Be My Guest: Welcoming Interoperability into IBC-Incompatible Blockchains

Michał Nazarewicz*, Dhruv D. Jain†, Miguel Matos‡, Blas Rodriguez†
*No Affiliation, †Inclusive Layer, ‡IST Lisbon & INESC-ID

***Abstract*—The rise of cryptocurrencies has led to the creation of numerous isolated blockchains. A limitation of the space is the absence of seamless interoperability, which has hindered users' ability to interact across different chains. To tackle this challenge, the Inter-Blockchain Communication (IBC) protocol has emerged as one possible trustless solution. Unfortunately, some blockchains (for example Solana, NEAR, and TRON) do not meet IBC's technical criteria, preventing their integration with the wider IBC ecosystem. This paper introduces the concept of a *guest blockchain* which runs on top of an unsupported blockchain and provides all the features necessary for IBC integration. We demonstrate our approach by deploying it on top of Solana which is currently live in the main network and enables IBC-based communication with the Solana blockchain with performance comparable to native IBC-blockchains.**

I. Introduction and Related Work

Since Bitcoin's inception [1], blockchains have attracted the attention of the industry, academia and the general public, resulting in a wide-range of systems that cater to different use cases, business models, and security requirements [2]–[9]. While these specialized designs have contributed to the popularity of blockchain ecosystems, they've also led to fragmentation since most blockchains were not built with interoperability in mind.

As blockchain systems grow in value and users, the ability to perform cross-chain communication (i.e. blockchain interoperability) allows for the development of new use cases offering value to users including cross-chain governance, yield aggregation, decentralised lending. The demand for blockchain interoperability can be seen in the plethora of proposals aiming at connecting blockchain ecosystems [10]–[12].

Early approaches for cross-chain communication relied on a trusted third-party to pass packets between blockchains. A Trusted Relayer listens to changes made on one blockchain and sends matching transactions to another. The safety and liveness of this approach completely dependens on this centralised entity which runs counter to the common motivation for using blockchains as decentralised trustless systems.

Trustless bridges aim to remove the need for trusted third parties. The Inter-Blockchain Communication (IBC) protocol [13] is an example of a technology enabling trustless bridging with 100+ interconnected blockchains and cross-chain transfers worth over 1.5 billion dollars each month[1]. It operates similarly to trusted bridges except that its relayers include cryptographic proofs for each executed transaction. Those proofs are independently verified on each blockchain, making it impossible to falsify packets.

To operate correctly, IBC imposes technical requirements on the target blockchains. Not all blockchains meet them, including widely used systems such as NEAR [3][2], Solana [4], and TRON [5]. Modifying those systems to support IBC is not trivial and can require substantial technical effort.

We introduce the concept of a *guest blockchain* for connecting to the IBC ecosystem blockchains that do not meet IBC's requirements. Unlike approaches that rely on trusted third-parties [15]–[17] or leverage blockchain-specific characteristics [6], [7], our approach is trustless and can be applied to all blockchains supporting smart contracts and on-chain storage.

Our key insight is to leverage existing functionalities, namely: the ability to run smart contracts to emulate a guest blockchain that provides the necessary functionalities to satisfy IBC requirements; and the communication mechanism defined by the IBC protocol to allow interoperability between blockchains. Because the guest blockchain runs on top of an existing blockchain (i.e. the host), it inherits its security, liveness, and trust properties. Overall, our results show that the guest blockchain has performance and costs comparable to other IBC approaches.

II. IBC Background

IBC is a stateful, connection-oriented protocol for reliable and authenticated communication between independent blockchains [13], [18]. It offers bi-directional channels with a relatively short delay—averaging one minute per packet between blockchains [19]. IBC defines the following elements:

- **Provable storage** a key-value store that can cryptographically prove the presence or absence of data to external verifiers, typically implemented as a Merkle trie.
- **Counterparty's light client** an on-chain component that validates block headers of the counterparty blockchain.
- **IBC module** handles the protocol logic and maintains the state necessary for packet exchange.
- **Runtime environment** supports transactional execution and mechanisms for sending/receiving IBC packets.

