On HTLC-Based Protocols for Multi-Party Cross-Chain Swaps

Emily Clark
University of California, Riverside, CA, USA

Chloe Georgiou
University of California, Riverside, CA, USA

Katelyn Poon
University of California, Riverside, CA, USA

Marek Chrobak
University of California, Riverside, CA, USA

Abstract

In his 2018 paper, Herlihy introduced an atomic protocol for multi-party asset swaps across different blockchains. Practical implementation of this protocol is hampered by its intricacy and computational complexity, as it relies on elaborate smart contracts for asset transfers, and specifying the protocol's steps on a given digraph requires solving an $\mathbb{NP}$-hard problem of computing longest paths. Herlihy left open the question whether there is a simple and efficient protocol for cross-chain asset swaps in arbitrary digraphs. Addressing this, we study *HTLC-based protocols*, in which all asset transfers are implemented with standard hashed time-lock smart contracts (HTLCs). Our main contribution is a full characterization of swap digraphs that have such protocols, in terms of so-called reuniclus graphs. We give an atomic HTLC-based protocol for reuniclus graphs. Our protocol is simple and efficient. We then prove that non-reuniclus graphs do not have atomic HTLC-based swap protocols.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases distributed computing, blockchain, asset swaps

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2024.22

Related Version All proofs omitted in this extended abstract can be found in the full version.
Full Version: `https://doi.org/10.48550/arXiv.2403.03906` [5]

Funding Research partially supported by National Science Foundation grant CCF-2153723.

1 Introduction

In 2018, Herlihy [9] introduced a model for multi-party asset swaps across different blockchains, where an asset swap is represented by a strongly connected directed graph, with each vertex corresponding to one party and each arc representing a pre-arranged asset transfer between two parties. The goal is to design a protocol to implement the transfer of all assets. The protocol must guarantee, irrespective of the behavior of other parties, that each honest party will end up with an outcome that it considers acceptable. The protocol should also discourage cheating, so that any coalition of parties cannot improve its outcome by deviating from the protocol. These two conditions are called *safety* and *strong Nash equilibrium*, respectively. A protocol that satisfies these conditions is called *atomic*.

In this model, Herlihy [9] developed an atomic protocol for asset swaps in arbitrary strongly connected digraphs. While this result is a significant theoretical advance, its practical implementation is hampered by its intricacy and high computational complexity, as it relies on elaborate smart contracts for asset transfers, and specifying the protocol's steps on a given digraph requires solving an $\mathbb{NP}$-hard problem of computing longest paths. It also uses a cryptographic scheme with nested digital signatures that may reveal the graph's topology to all parties, raising concerns about privacy.

Herlihy [9] also mentions a simpler protocol that uses only standard smart contracts called *hashed time-lock contracts* (HTLC's), that require only one secret/hashlock pair and a time-out mechanism (see [19, 16]). This protocol, however, works correctly only for special types of digraphs that we call *bottleneck graphs.* This raises a natural question, already posed in [9]: *Is there a simple and efficient protocol for multi-party asset swaps that is atomic and works on all strongly connected digraphs?*

Our contributions. Motivated by this question, we study *HTLC-based protocols*, which are allowed to exchange assets only via HTLC's. As it turns out, the class of digraphs that have such protocols is much broader than bottleneck digraphs; in fact, we give a complete characterization of digraphs that admit HTLC-based protocols. We call them *reuniclus graphs.* Roughly, a reuniclus graph can be thought of as a tree of biconnected components, each being an induced bottleneck subgraph. In this terminology, our main contribution can be stated as follows:

▶ **Theorem 1.** *A swap digraph G has an atomic HTLC-based protocol if and only if G is a reuniclus digraph.*

The sufficiency condition is proved by providing an atomic HTLC-based protocol for reuniclus digraphs. The protocol itself is simple and efficient. Also, testing whether a given graph is a reuniclus graph and, if it is, computing the specification of the protocol for each party can be accomplished in linear time. (The key ingredient is the algorithm from [12] that can be used to recognize bottleneck graphs.) Our most technically challenging contribution is the proof of the necessity condition. This requires showing that the atomicity assumption implies some structural properties of the underlying graph. By carefully exploiting this approach, we prove that each digraph with an atomic protocol must have the reuniclus structure.

Our asset-swap model is in fact a slight generalization of the one in [9], as it uses a relaxed definition of the preference relation, which allows each party to customize some of their preferences.

Related work. The problem of securely exchanging digital products between untrustful parties has been studied since 1990s under the name of *fair exchange.* As simultaneous exchange is not feasible in a typical electronic setting, protocols for fair exchange rely on a trusted party – see for example [15, 7, 3, 1, 2]. In the model from [9], smart contracts play the role of trusted parties.

With users now holding assets on a quickly growing number of different blockchains, cross-chain interoperability tools became necessary to allow these users to trade their assets. An atomic swap concept was one of the proposed tools to address this issue. The concept itself and some early implementations of asset-swap protocols (see, for example [18]) predate the work of Herlihy [9].

In recent years there has been intensive research activity on asset-swap protocols. The preference relation of the participants in the model from [9] is very rudimentary, and some refinements of this preference model were studied in [4, 11]. Some proposals [14, 20] address the issue of " grieving", when one party needs to wait for the counter-party to act, while its assets are locked and unaccessible. Other directions of study include investigations of protocol's time and space complexity [11], privacy issues [6], security enhancements [13], and generalizations of swaps to more complex transactions [10, 17, 8].

2 Multi-Party Asset Swaps

The *multi-party asset swap problem* we address is this: There is a set V of parties, each with a set of assets that it wishes to exchange for some other assets. Suppose that there is a way to reassign assets from each current owner to their new owner in such a way that each party would receive exactly their desired assets. This reassignment is called a *multi-party asset swap*. The goal is now to arrange the transfers of these assets. The challenge is that some parties may deviate from the protocol, attempting to improve their outcome, or even behave irrationally. To address this, the asset-swap protocol must satisfy the following safety property in addition to correctness: an outcome of any honest party (that follows the protocol) must be guaranteed to be acceptable (not worse than its initial holdings), even if other parties deviate from the protocol or collaboratively attempt to cheat.

