as *ccevent*s. However, a varying number of domains might influence the overall latency of the system depending on their transaction generation rate. In other words, the faster a domain is, the faster the "Emit local transactions" phase will be, and consequently, the faster *cctx*s and *ccstate* will be processed. The latency of a cross-chain transaction $cctx_i$, denoted by $\delta(cctx_i)$, will be the sum of the individual latencies (in case local transactions are sequential), or bounded by the slowest domain (paralellized): $\delta(cctx_i) = \max\{\delta(d_1), \ldots, \delta(d_n)\}$. Of course, this will depend on the specific cross-chain logic, as there are cases where some local transactions can be parallelized and others not. We leave experiments on real-world bridges, with a variable number of domains for future work.

The complexity of transforming the receipts into events may vary significantly, but our experiments show a very low overhead. Furthermore, the setup phase only needs to be performed once. We conclude that our system is scalable in terms of latency and extensibility.

D. Preventing Cross-Chain Double-Spends

In this section, we run experiments that allow us to evaluate if Hephaestus can detect deviations from expected behavior, namely detecting double-spends.
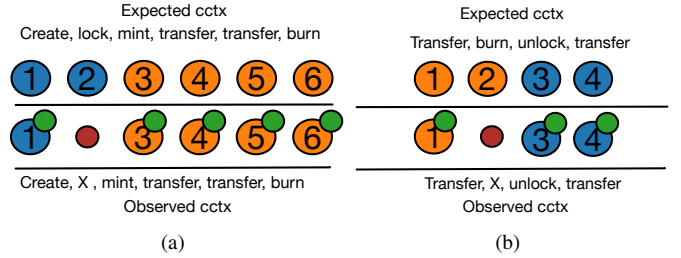

Fig. 7. Double spend detection. Figure a) shows a double spend direction source-to-target blockchain. Figure b) shows a double spend, direction target-to-source blockchain.

Expected behavior, or the specification, is given by the events we emit. After generating the *ccmodel*, we generate another set of events, this time in the following order: create asset, mint asset, transfer asset, burn asset. Note that the lock asset phase is not present - this emulates a user attempting to mint an asset without an appropriate lock (double spend case one). This mechanism models what happened in real-world bridge hacks, namely the PolyNetwork bridge (using a truncated function signature hash collisions and forced transaction inclusions) and the Meter bridge (via inconsistent deposit logic) [43]. Events originate on the source chain ●, or target chain ●. Events can lead to a *SYNC MOVE* ● or *MOVE ON LOG* ●. The latter indicates the point in the cross-chain model where an attack has happened.

Figure 7 shows the detection of double spending. We obtain detailed alignment information about transitions that did not execute correctly, namely the mint before the lock, according to Figure 7a). While it is possible to obtain a set of detailed metrics such as throughput and cost analysis of real-time flows, we stick to the conformance of the process (the fitness) for the sake of space. Fitness is a simple metric used in process mining that consists in measuring the ratio between *SYNC MOVE*s and *MOVE ON LOG*s. We obtain that a mint should occur after it has occurred ('MintAsset', '>>'), and fitness 82%. The trace generated by our implementation

and the expected traces differ, originating a *MOVE ON LOG* ●. We leave the study of applying different process mining algorithms to create cross-chain models of different real-world interoperability systems for future work. Along with this future work, we will study the different trade-offs between algorithms, including performance considerations (using metrics such as recall, precision, accuracy, F-Score, and error rate). Further research is needed to understand the optimal algorithms for each bridge and user cohort.

Double-spends can also happen in the contrary direction (Figure 7b). After a user bridged an asset, they can recover it by burning the bridged asset on the target blockchain and unlocking the original asset on the source blockchain. Double spending occurs when a false proof-of-burn is provided, making the user maintain the bridged token and the original token. This attack happened on the Polygon/Matic bridge (via incorrect proof-of-burn verification).

As soon as double-spends are detected, several defense mechanisms can be activated [43], limiting the scope of the attacks. If assets are frozen in due course, double spending can be not only alleviated but prevented. In conclusion, our system can detect several types of attacks directly mappable to real-world occurrences, without loss of generality.