Communication between blockchains requires establishing a connection through a handshake that verifies the identit and status of each blockchain. This requires each blockchain to

[1]https://mapofzones.com/zones?period=30d, retrieved 2024-11-01

[2]While Octopus Network has launched IBC on NEAR [14], their implementation is incomplete and leaves `validate_self_client` method blank (see https://github.com/octopus-network/near-ibc/blob/2b30af42/near-ibc/src/ibc_impl/core/validation_context.rs#L145).

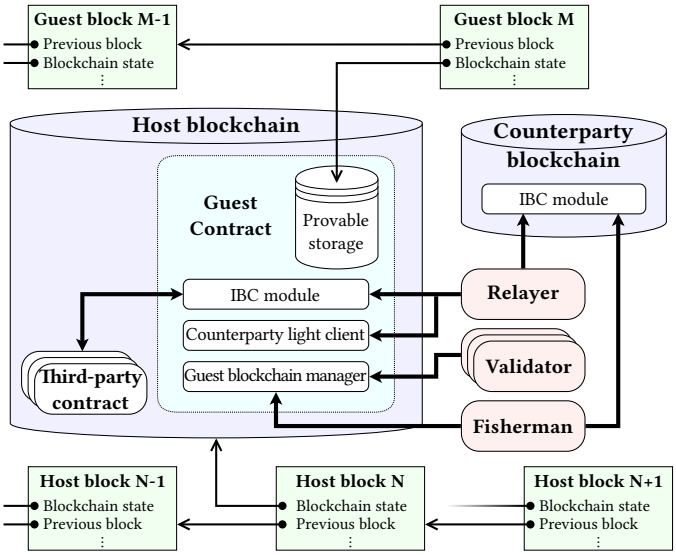

Fig. 1: The high-level architecture of a guest blockchain.

be able to introspect its own state, particularly recent block headers. Once connected, packet exchange takes the following steps: ① a smart contract generates a packet and stores its commitment in the source blockchain, ② a relayer notices the packet and forwards it with a proof to the destination blockchain, ③ the destination's IBC module verifies the proof and validity of the packet, ④ the destination verifies the proof and processes the packet, ⑤ a relayer forwards a receipt back to the source blockchain, ⑥ the source marks the packet as delivered and notifies the source smart contract.

Unfortunately not all blockchains meet IBC requirements. Solana and TRON lack state proofs [20], [21], while NEAR does not implement introspection [22]. While it is, in principle, possible to extend each system with the missing features, it is often complex and time consuming, and users of such blockchains may not have an option to wait for the necessary changes. Our guest blockchain approach addresses these limitations without modifying the host's blockchain.

III. The Guest Blockchain

The guest blockchain creates a virtual blockchain layer on top of the host blockchain to enable IBC compatibility. The host blockchain provides transaction atomicity and state persistence, while the guest layer adds the missing IBC features—provable storage, light client support, and block introspection.

Fig. 1 depicts the system architecture. The main logic is implemented in the *Guest Contract* (§III-A), a smart contract running on the host. It has two main roles: 1) generate blocks in the guest blockchain (the *guest blocks*) and 2) serve as a bridge between the host and guest blockchains. The Validators (§III-B) watch for new blocks produced by the Guest Contract, Relayers forward packets between the blockchains and Fishermen monitor Validator misbehaviour (§III-C).

A. Guest Contract

The Guest Contract maintains the guest blockchain state, including the list of guest blocks and the set of sent, delivered, and received IBC packets.

Provable Storage The guest blockchain uses a Merkle trie to keep its provable state. Guest blocks include the root hash of the trie which acts as a commitment for state membership and non-membership proofs [23]–[25]. To prevent double delivery, the Guest Contract has to remember all processed packets, i.e. as an IBC connection is used, more data is generated.

To address the unbounded space growth, we drew inspiration from Bitcoin's disk reclamation technique [1] and implemented a *sealable trie*. A sealable trie alters the Merkle trie by allowing nodes to be *sealed* — removing them from storage while preserving the trie's commitment. A sealed leaf node is removed from the underlying storage without changing its hash saved in the parent node. If all children of an internal node are sealed, that node is sealed as well.