Let $G = (V, A)$ be a digraph with vertex set V and arc set A, without self-loops or parallel arcs. By N_v^{in} we denote the set of in-neighbors of v, by A_v^{in} its set of incoming arcs, N_v^{out} are the out-neighbors of v and A_v^{out} are its outgoing arcs. Other graph notation and terminology is standard.

Clearing service. We assume the existence of a *market clearing service*, where each party submits its proposed exchange (the collections of its current and desired assets). If a swap is possible, the clearing service constructs a digraph $G = (V, A)$ representing this swap. Each arc (u, v) of G represents the intended transfer of one asset from its current owner u (the seller) to its new owner v (the buyer). For simplicity, we assume that any party can transfer only one asset to any other party, and we identify assets with arcs, so (u, v) denotes both an arc of G and the asset of u to be transferred to v. The service ensures that G satisfies the assumptions of the swap protocol, and it informs each party of the protocol's steps. Importantly, we *do not* assume that the parties trust the market clearing service.

Secrets and hashlocks. We allow each party v to create a *secret value* s_v, and convert it into a *hashlock value* $h_v = H(s_v)$, where $H()$ is a one-way permutation. The value of s_v is secret, meaning that no other party has the capability to compute s_v from h_v. The hashlock values can be made public.

Hashed time-lock contracts (HTLCs). Asset transfers are realized with smart contracts, which are simply pieces of code running on a blockchain. The contracts used in our model are called *hashed time-lock contracts*, or *HTLCs*, for short, and are defined as follows: Each contract is associated with an arc (u, v) of G, and is used to transfer the asset (u, v) from u to v. It is created by u, with u providing it with the asset, timeout value τ, and a hashlock value h. Once this contract is created, the possession of the asset is transferred from u to the contract. The counter-party v can access the contract to verify its correctness; in particular, it can learn the hashlock value h. There are two ways in which the asset can be released: (1) One, v can claim it. To claim it successfully, v must provide a value s such that $H(s) = h$ not later than at time τ. When this happens, the smart contract transfers the asset to v, and it gives s to u. (2) Two, the contract can expire. As soon as the current time exceeds τ, and if the asset has not been claimed, the contract returns the asset to u.

Further overloading notation and terminology, we will also refer to the contract on arc (u, v) as "contract (u, v)". If this contract has hashlock h_x of a party x (where x may be different from u and v), we will say that it is *protected by hashlock h_x* or simply *protected by party x*.

Protocols. In an execution of $\mathbb{P}$ there is no guarantee that all parties actually follow $\mathbb{P}$. When we refer to an *honest* or *conforming* party u, we mean that u follows $\mathbb{P}$, except when it can infer that not all parties follow $\mathbb{P}$. From that point on, u may behave arbitrarily (but still rationally).

In an *HTLC-based protocol*, all asset transfers are implemented with HTLCs, and no other interaction between the parties is allowed. Each party can create one secret/hashlock pair. These hashlock values are distributed via smart contracts (or can simply be made public).

Outcomes. For each party v, *v's outcome* associated with an execution of a protocol $\mathbb{P}$ is specified by the sets of assets that are relinquished and acquired by v. Thus such an outcome is a pair $\omega = \langle \omega^{in} \,|\, \omega^{out} \rangle$, where $\omega^{in} \subseteq A_v^{in}$ and $\omega^{out} \subseteq A_v^{out}$. To reduce clutter, instead of arcs, in $\langle \omega^{in} \,|\, \omega^{out} \rangle$ we can list only the corresponding in-neighbors and out-neighbors of v; for example, instead of $\langle \{(x,v),(y,v)\} \,|\, \{(v,z)\} \rangle$ we will write $\langle x, y \,|\, z \rangle$.

An outcome $\omega = \langle \omega^{in} \,|\, \omega^{out} \rangle$ of some party u is called *acceptable* if in this outcome u retains all its own assets or it gains all incoming assets. That is, either $\omega^{in} = A_v^{in}$ or $\omega^{out} = \emptyset$. The following two types of outcomes are particularly significant: $\textsc{Deal}_v = \langle A_v^{in} \,|\, A_v^{out} \rangle$ represents an outcome where all prearranged asset transfers involving v are completed, and $\textsc{NoDeal}_v = \langle \emptyset \,|\, \emptyset \rangle$ represents an outcome where none of the prearranged asset transfers involving v is completed. We will skip the subscript v in these notations whenever v is understood from context[1].

For a set C of parties, its set A_C^{in} of incoming arcs consists of arcs (u,v) with $u \notin C$ and $v \in C$. The set A_C^{out} of outgoing arcs is defined analogously. With this, the concept of outcomes and its properties extend naturally to sets of parties ("coalitions"). For example, an outcome of C is *acceptable* if it either contains all incoming arcs of C or does not contain any outgoing arcs of C.

The preference relation. A *preference relation* of a party v is a partial order on the set of all outcomes for v that satisfies the following three properties: **(p1)** If $\omega_1^{in} \subseteq \omega_2^{in}$ and $\omega_1^{out} \supseteq \omega_2^{out}$, then ω_2 is preferred to ω_1 ; **(p2)** If ω is unacceptable then $\textsc{NoDeal}$ is preferable to ω; **(p3)** $\textsc{Deal}$ is better than $\textsc{NoDeal}$. These are natural: (p1) says that it is better to receive more assets and relinquish fewer assets, and without (p3) v would have no incentive to participate in the protocol. The preference relation captures rational behavior, leading to the definition of Nash equilibrium property, below.

Protocol properties. Following [9], we define the following properties of a swap protocol $\mathbb{P}$:

Liveness: $\mathbb{P}$ is *live* if each party ends up in $\textsc{Deal}$, providing that all parties follow $\mathbb{P}$.

Safety: $\mathbb{P}$ is *safe* if each honest party ends up in an acceptable outcome, independently of the behavior of other parties.

Strong Nash Equilibrium: $\mathbb{P}$ has the *strong Nash equilibrium property* if for any set C of parties, if all parties outside C follow $\mathbb{P}$ then the parties in C cannot improve the outcome of C by deviating from $\mathbb{P}$.

Atomicity: $\mathbb{P}$ is called *atomic* if it's live, safe, and has the strong Nash equilibrium property.