E. Discussion

Hephaestus contributes to mitigating bridge hacks by 1) generating a *ccmodel* of the bridge protocols, allowing reasoning about the protocol flow, bottlenecks, and possible threats and vulnerabilities, and 2) minimizing the attack consequences by finding active monitoring and detecting suspicious behavior in real-time. Cross-chain models allow expressing complex cross-chain logic without having the protocol designer focus on timeouts, missing or corrupted information, and the technicalities of ad-hoc protocols. This allows the designer to focus instead on the business logic and its monitoring and achieve a separation of concerns.

Our tool can be extended to incorporate an incident framework that is activated upon detection of a *MOVE ON LOG* (e.g., freeze certain types of transactions), according to what is starting to be explored in the industry [42]. For this, adequate *ccmodel* representations are needed. We generate BPMN models, that are good for expressing the semantics of a cross-chain use case graphically, but research on what is an appropriate representation of cross-chain processes is still lacking. An important assumption is that the model is complete, i.e., models all the desired behavior. However, this is not always the case, and some *MOVE ON LOG* events can be false negatives. Creating robust models that tolerate noise and evaluating those models is an evolving, core challenge in the process mining area that would have repercussions in generating and maintaining *ccmodel*s [28]. A consideration of cross-chain security is that cross-chain models are deemed correct when certain predicates on incoming events are satisfied. To this end, designing the rules is of utmost importance, likely to be an interactive process involving different stakeholders; another consideration is that sometimes the events that are checked against rules are not controlled by the bridge operators

- leaving a wide attack surface for hackers. Regarding privacy, there are multiple views on the interoperability literature [46]–[49]. It seems that the main property for cross-chain transactions is unlinkability: the inability of an external observer to link the lock to the mint transactions. However, our system does not provide asset transfer privacy. Further investigation on the security and privacy of *ccmodel*s is needed.

Finally, our system is modular - new blockchains can easily be supported. It has the potential to be integrated into cross-chain APIs for such purposes. The possibility of retrieving the cumulative metrics for all *cctx*s processed in the *ccmodel* allows enhanced and fine-grain monitoring of cross-chain logic. For example, the revenue and cost parameters can be adjusted according to the use case, allowing semantically enriching each transaction. Associating cost and revenue values to transaction receipts would help calculate capital profit taxes for a certain jurisdiction, for instance.

VIII. RELATED WORK

Hephaestus is the result of an inter-disciplinary work that combines the fields of blockchain interoperability, on-chain analytics, and process mining applied to the blockchain.

A. Bridge Hacks and Monitoring

Lee et al. explored a systematization of cross-chain bridge hacks that supports our modeling of double spend [43]. Although some work on bridge security has been done recently [46], [49]–[54], the space still needs systematization. Our work puts forward the cross-chain model, a concept that unites several efforts in the area. The work by Zhang et al. [50] seems to be the most similar to ours. The authors create a tool to identify miss-conformance in the lock-unlock bridge mechanism. However, this work is directed specifically at bridges and not arbitrary cross-chain use cases. On the other hand, BUNGEE is a general-agnostic framework that inspires this work. In this paper, a tool that produces consolidated views over user activity on different blockchains [2] is proposed. Hephaestus can complement BUNGEE to generate metrics, protocol behavior patterns, and individual user activity. Hephaestus can be deployed over interoperability protocols such as ODAP/SAT [23], [55], [56], XCLAIM [37], and many others [1], to provide a monitoring layer.

B. On-chain Analytics

In the field of on-chain analytics, some industry solutions exist and are well-adopted: the Dune tool allows to Explore, create and share crypto analytics, including key metrics for DeFi, NFTs, and more, expressive queries, and the visualization of information in dashboards. Hephaestus would allow for the creation of a cross-chain Dune tool by cross-referencing transactions in multiple chains [57]. Chainanalysis provides a dashboard for investigation, compliance, and risk management tools to assert compliance with jurisdictions and fight fraud and illicit activities [58]. For example, it allows one to visualize the flow of funds and track movements across currencies. Our tool would provide possibilities to port

this monitoring for the cross-chain scenario. Metla finance [59], Morali [60], and Rokti [61] allows a unified view of user assets over different blockchains. `Hephaestus` would allow extending the views to support arbitrary states across blockchains. Certik provides a monitoring layer for analyzing and monitoring blockchain protocols and DeFi projects, but only from a security perspective [62]. Token Flow is the closest work to ours, an analytics tool to track cross-chain asset transfers [63]. However, Token Flow does not support arbitrary cross-chain use cases. On the academic side, we have several tools that allow on-chain analysis of smart contracts for security purposes [40], [64], [65], performance [66], [66], [67], compliance and anti-fraud [68], and others [69], [70] . However, such projects provide a sort of meta-view over user activity, do not provide specific information about interaction with protocols, and are not generalizable, contrarily to this work.