Sealed nodes cannot be accessed so only values which are no longer needed may be sealed. For example, when a packet is delivered, the Guest Contract saves it in the trie and then seals its node. The next time the same packet is delivered, the Guest Contract is unable to access the sealed node which, as required, prevents double delivery.

With this design, the size Guest Contract's provable storage depends on the number of open channels and packets in flight only and does not grow over time.

Algorithm The Guest Contract's algorithm is depicted in Alg. 1. Note that IBC multiplexes streams of packets between two blockchains. Each stream, called a channel, is identified by a ⟨name, port⟩ pair [26]. For brevity we omit those details from the algorithm (e.g. `out_seq_num` is tracked per channel).

The sender submits a packet (step ①) via the `SendPacket` procedure. The Guest Contract collects fees, generates a packet with the next available sequence number, and stores its commitment in the guest blockchain's provable storage.

Before further processing, the packet is included in a new guest block first. The `GenerateBlock` procedure which can be invoked by anyone (e.g. whenever a host block is produced) handles that; it checks if: 1) the current head of the guest blockchain is finalised (i.e. a quorum of Validators have signed it) and 2) the state root has changed or the head is older than a predefined Δ. If both conditions are met, a new block is created and the Guest Contract emits a `NewBlock` event.

The Δ parameter is necessary to support IBC timeouts. The counterparty needs to observe a guest block to verify time seen by the guest blockchain. Δ guarantees a guest block with a new timestamp is always eventually generated.

Validators listen to the `NewBlock` event (§III-B) and submits the block signature via the `Sign` procedure. The Guest Contract checks the signature and adds the Validator to block's signers set if its an active validator who has not signed the block yet. Once quorum is reached, the block is finalised and a `FinalisedBlock` event emitted. This event, which correspond to step ②, is picked up by a Relayer (§III-C).

```
1  init
2      blocks ← ⟨genesis_block⟩                    // Guest blocks
3      out_seq_num ← 0                             // Number of packets sent
4      trie ← ∅                                    // Guest blockchain state
5      Δ ← ...                                     // Max age, system parameter
6  procedure SendPacket(destination, payload)
7      collect_fees(payload)
8      n ← out_seq_num
9      packet ← gen_packet(n, destination, payload)
10     out_seq_num ← n + 1
11     trie ← trie ∪ {hash(packet)}
12 procedure GenerateBlock()
13     head ← last(blocks)
14     assert head.finalised
15     assert head.trie ≠ trie ∨ age(head) ≥ Δ
16     block ← create_new_block(head, trie)
17     blocks ← blocks ∪ {block}
18     emit event NewBlock(block)
19 procedure Sign(height, pubkey, signature)
20     block ← blocks_height
21     assert block ≠ ⊥                            // Invalid height
22     assert pubkey ∈ block.epoch.validators
23     assert pubkey ∉ block.signers
24     assert check_signature(block, pubkey, signature)
25     block.signers ← block.signers ∪ {pubkey}
26     if ¬block.finalised then
27         votes ← Σ_{v∈block.signers} block.epoch.stake(v)
28         if votes ≥ block.epoch.quorum then
29             block.finalised ← True
30             emit event FinalisedBlock(block)
31     blocks_height ← block
32 procedure ReceivePacket(packet, height, proof)
33     block ← get_counterparty_block(height)
34     assert block ≠ ⊥          // Unknown counterparty block
35     ph ← hash(packet)
36     assert check_proof(ph ∈ block.state, proof)
37     assert ph ∉ trie
38     trie ← trie ∪ {ph}
39     deliver packet.payload to packet.destination
```

Alg. 1: Guest Contract

For incoming packets, the Relayer picks packets from the counterparty blockchain and calls the `ReceivePacket` procedure. It verifies the membership proof for the packet (step ④) and if the packet has not been received yet, it is added to the guest blockchain's provable storage and delivered to the target smart contract on the host chain (step ⑤).