The lemma below is a mild extension of the one in [9]. The point of the lemma is that, in Herlihy's preference model, the strong Nash equilibrium property comes for free. The strong connectivity assumption is necessary for the safety property to hold, see [9]. (See the full paper for the easy proof.)

[1] Herlihy [9] defines other types of outcomes: $\textsc{Discount}$, $\textsc{FreeRide}$ and $\textsc{Underwater}$, but these are not essential – see the full version of the paper.

▶ **Lemma 2.** *Assume that digraph G is strongly connected. If a protocol $\mathbb{P}$ is live and safe then $\mathbb{P}$ is atomic.*

3 An Atomic Protocol for Reuniclus Digraphs

In this section we give an atomic asset-swap protocol for reuniclus digraphs. First, in Section 3.1, we describe an atomic protocol for bottleneck digraphs called Protocol BDP. This protocol is mostly equivalent to the simplified protocol from [9, Section 4.6]. We include it here, because it serves as a stepping stone to our full protocol for reuniclus digraphs that is presented in Section 3.2.

3.1 Protocol BDP for Bottleneck Digraphs

Protocol BDP for leader ℓ:

- *At time* 0*:* Create a secret s_ℓ and compute $h_\ell = H(s_\ell)$. For each arc (ℓ, v), create contract with hashlock h_ℓ and timeout $\tau_{\ell v} = D^* + D^+_v$.
- *At time* D^**:* Claim all incoming assets using secret s_ℓ.

Protocol BDP for a follower u:

- *At time* D^-_u*:* For each arc (u, v), create contract with hashlock h and timeout $\tau_{uv} = D^* + D^+_v$.
- *At time* $D^* + D^+_u$*:* Let s be the secret obtained from the contract for some claimed outgoing assets. Use s to claim all incoming assets.

Figure 1 Protocol BDP, for the leader on the left, and for the followers on the right. Each bullet-point step takes one time unit. In the description we tacitly assume that u aborts if it detects any deviation from the protocol.

A vertex v in a digraph G is called a *bottleneck vertex* if it belongs to each cycle of G. If G is strongly connected and has a bottleneck vertex then we refer to G as a *bottleneck digraph.*

We now describe Protocol BDP for a bottleneck digraph G. One bottleneck vertex of G is designated as the *leader*. This leader, denoted ℓ, creates its secret/hashlock pair (s_ℓ, h_ℓ). The other vertices are called *followers*. Protocol BDP has two phases. The first phase, initiated by ℓ, creates all contracts. Each follower waits for all the incoming contracts to be created, and then creates the outgoing contracts. For followers, the timeout values for all incoming contracts are strictly larger than the timeout values for all outgoing contracts. In the second phase the assets are claimed, starting with ℓ claiming its incoming assets. Now the process proceeds backwards. For each follower v, when any of its outgoing assets is claimed, v learns the secret value s_ℓ, and it can now claim its incoming assets.

The detailed description of this protocol is given in Figure 1. In the protocol, D^-_v denotes the *maximum distance from* ℓ *to* v, defined as the maximum length of a simple path from ℓ to v. In particular, $D^-_\ell = 0$. The values D^-_y are used in contract creation times. By D^* we denote the maximum length of a simple cycle in G, so $D^* = \max_{z \in N^{in}_\ell} D^-_z + 1$.

In the timeout values, the notation D^+_v is the maximum distance from v to ℓ. Naturally, we have $\max_{z \in N^{out}_\ell} D^+_z + 1 = D^*$. Note that, for each v, the timeouts of all incoming contracts (u, v) are equal to $D^* + D^+_v$, exactly when v claims them. Also, if $v \neq \ell$ then $D^* + D^+_v$ is larger than the timeout $D^* + D^+_w$ of each outgoing contract (v, w).

▶ **Theorem 3.** *If G is a bottleneck digraph, then Protocol BDP is an atomic swap protocol for G.*

The safety property should be intuitive: The leader protects its outgoing assets, so it will lose these assets only if it first claims its incoming assets. If a follower loses an outgoing asset, it has at least one time unit to claim its incoming assets. The formal proof is similar (in fact, easier) to that for reuniclus graphs and omitted. (See the full version.)

3.2 Protocol RDP for Reuniclus Digraphs

Reuniclus digraphs. A strongly connected digraph G is called a *reuniclus digraph* if there are vertices $b_1, b_2, ..., b_p \in G$, induced subgraphs $G_1, G_2, ..., G_p$ of G, and a rooted tree $\mathcal{K}$ whose nodes are $b_1, b_2, ..., b_p$, such that: (rg1) Each digraph G_j is a bottleneck subgraph, with b_j being its bottleneck vertex. We call G_j a *bottleneck component* of G and the *home component* of b_j. (rg2) If $i \neq j$, then $G_i \cap G_j = \{b_j\}$ if b_i is the parent of b_j in $\mathcal{K}$, and $G_i \cap G_j = \emptyset$ otherwise.

We refer to $\mathcal{K}$ as the *control tree* of G. (See Figure 2 for an example.) We extend the tree terminology to relations between bottleneck components, or between bottleneck vertices and components, in a natural fashion. Intuitively, a reuniclus graph G can be divided into bottleneck components. Overlaps are allowed only between two components if one is the parent of the other in the control tree $\mathcal{K}$, in which case the overlap is just a single vertex that is the bottleneck of the child component.

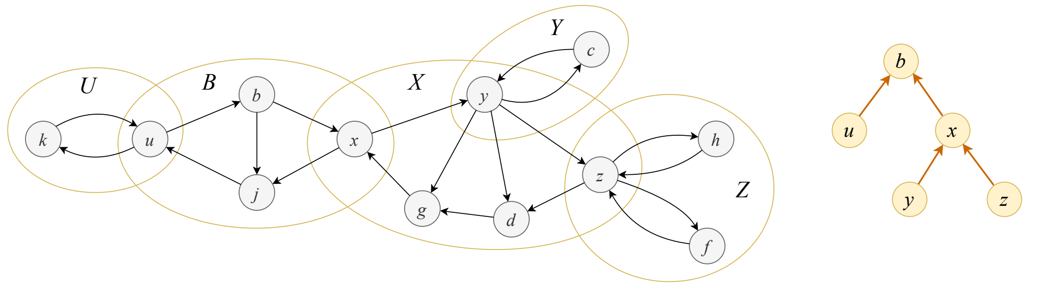

Figure 2 An example of a reuniclus graph (left) and its control tree (right). The bottleneck components (circled) are B, U, X, Y and Z. Their designated bottleneck vertices are b, u, x, y and z.