C. Process mining on blockchain

In the process mining area, some work has been done to apply it to the blockchain. In [71], the authors used process mining techniques to specify the behavior of the Augur protocol, discovering bottlenecks and proposing improvements. Some tools to automatize the creation of process models from blockchain protocols to facilitate multiple goals have been proposed [72]–[75], but none for the cross-chain scenario.

IX. Concluding Remarks

The need for multi-chain applications introduces additional challenges to end-users and developers, where we emphasize new attack vectors and a large attack surface. The exploitation of these threats leads to large-scale attacks on cross-chain bridges. We propose `Hephaestus` to address this problem.

`Hephaestus` generates cross-chain models from observed cross-chain events. By associating a set of transactions with a set of rules, transactional flow can be monitored and, therefore, deviations from the idealized process can be detected. This allows to timely act upon suspicious behavior that can indicate an attack. We implemented `Hephaestus`, several blockchain connectors, test ledgers, and a workload generator. Our evaluation includes creating a cross-chain use case on asset transfers (i.e., a bridge) composed of a pair of smart contracts and cross-chain logic. We tested our system with variable workloads to assess the performance and reliability of `Hephaestus`. We conclude that we have low latency in generating *ccmodel*s for the given use case and that our tool can scale with the number of blockchains and *cctx*s.

We pave the way to enable a better user experience for the end user and protocol operators by enabling the analysis, monitoring, and optimization of *ccmodel*s. In particular, `Hephaestus` can be applied over established blockchain interoperability protocols, serving as a monitoring and audit layer, providing better response capacity and thus enhanced proactive security. Use cases such as reconfiguring wallets across chains, better user interfaces for fund tracking across different chains, managing additional base layer tokens for gas, doing tax reports, and analyzing cross-chain maximal extractable value do not need to be complicated.

References

[1] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A Survey on Blockchain Interoperability: Past, Present, and Future Trends," *ACM Computing Surveys*, vol. 54, no. 8, pp. 1–41, May 2021. [Online]. Available: http://arxiv.org/abs/2005.14282

[2] R. Belchior, L. Torres, J. Pfannschmid, A. Vasconcelos, and M. Correia, "Is My Perspective Better Than Yours? Blockchain Interoperability with Views." TechRxiv, Jun. 2022. [Online]. Available: https://www.techrxiv.org/articles/preprint/Is_My_Perspective_Better_Than_Yours_Blockchain_Interoperability_with_Views/20025857/1

[3] B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, "Cross-blockchain technology: integration framework and security assumptions," *IEEE Access*, 2022, publisher: IEEE.

[4] P. Robinson, "Survey of crosschain communications protocols," *Computer Networks*, vol. 200, p. 108488, Dec. 2021, publisher: Elsevier. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128621004321

[5] R. Belchior, J. Süßenguth, Q. Feng, T. Hardjono, A. Vasconcelos, and M. Correia, "A Brief History of Blockchain Interoperability," 6 2023. [Online]. Available: https://www.techrxiv.org/articles/preprint/A_Brief_History_of_Blockchain_Interoperability/23418677

[6] H. Qureshi, "Axelar, Bridges, and Blockchain Globalization," Jun. 2022. [Online]. Available: https://medium.com/dragonfly-research/axelar-bridges-and-blockchain-globalization-11ef3bbce9f1

[7] Ethereum Engineering Group, "Security of Crosschain Transactions and Bridges," Nov. 2022. [Online]. Available: https://www.youtube.com/watch?v=DJyEJVaXMNo

[8] D. Berenzon, "Blockchain Bridges," Sep. 2021. [Online]. Available: https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8