B. Validators

Validators enable light clients and state proofs through a Proof-of-Stake mechanism for block finalisation [27]. They attest guest blocks were generated by the Guest Contract which allows counterparty blockchains to verify guest blocks and state proofs.

```
1  upon NewBlock(block)
2      signature ← sign(block, validator.pvtkey)
3      guest.Sign(block.height, validator.pubkey,
         signature)
4  upon FinalisedBlock(block)
5      if block.packets ≠ ∅ ∨ block.last_in_epoch then
6          counterparty.send_block(block)
7      foreach packet ∈ block.packets do
8          if packet ∉ relayer.outgoing_packets then
9              proof ← gen_proof(
                 hash(packet) ∈ block.state)
10             counterparty.deliver_packet(
                 packet, block.height, proof)
```

Alg. 2: Validators and Relayers

Validators are established for an epoch (a configurable number of blocks) and must stake assets with the Guest Contracts to become candidates. At the end of an epoch the contract selects the Validators with the most stake. Validators that misbehaves are slashed and removed from the set.

C. Relayer and Fishermen

Relayers poll events from and forward packets between blockchains. Since the guest blockchain provides a standard IBC interface, we reuse existing Relayer implementations. Through the state proofs, both blockchains can verify each other's state ensuring safety even if Relayers misbehave.

Fishermen monitor the guest blockchain for signs of misbehaviour. If they notice it they send proof to the Guest Contract which can be 1) a pair of signatures for different blocks with the same height, 2) a signature for a block with height larger than the blockchain's head, or 3) a signature for a block which differs from a known block at that height.

Relayers and Fishermen are both permissionless and can be run by anyone.

IV. Implementation and Deployment

We implemented and deployed the guest blockchain on Solana [4] and connected it with Picasso Network,[3] a Cosmos-based blockchain with native IBC support. We selected Solana because it is popular, offers good performance and low fees [28] but lacks IBC support.

Solana's performance comes at the cost of several constrain of the smart contract execution runtime such as: transaction size limit of 1232 bytes, compute time limit of 1.4 million compute units preventing implementing cryptographic functions in-contract, default memory allocator not supporting heap sizes over 32 KiB and lack of support for mutable global state [29], [30]. We implemented solutions for each of those restrictions [31]–[33] which further illustrates general applicability and the powerful abstraction our approach provides.

[3]https://picasso.network/

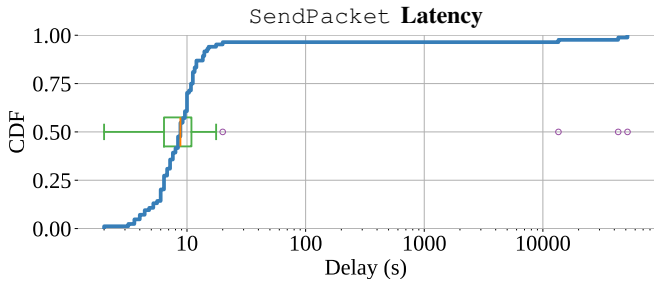

Fig. 2: Delay between sending a packet (`SendPacket` invocation) and time it is stored in a finalised guest block ready to be picked up by a Relayer (the `FinalisedBlock` event).

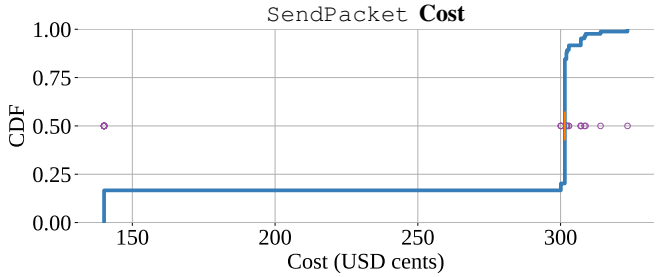

Fig. 3: Cost of sending a packet (`SendPacket` invocation).

The guest blockchain is operational on Solana's main network with the following configuration: $\Delta = 1\,\text{h}$, i.e. the minimum time between empty blocks; the minimum epoch length set to 100 thousand host blocks (roughly 12 hours); and Validator stake held for one week after exit.

V. Evaluation

We evaluated the guest blockchain deployed in Solana and connected to the Picasso network between September 1-29 2024. The guest blockchain had 24 Validators, with only one controlled by us for bootstrapping. The remaining Validators were operated by third parties who staked assets to participate in the system. The total stake value was 1.25 million dollars.