From the definition, the set of all bottleneck vertices in G forms a feedback vertex set of G. These bottleneck vertices are articulation vertices of G. Each bottleneck component may consist of several biconnected components that share the same bottleneck vertex.

Protocol RDP. Protocol RDP can be thought of as a hierarchical extension of Protocol BDP. Each bottleneck vertex b_j is called the leader of G_j. It creates a secret/hashlock pair (s_j, h_j), and its hashlock h_j is used to transfer assets within G_j, while the transfer of assets in the descendant components of b_j is "delegated" to the children of b_j in $\mathcal{K}$. We assume that the root component of $\mathcal{K}$ is G_1, and its bottleneck b_1 is called the *main leader*, denoted also by ℓ. All non-leader vertices are called *followers*.

Protocol RDP has two phases: contract creation and asset claiming. In the first phase, at time 0 all leaders create the outgoing contracts within their home bottleneck components. Then the contracts are propagated within the bottleneck components, to some degree independently; except that each leader b_j creates its outgoing contracts in its parent component G_i only after all its incoming contracts, *both in G_i and G_j*, are created. This ensures that at that time all contracts in its descendant components will be already created.

Protocol RDP for a leader $b_j \in G_i \cap G_j$:

- *At time* 0*:* Generate secret s_j and compute $h_j = H(s_j)$. For each arc (b_j, v) in G_j, create contract with hashlock h_j and timeout $\tau_{b_j v} = B^* + D_v^+$.
- *At time* $B_{b_j}^-$*:* For each arc (b_j, v) in G_i create its contract with hashlock h and timeout $\tau_{b_j v} = B^* + D_v^+$.
- *At time* $B^* + D_{b_j}^+$*:* Claim all incoming assets, using secret s in G_i and secret s_j in G_j.

Figure 3 Protocol RDP for a sub-leader b_j, namely the bottleneck vertex of G_j that also belongs to its parent component G_i. Recall that B_u^- denotes the maximum distance from some leader to u along a path that satisfies conditions (i)-(iii), and that $B^* = B_\ell^-$. D_v^+ is the maximum length of a simple path from v to ℓ. We assume that b_j aborts when it detects any deviation from the protocol.

In the asset claiming phase, the main leader ℓ is the first to claim the incoming contracts. The behavior of followers is the same as in Protocol BDP: they claim the incoming assets one step after all their outgoing assets were claimed. The behavior of all non-main leaders is more subtle. Each such sub-leader b_j waits until all its outgoing assets *in the parent component* are claimed, and then it claims all of its incoming assets.

The full protocol for non-main leaders b_j is given in Figure 3. Figure 4 shows timeout values for the reuniclus graph in Figure 2. In what follows we explain some notations used in the protocol.

As before, we use notation D_y^+ for the maximum distance from y to ℓ in G. We also need the concept analogous to the maximum distance from a leader, but this one is a little more subtle than for bottleneck graphs, because we now need to consider paths whose initial bottleneck vertex can be repeated once on the path. Formally, if $v \in G_i$, then B_v^- denotes the maximum length of a path with the following properties: (i) it starts at some leader b_j that is a descendant of b_i (possibly $b_j = b_i$), (ii) it ends in v, and (iii) it does not repeat any vertices, with one possible exception: it can only revisit b_j, and if it does, it either ends or leaves G_j (and continues in the parent component of b_j). (This can be interpreted as a maximum path length in a DAG obtained by splitting each leader into two vertices, one with the outgoing arcs into its home component and the other with all other arcs.) For example, in the graph in Figure 2, one allowed path for $v = u$ is $x - y - z - d - g - x - j - u$.

These values can be computed using auxiliary values B_{uv}^- defined for each edge (u, v). Call an edge (u, v) a *bottleneck edge* if u is a bottleneck vertex, say $u = b_j$, and $v \in G_j$. That is, bottleneck edges are the edges from bottleneck vertices that go into their home components. First, for each bottleneck edge (b_j, v) let $B_{b_j v}^- = 0$. Then, for each vertex u and each non-bottleneck edge (u, v) let $B_u^- = B_{uv}^- = \max_{(x,u)} B_{xu}^- + 1$, where the maximum is over all edges (x, u) entering u. By B^* we denote the value of B_ℓ^-.

The values B_z^- determine contract creation times. As shown in Figure 3, each leader b_j creates its contracts in its home component at time 0. Each other contract (u, v) will be created at time B_u^-. The last contract will be created by some in-neighbor of ℓ at time step $B^* - 1$. Then ℓ will initiate the contract claiming phase at time B^*. Analogous to Protocol BDP, each party u will claim its incoming contracts at time $B^* + D_u^+$, which is its timeout value.

▶ **Theorem 4.** *If G is a reuniclus digraph, then Protocol RDP is an atomic swap protocol for G.*

Proof. According to Lemma 2, it is sufficient to prove only the liveness and safety properties.

Liveness. The liveness property is quite straightforward. Each party $u \neq \ell$ has exactly one time unit, after its last incoming contract is created, to create its outgoing contracts. This will complete the contract creation at time $B^* - 1$. Thus at time B^* leader ℓ can claim

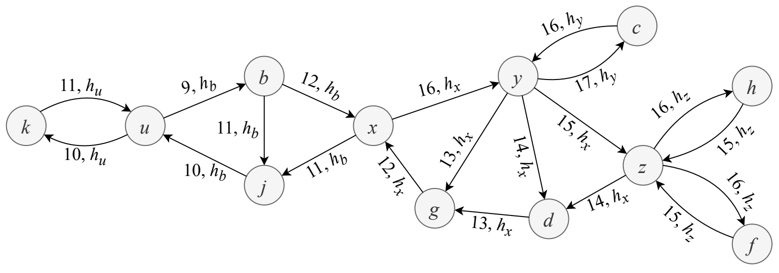

Figure 4 Timeout values and hashlocks for Protocol RDP for the graph in Figure 2. The main leader is $\ell = b$. We have $B^* = 9$ (this is the length of path $y - c - y - z - d - g - x - j - u - b$).

its incoming assets. For any other party u, each incoming asset (x, u) of u has timeout $\tau_{xu} = B^* + D_u^+$. If u is a follower then all the outgoing assets of u will be claimed before time τ_{xu}, and if u is a non-main leader then all of u's outgoing assets in its parent component will be claimed before time τ_{xu}. So u can claim all incoming assets at time τ_{xu}.