[9] P. KidBold, "The Wormhole Bridge Attack Explained," Feb. 2022. [Online]. Available: https://kaicho.substack.com/p/the-wormhole-bridge-attack-explained

[10] C. Faife, "Wormhole cryptocurrency platform hacked for $325 million after error on GitHub," Feb. 2022. [Online]. Available: https://www.theverge.com/2022/2/3/22916111/wormhole-hack-github-error-325-million-theft-ethereum-solana

[11] Rekt, "Rekt - THORChain," 2022. [Online]. Available: https://www.rekt.news/

[12] R. Behnke, "Explained: The Wormhole Hack (February 2022)," Feb. 2022. [Online]. Available: https://halborn.com/explained-the-wormhole-hack-february-2022/

[13] FreddieChopin, "FYI, the hacker who exploited Harmony bridge for 100 M$ 3 days ago has already started sending stolen ETH to Tornado Cash mixer," Jun. 2022. [Online]. Available: www.reddit.com/r/CryptoCurrency/comments/vlt4xs/fyi_the_hacker_who_exploited_harmony_bridge_for/

[14] M. Barrett, "Harmony's Horizon Bridge Hack," Jun. 2022. [Online]. Available: https://medium.com/harmony-one/harmonys-horizon-bridge-hack-1e8d283b6d66

[15] C. Faife, "Nomad crypto bridge loses $200 million in "chaotic" hack," Aug. 2022. [Online]. Available: https://www.theverge.com/2022/8/2/23288785/nomad-bridge-200-million-chaotic-hack-smart-contract-cryptocurrency

[16] R. News. Multichain hacked for the third time - R3KT news. rekt. [Online]. Available: https://www.rekt.news/

[17] The Block Research, "Largest DeFi exploits," 2022. [Online]. Available: https://www.theblock.co/data/decentralized-finance/exploits/largest-defi-exploits

[18] Zach, Ally, "A Year of Bridge Exploits," Aug. 2022. [Online]. Available: https://messari.io/report/a-year-of-bridge-exploits

[19] The Straits Times, "Cryptocurrency-bridge hacks top $1.36 billion in little over a year," *The Straits Times*, Apr. 2022. [Online]. Available: https://www.straitstimes.com/tech/tech-news/cryptocurrency-bridge-hacks-top-136-billion-in-little-over-a-year

[20] N. Team, "The Road to Recovery," Aug. 2022. [Online]. Available: https://medium.com/nomad-xyz-blog/the-road-to-recovery-6abe5eec8ff1

[21] V. Buterin, "Vitalik Buterin on cross-chain bridges," 2022. [Online]. Available: www.reddit.com/r/ethereum/comments/rwojtk/ama_we_are_the_efs_research_team_pt_7_07_january/hrngyk8/

[22] D. Avrilionis and T. Hardjono, "Towards Blockchain-enabled Open Architectures for Scalable Digital Asset Platforms," Oct. 2021, publisher: ArXiv. [Online]. Available: https://www.scienceopen.com/document?vid=c60d84b9-911e-45a5-ab92-864ee24ec771

[23] R. Belchior, A. Vasconcelos, M. Correia, and T. Hardjono, "HERMES: Fault-Tolerant Middleware for Blockchain Interoperability," *Future Generation Computer Systems*, Mar. 2021.

[24] C. Ko, M. Ruschitzka, and K. Levitt, "Execution monitoring of security-critical programs in distributed systems: a specification-based approach," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, May 1997, pp. 175–187, iSSN: 1081-6011.

[25] H. Montgomery, H. Borne-Pons, J. Hamilton, M. Bowman, P. Somogyvari, S. Fujimoto, T. Takeuchi, T. Kuhrt, and R. Belchior, "Hyperledger Cactus Whitepaper," Hyperledger Foundation, Tech. Rep., 2020. [Online]. Available: https://github.com/hyperledger/cactus/blob/master/docs/whitepaper/whitepaper.md

[26] L2BEAT – The state of the layer two ecosystem. [Online]. Available: https://l2beat.com/scaling/summary

[27] R. Belchior, S. Guerreiro, A. Vasconcelos, and M. Correia, "A survey on business process view integration: past, present and future applications to blockchain," *Business Process Management Journal*, vol. ahead-of-print, no. ahead-of-print, Jan. 2022. [Online]. Available: https://doi.org/10.1108/BPMJ-11-2020-0529

[28] W. Van Der Aalst, "Process mining: Overview and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 3, no. 2, pp. 1–17, 2012, publisher: ACM New York, NY, USA.