We focus the evaluation on the cost and latency from different perspectives. They are naturally influenced by the host blockchain and this angle, Solana is an ideal evaluation scenario since its sub-second block speed and low fees minimise the host's influence on the obtained results. Furthermore, we do not evaluate the cost or latency involved in calling the counterparty blockchain and restrict ourselves to assess the contributions of the guest blockchain to each aspect.

Fees on Solana are paid in SOL, Solana's native currency. To make values more intuitive, we present costs in US dollars assuming a SOL price of 200 USD which corresponds roughly to the highest value over the last 12 months[4]. All evaluation data are available at [34].

A. Client Perspective

We start by evaluating the guest blockchain from the perspective of a client of the system, i.e. a client smart contract sending/receiving packets to/from the counterparty blockchain.

[4]https://coinmarketcap.com/currencies/solana/ retrieved 2024-11-01

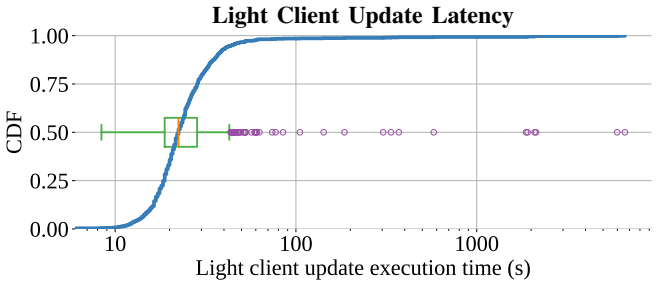

Fig. 4: Latency of the light client updates sent by the Relayer to, i.e. time between execution of the first and last Solana transaction comprising the light client update.

Sending a packet The time to send a packet captures the elapsed time between a user initiating an outgoing packet and the packet being included in a finalised block. This corresponds to the period from the invocation of the `SendPacket` procedure in the Guest Contract to the moment the `FinalisedBlock` event is emitted. Note that the total time observed by the client also includes the time the transaction spends in the host blockchain mempool, the time Relayer takes to deliver the packet to the counterparty blockchain. Those are outside our control and applies equally to all transactions submitted to the host.

The results are depicted in Fig. 2 and show that all but three transfers were completed within 21 seconds. The three stragglers were caused by delays from the Validators when signing the blocks. These operational issues are prone to happen in early stage deployments such as the guest blockchain but we expect them to be smoothened out as the system matures.

Fig. 3 shows the cost of sending a packet with a clear division into two clusters. The grouping corresponded to different fee policies: in 17% of the cases Solana priority fees [29] were used with cost of 1.40 USD; in the remaining cases block bundles were used with cost of about 3.02 USD.[5] In time we expect to settle on a single strategy which balances cost and latency (§VI-B).

Receiving a packet Receiving a packet is a multi-step process as described in §II, steps ③–⑤ (with the guest blockchain in role of the destination blockchain). Firstly, a Relayer updates the light client with the state of the counterparty blockchain (line 6 of Alg. 2). Due to the Solana limitations described in §IV, those operations are split into multiple Solana transactions which impacts costs and latency.

In particular, the light client update required on average 36.5 individual Solana transactions with standard deviation of 5.8. Still, as Fig. 4 shows, 50% of the updates took less than 25 seconds and 96% took less than a minute.

The final step corresponds to the `ReceivePacket` procedure where the Guest Contract records the incoming packet and delivers it to the destination on the host blockchain.

[5]Specifically, we have used Jito block bundles described in [35]. They offer alternative way of prioritising transaction execution.

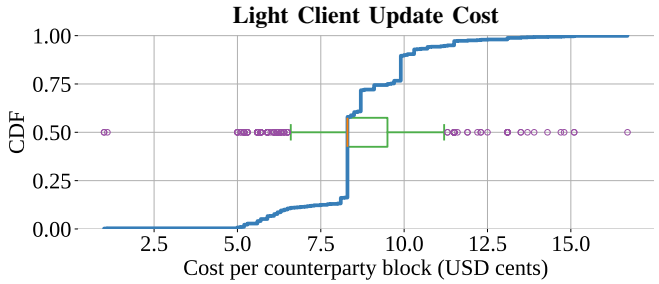

Fig. 5: Cost of the light client update by the Relayer, i.e. total cost of all the Solana transactions in the light client update.