Safety. The proof of the safety condition for the main leader ℓ and pure followers is the same as in Protocol BDP for bottleneck digraphs. So here we focus only on non-main leaders.

Let b_j be a non-main leader whose parent component is G_i. Assume that b_j follows the protocol. So, according to Protocol RDP, b_j will create its outgoing contracts in G_j at time 0. Before creating its outgoing contracts in G_i, b_j checks if *all* incoming contracts are created. If any of its incoming contracts is not created or not valid, b_j will abort without creating its outgoing contracts in G_i. Thus its outcome will be NoDeal.

We can thus assume that all incoming contracts of b_j are created and correct; in particular all incoming contracts in G_j have hashlock h_j and all incoming contracts in G_i have the same hashlock h (which may or may not be equal to h_i). Then b_j creates its outgoing contracts in G_i, as in the protocol. We now need to argue that if any of b_j's outgoing assets is successfully claimed then b_j successfully claims *all* its incoming assets.

Suppose that some outgoing asset of b_j, say (b_j, w) is successfully claimed by w. Two cases arise, depending on whether w is in G_i or G_j.

If $w \in G_i$ then from contract (b_j, w) will provide b_j with some secret value s for which $H(s) = h$, because b_j used h for its outgoing contracts in G_i. At this point, b_j has both secret values s and s_j, and the timeout of all incoming contracts of b_j is greater than the timeout of (b_j, w). Therefore b_j has the correct secrets and at least one time unit to claim all incoming contracts, and its outcome will be Deal or Discount, thus acceptable.

In the second case, $w \in G_j$, the home component of b_j. For w to successfully claim (b_j, w), it must have the value of s_j. But, as b_j follows the protocol, it releases s_j only when claiming all incoming assets. So at this time b_j already has all incoming assets. Therefore in this case the outcome of b_j is also either Deal or Discount. ◀

4 A Characterization of Digraphs that Admit HTLC-Based Protocols

This section proves Theorem 1. By straightforward inspection, Protocol RDP from Section 3.2 is HTLC-based: each party creates at most one secret/hashlock pair, and all contracts are transferred using HTLC's. This already proves the ($\Leftarrow$) implication in Theorem 1.

It remains to prove the ($\Rightarrow$) implication, namely that the existence of an atomic HTLC-based protocol implies the reuniclus property of the underlying graph. We divide the proof into two parts. First, in Section 4.1 we establish some basic properties of HTLC-based protocols. Using these properties, we then wrap up the proof of the ($\Rightarrow$) implication in Section 4.2.

4.1 Basic Properties of HTLC-Based Protocols

Let $\mathbb{P}$ be an HTLC-based protocol for a strongly connected digraph G, and for the rest of this section assume that $\mathbb{P}$ is atomic. We now establish some fundamental properties that must be satisfied by $\mathbb{P}$.

Most of the proofs of protocol properties given below use the same fundamental approach, based on an argument by contradiction: we show that if $\mathbb{P}$ did not satisfy the given property then there would exist a (non-conforming) execution of $\mathbb{P}$ in which some parties, by deviating from $\mathbb{P}$, would force a final outcome of some conforming party to be unacceptable, thus violating the safety property.

For illustration, we include the proofs for Lemma 5 and for some other theorems and corollaries later in the section. See the full version [5] for the missing proofs.

▶ **Lemma 5.** *If some party successfully claims an incoming asset at some time t, then all contracts in the whole graph must be placed before time t.*

Proof. Assume that a party v successfully claims an asset (u, v) at time t. Towards contradiction, suppose that there is some arc (x, y) for which the contract is still not placed at time t. Since G is strongly connected, there is a path $y = u_1, u_2, ..., u_p = u$ from y to u in G. Let also $u_0 = x$ and $u_{p+1} = v$. Now consider an execution of $\mathbb{P}$ in which all parties except x are conforming, x follows $\mathbb{P}$ up to time $t - 1$, but later it never creates contract (x, y). This execution is indistinguishable from the conforming execution up until time $t - 1$, so at time t node v will claim contract (u, v). Since the first asset on path $u_0, u_1, ..., u_{p+1}$ is not transferred and the last one is, there will be a party u_j on this path, with $1 \le j \le p$, for which asset (u_{j-1}, u_j) is not transferred but (u_j, u_{j+1}) is. But then the outcome of u_j is unacceptable even though u_j is honest, contradicting the safety property of $\mathbb{P}$. ◀

Lemma 5 is important: it implies that $\mathbb{P}$ must consist of two phases: the *contract creation* phase, in which all parties place their outgoing contracts (by the liveness property, all contracts must be created), followed by the *asset claiming* phase, when the parties claim their incoming assets.

▶ **Lemma 6.** *Suppose that at a time t a party v creates an outgoing contract protected by a party different than v. Then all v's incoming contracts must be created before time t.*

Consider now the snapshot of of $\mathbb{P}$ right after the contract creation phase, when all contracts are already in place but none of the assets are yet claimed. Lemma 6 implies the following:

▶ **Corollary 7.**

(a) *If on some path each contract except possibly the first is not protected by its seller, then along this path the contract creation times strictly increase.*

(b) *Each cycle must contain a contract protected by its seller.*

Next, we establish some local properties of $\mathbb{P}$; in particular we will show that for each party v there is at most one other party that protects contracts involving v.