[29] J. Küster, K. Ryndina, and H. Gall, "Generation of business process models for object life cycle compliance," in *International Conference on Business Process Management*, vol. 4714 LNCS. Springer, Berlin, 2007, pp. 165–181.

[30] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584908000323

[31] R. Belchior, L. Torres, J. Pfannschmid, A. Vasconcelos, and M. Correia, "Can We Share the Same Perspective? Blockchain Interoperability with Views," Oct. 2022. [Online]. Available: https://www.techrxiv.org/articles/preprint/Is_My_Perspective_Better_Than_Yours_Blockchain_Interoperability_with_Views/20025857/3

[32] J. Garay, A. Kiayias, and N. Leonardos, "The Bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology*, vol. 9057, 2015, pp. 281–310, iSSN: 16113349.

[33] R. Belchior, L. Riley, T. Hardjono, A. Vasconcelos, and M. Correia, "Do You Need a Distributed Ledger Technology Interoperability Solution?" *Distributed Ledger Technologies: Research and Practice*, Sep. 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3564532

[34] J. Han, H. E, G. Le, and J. Du, "Survey on NoSQL database," in *2011 6th International Conference on Pervasive Computing and Applications*, 2011, pp. 363–366.

[35] P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. Sears, "Dedalus: Datalog in Time and Space," in *Datalog Reloaded*, ser. Lecture Notes in Computer Science, O. de Moor, G. Gottlob, T. Furche, and A. Sellers, Eds. Berlin, Heidelberg: Springer, 2011, pp. 262–281.

[36] T. J. Green, S. S. Huang, B. T. Loo, and W. Zhou, "Datalog and Recursive Query Processing," *Foundations and Trends® in Databases*, vol. 5, no. 2, pp. 105–195, Nov. 2013, publisher: Now Publishers, Inc. [Online]. Available: https://www.nowpublishers.com/article/Details/DBS-017

[37] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 193–210.

[38] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt, "Sok: Communication across distributed ledgers," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 3–36.

[39] I. Mihaiu, R. Belchior, S. Scuri, and N. Nunes, "A Framework to Evaluate Blockchain Interoperability Solutions," TechRxiv, Tech. Rep., Dec. 2021. [Online]. Available: https://www.techrxiv.org/articles/preprint/A_Framework_to_Evaluate_Blockchain_Interoperability_Solutions/17093039

[40] B. Putz and G. Pernul, "Detecting Blockchain Security Threats," in *2020 IEEE International Conference on Blockchain (Blockchain)*, Nov. 2020, pp. 313–320.

[41] C. Page, "Binance hit by $100 million blockchain bridge hack," Oct. 2022. [Online]. Available: https://techcrunch.com/2022/10/07/blockchain-bridge-hack/

[42] Chainlink, "Cross-Chain Interoperability Protocol (CCIP) | Chainlink," 2022. [Online]. Available: https://chain.link/cross-chain

[43] S.-S. Lee, A. Murashkin, M. Derka, and J. Gorzny, "SoK: Not Quite Water Under the Bridge: Review of Cross-Chain Bridge Hacks," Oct. 2022, arXiv:2210.16209 [cs]. [Online]. Available: http://arxiv.org/abs/2210.16209

[44] A. Berti, S. J. Van Zelst, and W. van der Aalst, "Process mining for python (PM4Py): bridging the gap between process-and data science," *arXiv preprint arXiv:1905.06169*, 2019.

[45] W. M. van der Aalst and A. Berti, "Discovering object-centric Petri nets," *Fundamenta informaticae*, vol. 175, no. 1-4, pp. 1–40, 2020, publisher: IOS Press.

[46] T. Haugum, B. Hoff, M. Alsadi, and J. Li, "Security and Privacy Challenges in Blockchain Interoperability - A Multivocal Literature Review," in *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, ser. EASE '22. New York, NY, USA: Association for Computing Machinery, Jun. 2022, pp. 347–356. [Online]. Available: https://doi.org/10.1145/3530019.3531345

[47] A. Deshpande and M. Herlihy, "Privacy-Preserving Cross-Chain Atomic Swaps," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, M. Bernhard, A. Bracciali, L. J. Camp, S. Matsuo, A. Maurushat, P. B. Rønne, and M. Sala, Eds. Cham: Springer International Publishing, 2020, pp. 540–549.