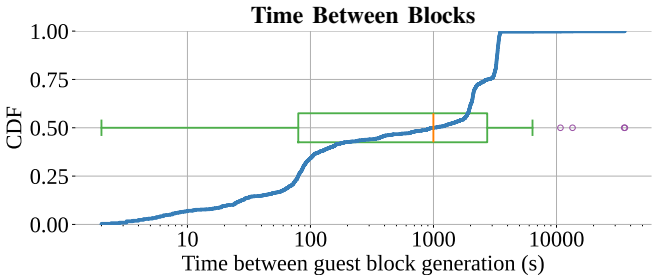

Fig. 6: Interval between generation time of two consecutive guest blocks.

Depending on the size of the packet this process required 4–5 Solana transactions. In all cases, all those transactions were included together in a single Solana block resulting in no latency introduced by the guest blockchain.

B. Relayers perspective

Relayers are responsible for: updating the light clients and transmitting packets between the blockchains. The Relayer operates by submitting transactions which correspond to those operations. Relayer's costs correlate with the number of transactions and signature checks required to execute given operation. During our month-long experiment we used the default Solana fee model costing us 0.1 cents per transaction and additional 0.1 cents per signature.

The costs for updating the light client are shown in Fig. 5. The variance in costs is due to variance in amount of data in a light client update and number of signatures checked when validating the update. As discussed above, `ReceivePacket` calls took 4–5 transactions; their cost was 0.4 cents in 98.2% of the cases and 0.5 cents in the remaining cases.

C. Validators perspective

Validators are essential to ensure the safety and liveness of the system. Their costs comprise the cost of the infrastructure and the cost of submitting the `Sign` transactions. The former is modest since their work is lightweight and does not require substantial computing resources (they only need to submit a single signature each time a new guest block is generated, see Alg. 2). The latter is paid on-chain and depends on the fee model used by the Validator.

During our one-month evaluation, each Validator used a fixed priority fee as shown in Table I together with latency statistics of each Validator (i.e. time between block generation and Validator submitting a signature). We have observed correlation coefficient of 0.007 between cost and latency indicating that Validators who used higher fees were likely overpaying and their costs could be lowered (§VI-B).

The table also illustrates the importance of incentives. Out of 24 Validators, 7 did not submit any signatures. As a result, when due to operator error Validator #1 stopped working, the guest block could not be finalised because remaining well-behaving Validators could not form a quorum. However, since automatic slashing and rewards was not implemented, those Validators kept their stake intact. We expect that with a full implementation of all the incentives, Validators will engage in the system since otherwise they would face loss of assets.

TABLE I: Validator Signing Statistics

	No. of sigs	Cost (cents)	Block signing latency (s) Min	Q1	Med.	Q3	Max	μ	σ
#1	1535	1.00	0.8	3.6	5.6	7.6	35957.6	77.4	1373.6
#2	977	1.40	0.4	2.0	3.2	5.2	10.4	3.6	1.9
#3	790	0.25	0.4	2.0	3.2	5.6	26.8	3.9	2.5
#4	622	1.40	1.2	3.2	4.0	6.0	19.2	4.5	2.3
#5	618	0.23	0.8	2.4	3.6	5.2	14.4	4.1	2.2
#6	603	0.23	1.2	2.4	3.6	5.2	20.0	4.1	2.3
#7	464	1.40	0.8	2.8	4.0	6.0	27.6	4.7	3.0
#8	442	0.60	2.0	3.6	4.8	6.4	17.6	5.3	2.4
#9	250	0.23	1.2	2.8	3.6	4.8	261.6	8.4	24.6
#10	209	0.23	1.2	2.4	3.2	5.2	9.6	3.9	1.9
#11	143	1.40	0.8	3.2	4.8	6.4	17.6	4.8	2.4
#12	118	1.40	1.2	2.8	3.6	5.6	16.0	4.4	2.6
#13	117	1.40	0.8	2.8	4.4	6.4	27.2	4.8	3.1
#14	109	1.40	1.2	3.2	4.4	6.0	16.0	4.6	2.3
#15	21	1.40	1.6	2.0	3.2	3.2	8.0	3.3	1.8
#16	41	0.20	0.8	2.4	3.2	4.4	10.8	3.7	2.0
#17	61	0.20	1.2	2.8	3.2	4.8	9.6	3.8	1.9