▶ **Lemma 8.** *If a party v has an incoming contract protected by some party x different from v then:*

(a) *Party v has at least one outgoing contract protected by x;*

(b) *All contracts involving v are protected either by v or by x.*

▶ **Lemma 9.** *Let $P = u_1, u_2, ..., u_k$ be a simple path whose last contract is protected by some party $z \notin \{u_1, u_2, ..., u_{k-1}\}$. Then for each $i = 1, ..., k-1$, contract (u_i, u_{i+1}) is protected by one of the parties $u_{i+1}, u_{i+2}, ..., u_{k-1}, z$. Consequently, each contract on P is not protected by its seller.*

▶ **Lemma 10.** *If all incoming contracts of a party v are protected by v then all outgoing contracts of v are also protected by v.*

Intuitively, if v had an outgoing contract protected by some other party x but not an incoming contract protected by x, then this outgoing contract would be "redundant" for v, since v does not need the secret from this contract to claim an incoming contracts. The lemma shows that the issue is not just redundancy – this is in fact not even possible if the protocol is atomic.

▶ **Lemma 11.** *If a party v has an outgoing contract protected by some party x different from v then it has an incoming contract protected by x.*

▶ **Lemma 12.** *A party v has an incoming contract protected by v if and only if it has an outgoing contract protected by v.*

The theorem below summarizes the local properties of the contracts involving a party v.

▶ **Theorem 13.** *Consider the contracts involving a party v, both incoming and outgoing.*

(a) *For each party x (which may or may not be v), v has an incoming contract protected by x if and only if v has an outgoing contract protected by x.*

(b) *If there are any contracts protected by v, then at least one incoming contract protected by v has a smaller timeout than all outgoing contracts protected by v.*

(c) *There is at most one party $x \neq v$ that protects a contract involving v. For this x, all timeouts of the outgoing contracts protected by x are smaller than all timeouts of the incoming contracts (no matter what party protects them).*

Proof.

(a) This part is just a restatement of the properties established earlier. For $x = v$, the statement is the same as in Lemma 12. For $x \neq v$, if v has an incoming contract protected by x then, by Lemma 8, it must have an outgoing contract protected by x, and if v has an outgoing contract protected by x then, by Lemma 11, it must have an incoming contract protected by x.

(b) Let (v, w) be an outgoing contract protected by v whose timeout value τ_{vw} is smallest. Consider any path P from w to v with all contracts on P protected by v. (This path must exist. To see this, starting from w follow contracts protected by v. By Corollary 7(b), eventually this process must end at v.) Then part (b) of Theorem 13 implies that along this path timeout values must decrease, so its last contract (u, v) must satisfy $\tau_{uv} < \tau_{vw}$. Thus, by the choice of w, the timeout value of (u, v) is smaller than all timeout values of the outgoing contracts protected by v.

(c) Let $x \neq v$. If v has an incoming contract protected by x then, by Lemma 8, all contracts involving v but not protected by v are protected by x. If v has an outgoing contract protected by x then Lemma 11 implies that some incoming contract is protected by x.

We now consider the claims about the timeout values. Let (v, w) be an outgoing contract protected by x, and let (u, v) be an incoming contract. Towards contradiction, suppose that in $\mathbb{P}$ the timeouts of these contracts satisfy $\tau_{uv} \le \tau_{vw}$. Denote by t^* the first step of $\mathbb{P}$ after the contract creation phase. We consider two cases, depending on whether (u, v) is protected by x or v.

Case 1: contract (u, v) is protected by x. We consider an execution of $\mathbb{P}$ where all parties follow $\mathbb{P}$ until time $t^* - 1$. Then, starting at time t^*, we alter the behavior of some parties, as follows: all parties other than v, w and x will abort the protocol, x will provide its secret s_x to w, and w will claim asset (v, w) at time τ_{vw}. This way, the earliest v can claim asset (u, v) is at time $\tau_{vw} + 1$, which is after timeout τ_{uv}. Thus v ends up in an unacceptable outcome, giving us a contradiction, which completes the proof of (b).

Case 2: contract (u, v) is protected by v. We can assume that its timeout τ_{uv} is minimum among all incoming contracts protected by v. (Otherwise, in the argument below replace (u, v) by the incoming contract protected by v that has minimum timeout.) Let (v, y) be any outgoing contract protected by v. From part (b), we have that $\tau_{uv} < \tau_{vy}$. Let also (z, v) be any incoming contract protected by x.

We now consider an execution of $\mathbb{P}$ where all parties follow the protocol until time $t^* - 1$. At time t, all parties other than u, w, x, y, z abort the protocol, and x gives its secret s_x to w. As time proceeds, v may notice that some parties do not follow the protocol, so, even though v is honest, from this time on it is not required to follow the protocol. We show that independently of v's behavior, it can end up in an unacceptable outcome, contradicting the safety property of $\mathbb{P}$.

To this end, we consider two possibilities. If v does not claim (u, v) at or before time τ_{uv}, then w can claim (v, w) at time τ_{vw}, so v will lose asset (v, w) without getting asset (u, v). On the other hand, if v claims (u, v), then u can give secret s_v to y who can then claim asset (v, y), and w will not claim asset (v, w), so v will not be able to claim asset (z, v), as it will not have secret s_x. In both cases, the outcome of v is unacceptable. ◀

We now use the above properties to establish some global properties of $\mathbb{P}$. The first corollary extends Corollary 7, and is a direct consequence of Theorem 13(b).

▶ **Corollary 14.** *If on some path each contract except possibly the first is not protected by the seller, then along this path the timeout values strictly decrease.*

The next corollary follows from Corollaries 7 and 14.

▶ **Corollary 15.** *Let P be a path such that all contracts on P except possibly the first are protected by a party x that is not an internal vertex of P. Then all contract creation times along P strictly increase and all timeout values strictly decrease.*

▶ **Corollary 16.**

(a) *Let (u, v) be a contract protected by some party x other than v. Consider a path P starting with arc (u, v), that doesn't contain x as an internal vertex and on which each contract is not protected by its seller. Then all contracts along P are protected by x.*

(b) *Let (u, v) be a contract protected by some party x other than u. Consider a path P ending with arc (u, v), that doesn't contain x as an internal vertex and on which each contract is not protected by its buyer. Then all contracts along P are protected by x.*

Proof.