[48] Z. Yin, B. Zhang, J. Xu, K. Lu, and K. Ren, "Bool Network: An Open, Distributed, Secure Cross-Chain Notary Platform," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3465–3478, 2022, conference Name: IEEE Transactions on Information Forensics and Security.

[49] J. Wang, J. Cheng, Y. Yuan, H. Li, and V. S. Sheng, "A Survey on Privacy Protection of Cross-Chain," in *Advances in Artificial Intelligence and Security*, ser. Communications in Computer and Information Science, X. Sun, X. Zhang, Z. Xia, and E. Bertino, Eds. Cham: Springer International Publishing, 2022, pp. 283–296.

[50] J. Zhang, J. Gao, Y. Li, Z. Chen, Z. Guan, and Z. Chen, "Xscope: Hunting for Cross-Chain Bridge Attacks," Aug. 2022, arXiv:2208.07119 [cs]. [Online]. Available: http://arxiv.org/abs/2208.07119

[51] Y. Zhang, Z. Ge, Y. Long, and D. Gu, "UCC: Universal and Committee-based Cross-chain Framework," in *Information Security Practice and Experience*, ser. Lecture Notes in Computer Science, C. Su, D. Gritzalis, and V. Piuri, Eds. Cham: Springer International Publishing, 2022, pp. 93–111.

[52] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkBridge: Trustless Cross-chain Bridges Made Practical," Oct. 2022, arXiv:2210.00264 [cs]. [Online]. Available: http://arxiv.org/abs/2210.00264

[53] C. Pedreira, R. Belchior, M. Matos, and A. Vasconcelos, "Securing Cross-Chain Asset Transfers on Permissioned Blockchains," Jun. 2022. [Online]. Available: https://www.techrxiv.org/articles/preprint/Trustable_Blockchain_Interoperability_Securing_Asset_Transfers_on_Permissioned_Blockchains/19651248/3

[54] A. Augusto, R. Belchior, A. Vasconcelos, and T. Hardjono, "Resilient Gateway-Based N-N Cross-Chain Asset Transfers," Nov. 2022. [Online]. Available: https://www.techrxiv.org/articles/preprint/Resilient_Gateway-Based_N-N_Cross-Chain_Asset_Transfers/20016815/2

[55] M. Hargreaves, T. Hardjono, and R. Belchior, "Open Digital Asset Protocol draft 02," Internet Engineering Task Force, Tech. Rep., 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-hargreaves-odap-02

[56] R. Belchior, M. Correia, and T. Hardjono, "Gateway Crash Recovery Mechanism draft v1," IETF, Tech. Rep., 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-belchior-gateway-recovery/

[57] "Dune." [Online]. Available: https://dune.com/home

[58] Chainanalysis, "The Blockchain Data Platform - Chainalysis," 2022. [Online]. Available: https://www.chainalysis.com/

[59] Metla, "Metla - the ultimate crypto dashboard," 2022. [Online]. Available: https://metla.com/

[60] Moralis, "Moralis The Web3 Development Workflow," 2022. [Online]. Available: https://moralis.io/

[61] Rokti, "Rokti portfolio tracker," 2022. [Online]. Available: https://rotki.com

[62] Certik, "CertiK Blockchain Security Leaderboard," 2022. [Online]. Available: https://www.certik.com

[63] Token Flow, "Token Flow Insights," 2022. [Online]. Available: https://tokenflow.live

[64] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.

[65] B. Putz, F. Böhm, and G. Pernul, "HyperSec: Visual Analytics for blockchain security monitoring," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2021, pp. 165–180.

[66] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A Detailed and Real-Time Performance Monitoring Framework for Blockchain Systems," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, May 2018, pp. 134–143.

[67] M. Bartoletti, S. Lande, L. Pompianu, and A. Bracciali, "A general framework for blockchain analytics," in *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, 2017, pp. 1–6.