In the absence of pending changes, a new guest block is only generated after Δ time (§III) which in our deployment was set to 1 hour (§IV). Fig. 6 shows the time between two consecutive guest blocks. The distribution roughly follow the rate at which new packets are sent up to Δ where an empty block is generated. About a quarter of the guest blocks were generated at this cut-off suggesting they were empty. During the time we observed five blocks with time vastly over an hour which was caused by long Validator signing delays.

D. Storage Costs

The final cost is the price of storing the state on the host blockchain. We have allocated a 10 MiB for the guest blockchain's state (the largest possible account size on Solana [29]) which is sufficient to store over 72 thousand key-value pairs and thanks to the sealable trie approach (§III-A) it should be sufficient for our deployment in the long term.

Initialising such a large account required a deposit of 14.6 thousand dollars. While a significant amount, the assets can be recovered when the account is shrunk or deleted. Because of that we did not try to minimise the size right away.

VI. Limitations and Future Work

Our experimental evaluation allows us to conclude that the guest blockhain has low costs and good performance, in line with what was reported by previous studies on IBC-compatible blockchains [18], [19]. However, like any system, it is not without limitations. In this section we discuss aspects that implementers needs to be aware of which go beyond the scope of this paper and are the possible subject for future work.

A. Last Validator Wishing to Quit

In Proof-of-Stake blockchains, validators have to freeze assets while they participate in the blockchain. However, to operate blockchain must have a minimum set of validators. This means validators may be unable to recover their assets unless someone else is willing to take their place; if the blockchain stops, their assets become inaccessible forever.

In systems where the native token is staked, if the blockchain stops the native token loses its value and validators being unable to recover their assets is not a problem. In the guest blockchain however, Validators stake assets whose value is independent from it—even if the guest blockchain stops, the frozen assets retain their value. This could lead to a bank run where Validators quit validating out of fear of becoming the last Validator unable to withdraw their funds.

One possible mitigation is to extend the Guest Contract with self destruction functionality—once enough time passes since last guest block generation, the Guest Contract may releases all assets to the remaining Validators. A non-technical alternative is to sign contracts with organisations running the Validators however that introduces potential trust concerns.

B. Fee optimisation

Many blockchains, including Solana, allow for prioritising transactions through a fee which encourages block producers to pick transactions with higher rewards [29]. Alternatively, users can tip block producers directly to include a transaction in a block [35]. The current implementation uses fixed fee models which often results in good latency but is inflexible.

During low host chain usage the costs may be reduced and during high usage the fees do not prevent long tail latency (§V-A). Further research is necessary to dynamically adjust the fees according to the demand on the host blockchain.

C. Security

The security of cross-chain applications remains a critical area of research, given the millions of dollars at stake [11], [36], [37]. IBC security relies on the safety of the involved blockchains (but not on the Relayer). If any of the blockchains gets compromises, the attacker can inject arbitrary packets in the communication channel or control the guest blockchain together with the IBC connections it facilities.

Since the guest blockchain assumes the safety of the host blockchain (§III) it does not provide countermeasures to such attack. To further mitigate the effects of possible attack, implementers should rate limit the light clients. In the event of a security breach, this gives honest actors more time to recognise the issue and react by, for example, shutting down the light clients and hence restrict impact of the attack.