(a) The corollary follows easily by repeated application of Theorem 13. Let (v, w) be the second arc on P. By the assumption, (v, w) is not protected by v, and since v has an incoming contract protected by x and $x \neq v$, Theorem 13 implies that contract (v, w) must be also protected by x. If $w = x$, this must be the end of P. If $w \neq x$, then w has an incoming contract protected by x, so we can repeat the same argument for w, and so on. This implies part (a).

(b) The proof for this part is symmetric to that for part (a), with the only difference being that we proceed now backwards from u along P. ◀

▶ **Theorem 17.**

(a) *Let P be a simple path starting at a vertex x whose last contract is protected by x. Then all contracts on P are protected by x.*

(b) *Let Q be a simple path ending at a vertex x whose first contract is protected by x. Then all contracts on Q are protected by x.*

Proof. (a) Let $P = u_1, u_2, ..., u_{p+1}$, where $u_1 = x$ and (u_p, u_{p+1}) is protected by x. The proof is by contradiction. Suppose that P violates part (a), namely it contains a contract not protected by x. (In particular, this means that $p \geq 2$.) We can assume that among all simple paths that violate property (a), P is shortest. (Otherwise replace P in the argument below by a shortest violating path.) Then (u_{p-1}, u_p) is not protected by x, because otherwise the prefix of P from x to u_p would be a violating path shorter than P. So (u_{p-1}, u_p) is protected by u_p. Using Lemma 9, each contract on the path $u_1, u_2, ..., u_p$ is not protected by the seller. Since (u_p, u_{p+1}) is protected by x and $x \neq u_p$, each contract on P is not protected by its seller.

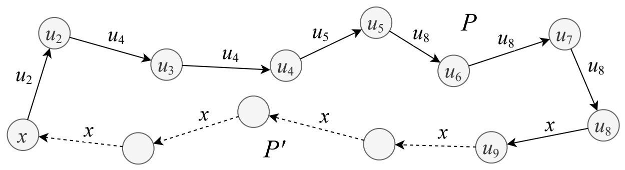

Figure 5 Illustration of the proof of Theorem 17(a). Path P is marked with solid arrows, path P' is marked with dashed arrows. Here, $p = 8$ and $u_1 = x$. The labels on edges show the parties that protect them.

Next, we claim that there is a simple path P' from u_{p+1} to x whose all contracts are protected by x. If $u_{p+1} = x$, this is trivial, so assume that $u_{p+1} \neq x$. Then, since u_{p+1} has an incoming contract protected by x and $x \neq u_{p+1}$, u_{p+1} must have an outgoing contract (u_{p+1}, w) protected by x. If $w = x$, we are done. Else, we repeat the process for w, and so on. Eventually, extending this path we must end at x, for otherwise we would have a cycle with all contracts protected by x but not containing x, contradicting Corollary 7(b). This proves that such path P' exists. Since all contracts on P' are protected by x, they are not protected by their sellers.

Finally, let C be the cycle obtained by combining paths P and P'. (See Figure 5.) Then every contract on C is not protected by its seller, contradicting Corollary 7, completing the proof. ◀

4.2 Proof of the ($\Leftarrow$) Implication in Theorem 1

In this section we use the protocol properties established in the previous section to prove necessary conditions for a digraph to admit an atomic HTLC protocol. We start with protocols that use only one common hashlock for the whole graph. (See the full version for the missing proofs.)

▶ **Lemma 18.** *Suppose that G has an atomic HTLC protocol $\mathbb{P}$ in which only one party creates a secret/hashlock pair. Then G must be a bottleneck graph.*

We now consider the general case, when all parties are allowed to create secret/hashlock pairs. The lemma below establishes the ($\Rightarrow$) implication in Theorem 1.

▶ **Lemma 19.** *Suppose that G has an atomic HTLC protocol $\mathbb{P}$. Then G must be a reuniclus digraph.*

Proof. Recall what we have established so far in Section 4.1. From Theorem 13(c) we know that, for each party u, all contracts involving u are protected either by u or by just one other party. Using this property, we define the control relation on parties, as follows: If u has any contracts protected by some other party x, we will say that x *controls* u. Let $\mathcal{K}$ be a digraph whose vertices are the parties that created secret/hashlock pairs, and each arc (u, x) represents the control relation, meaning that x controls u. We want to prove that $\mathcal{K}$ is a tree.

Each node in $\mathcal{K}$ has at most one outgoing arc. This property already implies that each connected component of $\mathcal{K}$ is a so-called *1-tree*, namely a graph that has at most one cycle. So in order to show that $\mathcal{K}$ is actually a tree, it is sufficient to show the two claims below.

▷ Claim 20. $\mathcal{K}$ does not have any cycles.

We now prove Claim 20. Towards contradiction, suppose that $\mathcal{K}$ has a cycle $C = v_1, v_2, ..., v_k, v_{k+1}$, where $v_{k+1} = v_1$. Consider any arc (v_i, v_{i+1}) on C. This arc represents that v_{i+1} controls v_i. So, in G, v_i has an outgoing contract protected by v_{i+1}. Let P_i be any path in G starting with this contract and ending at v_{i+1}. Then Theorem 17(b) gives us that all contracts on P_i are protected by v_{i+1}. Combining these paths $P_1, ..., P_k$ we obtain a cycle C' in G. Then in C', each contract is not protected by its seller, which would contract Corollary 7(b). This completes the proof of Claim 20.

▷ Claim 21. $\mathcal{K}$ has only one tree.

From Claim 20 we know that each (weakly) connected component of $\mathcal{K}$ is a tree. The roots of these trees have the property that their contracts are not protected by any other party. To prove Claim 21, suppose towards contradiction that $\mathcal{K}$ has two different trees, and denote by r and r' the roots of these trees. Since r, r' are roots of trees, all contracts involving r are protected by r and all contracts involving r' are protected by r'. Consider any simple path $P = u_1, u_2, ..., u_k$ from $u_1 = r$ to $r' = u_k$. Since the last contract on P is protected by r' and r' is not in $\{u_1, u_2, ..., u_{k-1}\}$, Lemma 9 implies that all contracts on this path are not protected by their sellers. But this contradicts the fact that (u_1, u_2) is protected by u_1. This completes the proof of Claim 21.