[68] D. N. Dillenberger, P. Novotny, Q. Zhang, P. Jayachandran, H. Gupta, S. Hans, D. Verma, S. Chakraborty, J. Thomas, M. Walli, and others, "Blockchain analytics and artificial intelligence," *IBM Journal of Research and Development*, vol. 63, no. 2/3, pp. 5–1, 2019, publisher: IBM.

[69] N. Tovanich, N. Soulié, N. Heulot, and P. Isenberg, "An Empirical Analysis of Pool Hopping Behavior in the Bitcoin Blockchain," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2021, pp. 1–9.

[70] B. Nasrulin, M. Muzammal, and Q. Qu, "Chainmob: Mobility analytics on blockchain," in *2018 19th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2018, pp. 292–293.

[71] R. Hobeck, C. Klinkmüller, H. Bandara, I. Weber, and W. M. van der Aalst, "Process mining on blockchain data: a case study of Augur," in *International conference on business process management*. Springer, 2021, pp. 306–323.

[72] M. Müller and P. Ruppel, "Process Mining for Decentralized Applications," in *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, Apr. 2019, pp. 164–169.

[73] C. Klinkmüller, A. Ponomarev, A. B. Tran, I. Weber, and W. van der Aalst, "Mining Blockchain Processes: Extracting Process Mining Data from Blockchain Applications," in *Business Process Management: Blockchain and Central and Eastern Europe Forum*, ser. Lecture Notes in Business Information Processing, C. Di Ciccio, R. Gabryelczyk, L. García-Bañuelos, T. Hernaus, R. Hull, M. Indihar Štemberger, A. Kő, and M. Staples, Eds. Cham: Springer International Publishing, 2019, pp. 71–86.

[74] R. Mühlberger, S. Bachhofner, C. Di Ciccio, L. García-Bañuelos, and O. López-Pintado, "Extracting Event Logs for Process Mining from Data Stored on the Blockchain," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, C. Di Francescomarino, R. Dijkman, and U. Zdun, Eds. Cham: Springer International Publishing, 2019, pp. 690–703.

[75] F. Corradini, F. Marcantoni, A. Morichetta, A. Polini, B. Re, and M. Sampaolo, "Enabling Auditing of Smart Contracts Through Process Mining," in *From Software Engineering to Formal Methods and Tools, and Back: Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday*, ser. Lecture Notes in Computer Science, M. H. ter Beek, A. Fantechi, and L. Semini, Eds. Cham: Springer International Publishing, 2019, pp. 467–480. [Online]. Available: https://doi.org/10.1007/978-3-030-30985-5_27

Rafael Belchior is a researcher at INESC-ID (Distributed Systems Group) and a Ph.D. student at Técnico Lisboa. Studying the intersection of blockchain interoperability and security, he focuses on enabling interoperability across heterogeneous systems and building resilient blockchain infrastructure.

Peter Somogyvari is a technology architect manager at Accenture, where he is one of the maintainers of the top-level Hyperledger project called Cacti, which aims to be an enterprise-grade framework for blockchain integration/interoperability. He has spoken at Hyperledger Global Forum in 2020 and 2021 and several other technology conferences.

Jonas Pfannschmidt has more than 15 years of professional experience in software engineering with a focus on Blockchain, Cloud, and Financial Services. In his current roles as Principal Blockchain Engineer, R&D lead, and Director of Blockdaemon Ltd. he builds institutional-grade Blockchain infrastructure to stake, deploy, and scale leading Blockchain networks.

André Vasconcelos is Assistant Professor (Professor Auxiliar) in the Department of Computer Science and Engineering, Instituto Superior Tecnico, Lisbon University, and researcher in Information and Decision Support Systems Lab at INESC-ID, in Enterprise Architecture domains, namely representation and modeling of Architectures of Information Systems, and Evaluation of Information Systems Architectures.

Miguel Correia is a Full Professor at Instituto Superior Tecnico (IST), Universidade de Lisboa, in Lisboa, Portugal. He is vice president for faculty at DEI. He is the coordinator of the Doctoral Program in Information Security at IST. He is a senior researcher at INESC-ID, and a member of the Distributed Systems Group (GSD). He is co-chair of the European Blockchain Partnership that is designing the European Blockchain Services Infrastructure (EBSI).