We now continue with the proof of the theorem. Denote by $b_1, b_2, ..., b_p$ the nodes of $\mathcal{K}$. For each b_j, define G_j to be the subgraph induced by the contracts protected by b_j. That is, for each contract (u, v) protected by b_j we add vertices u, v and arc (u, v) to G_j. The necessary properties (rg1) and (rg2) follow from our results in Section 4.1. It remains to show that subgraphs $G_1, G_2, ..., G_p$, together with tree $\mathcal{K}$, satisfy conditions (rg1) and (rg2) that characterize reuniclus graphs.

Consider some $u \neq b_j$ that is in G_j. By the definition of G_j, u is involved in a contract protected by b_j. Then Theorem 13 gives us that u has both an incoming and outgoing contract protected by b_j. Take any path P starting at an outgoing contract of u protected by b_j and ending at b_j. Then Theorem 17(b) implies that all contracts on P are protected by b_j. By the same reasoning, there is a path P' starting at b_j and ending with an incoming contract of u protected by b_j. Then, by Theorem 17(a) all contracts on P' are protected by b_j. This gives us that G_j is strongly connected. And, by Corollary 7, G_j cannot contain a cycle not including b_j. Therefore G_j is a bottleneck graph with b_j as its bottleneck.

We also need to prove that G_j is in fact an induced subgraph, that is, if $u, v \in G_j$ and G has an arc (u, v), then $(u, v) \in G_j$ as well. That is, we need to prove that (u, v) is protected by b_j. Suppose, towards, contradiction, that (u, v) is protected by some $b_i \neq b_j$. Then both

u and v are involved in contracts protected by both b_i and b_j, and this implies that $u = b_i$ and $v = b_j$, or vice versa. And this further implies that b_i would be protected by b_j and vice versa, which would be a cycle in $\mathcal{K}$, contradicting that $\mathcal{K}$ is a tree. This completes the proof of property (rg1).

Finally, consider property (rg2). If a vertex u is not any of designated bottleneck vertices $b_1, b_2, ..., b_p$, then, by Theorem 13, all its contracts are protected by the same party, which means that it belongs to exactly one graph G_j. On the other hand, if $u = b_j$, then again by Theorem 13, it is involved only in contracts protected by itself and one other party, say b_i. But then it belongs only to G_j and G_i, completing the proof of (rg2). ◀

5 Final Comments

We have provided a full characterization of digraphs for which there exist atomic HTLC-based protocols for asset swaps, by proving that these digraphs are exactly the class of reuniclus graphs. Our work raises several natural open problems and leads to new research directions, including the following:

- One natural extension of HTLCs would be to allow multiple hashlocks in a contract. We can show that there are non-reuniclus graphs that have atomic protocols based on such contracts, and that there are strongly connected digraphs that don't, although at this time we do not have a full characterization of digraphs that admit such protocols.
  A further extension would be to allow different hashlocks in the same contract have different timeouts. What types of graphs can be handled with such protocols?
  More generally, are there any simple generalizations of HTLCs that lead to atomic protocols for arbitrary graphs?
- Herlihy's method requires computing longest paths in graphs in order to specify the protocol's steps on a given digraph. Are longest paths truly necessary to assure the safety property in arbitrary graphs? If so, it would be interesting to prove this, say via some sort of computational-hardness result.

References

1 N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 7–17, New York, NY, USA, 1997. ACM. doi:10.1145/266420.266426.

2 N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18:593–610, 1997.

3 M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Trans. Inf. Theor.*, 36(1):40–46, September 2006.

4 Eric Chan, Marek Chrobak, and Mohsen Lesani. Cross-chain swaps with preferences. In *36th IEEE Computer Security Foundations Symposium, CSF 2023, Dubrovnik, Croatia, July 10-14, 2023*, pages 261–275. IEEE, 2023. doi:10.1109/CSF57540.2023.00031.

5 Emily Clark, Chloe Georgiou, Katelyn Poon, and Marek Chrobak. On htlc-based protocols for multi-party cross-chain swaps. *CoRR*, abs/2403.03906, 2024. arXiv:2403.03906, doi:10.48550/arXiv.2403.03906.

6 Apoorvaa Deshpande and Maurice Herlihy. Privacy-preserving cross-chain atomic swaps. In *International Conference on Financial Cryptography and Data Security*, pages 540–549. Springer, 2020. doi:10.1007/978-3-030-54455-3_38.

7 Matt Franklin and Gene Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In Rafael Hirchfeld, editor, *Financial Cryptography*, pages 90–102, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

8 Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. The Arwen trading protocols. In *International Conference on Financial Cryptography and Data Security*, pages 156–173. Springer, 2020. doi:10.1007/978-3-030-51280-4_10.

9 Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 245–254, New York, NY, USA, 2018. ACM. URL: https://dl.acm.org/citation.cfm?id=3212736.

10 Maurice Herlihy, Barbara Liskov, and Liuba Shrira. Cross-chain deals and adversarial commerce. *arXiv preprint arXiv:1905.09743*, 2019. arXiv:1905.09743.

11 Soichiro Imoto, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Atomic cross-chain swaps with improved space and local time complexity, 2019. arXiv:1905.09985.

12 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1916–1933, 2018. doi:10.1137/1.9781611975031.125.

13 Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL: https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/.

14 Subhra Mazumdar. Towards faster settlement in htlc-based cross-chain atomic swaps, 2022. arXiv:2211.15804, doi:10.48550/arXiv.2211.15804.

15 Silvio Micali. Simple and fast optimistic protocols for btcwiki electronic exchange. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 12–19, New York, NY, USA, 2003. ACM.

16 Wallstreetmojo Team. Hashed timelock contract. https://www.wallstreetmojo.com/hashed-timelock-contract/, 2024.

17 Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sánchez. Universal atomic swaps: Secure exchange of coins across all blockchains. *Cryptology ePrint Archive*, 2021.

18 Tier Nolan. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949, 2013. [Online; accessed 23-January-2021].

19 Aisshwarya Tiwari. An introductory guide to hashed timelock contracts. https://crypto.news/learn/an-introductory-guide-to-hashed-timelock-contracts/, 2022.

20 Yingjie Xue and Maurice Herlihy. Hedging against sore loser attacks in cross-chain transactions. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 155–164, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467904.