D. Expansion to Additional Blockchains

The guest blockchain has been designed with minimal assumptions in order to make it broadly applicable to many IBC-incompatible blockchains. While the current implementation is for Solana, we have already conducted preliminary work on how our approach could be applicable to other blockchains. Next, we provide a brief discussion of our findings:

- NEAR supports light clients and state proofs but does not offer a host function to view past block hashes [22] (a requirement when establishing IBC connections). The guest blockchain addresses this by having the Guest Contract track past guest blocks.
- TRON lacks state proofs [21]. The guest blockchain addresses this by offering a Merkle trie managed through a Proof-of-Stake consensus mechanism.
- While we are unaware of blockchains lacking introspection of block timestamps, they may theoretically exist and could be an application for the guest blockchain. A timestamp can be introduced by using the median of the signer's timestamp [38] or other decentralised methods [39].
- Lastly, the guest blockchain may be useful in systems whose light clients have high resource demands. Since the guest blockchain design is simple and comes with a lightweight light client implementation, it might replace the host light client on the counterparty blockchain.

VII. Conclusion

This paper introduces the guest blockchain, a novel approach that brings IBC interoperability to previously isolated blockchain ecosystems. Our work makes three key contributions. First, we present a blockchain-agnostic design that enables IBC compatibility without modifying the host blockchain. By leveraging existing smart contract capabilities, our approach provides all required IBC features—including provable storage, light client support, and block introspection—while inheriting the security and liveness guarantees of the host blockchain. Second, we demonstrate the practical viability of our approach through a deployment on Solana's main network. Despite Solana's restrictive runtime, we successfully implemented all required IBC components. This shows that our design can work even in challenging environments with strict limitations. Third, our evaluation shows that the guest blockchain achieves performance and costs comparable to native IBC implementations. The measured latency and fees demonstrate that adding this interoperability layer introduces minimal overhead while maintaining all security guarantees.

Beyond these immediate contributions, our work opens new possibilities for blockchain interoperability. The guest blockchain approach can be adapted to most modern blockchains, provided they offer basic smart contract functionality. This makes it a powerful tool for expanding the IBC ecosystem, connecting assets across previously incompatible chains.

Acknowledgments: This work was supported by Fundação para a Ciência e a Tecnologia (FCT) under plurianual grant UIDB/50021/2020 and project ScalableCosmosConsensus.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[3] "The NEAR white paper," NEAR Foundation, 2020. [Online]. Available: https://pages.near.org/papers/the-official-near-white-paper/

[4] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," The Solana Foundation, Tech. Rep., 2018. [Online]. Available: https://solana.com/solana-whitepaper.pdf

[5] "TRON: Advanced decentralized blockchain platform," 2018, whitepaper Version 2.0. [Online]. Available: https://tron.network/static/doc/white_paper_v_2_0.pdf

[6] J. Kwon and E. Buchman, "Cosmos whitepaper," 2016. [Online]. Available: https://v1.cosmos.network/resources/whitepaper

[7] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. K. Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of Polkadot and its design considerations," Jun. 2020. [Online]. Available: https://arxiv.org/abs/2005.13456

[8] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: ACM, 2021, pp. 35–48.

[9] N. Mittal, S. Pal, A. Joshi, A. Sharma, S. Tayal, and Y. Sharma, *Comparative Analysis of Various Platforms of Blockchain*. John Wiley & Sons, Ltd, 2021, ch. 23, pp. 323–340.

[10] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *ACM Comput. Surv.*, vol. 54, no. 8, Oct. 2021.

[11] K. Ren, N.-M. Ho, D. Loghin, T.-T. Nguyen, B. C. Ooi, Q.-T. Ta, and F. Zhu, "Interoperability in blockchain: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 750–12 769, Dec. 2023.

[12] J. Zheng, D. K. C. Lee, and D. Quian, "An in-depth guide to cross-chain protocols under multi-chain world," *World Scientific Annual Review of Fintech*, vol. 1, Jun. 2023. [Online]. Available: https://ssrn.com/abstract=4476061

[13] C. Goes, "The Interblockchain Communication Protocol: An overview," Jun. 2020. [Online]. Available: https://arxiv.org/abs/2006.15918

[14] MiX, "Introduction to NEAR-IBC," Aug. 2023. [Online]. Available: https://medium.com/omnity/introduction-to-near-ibc-how-to-implement-the-ibc-protocol-with-smart-contracts-a8b0e13cb886

[15] "Token bridge app," Wormhole, 2023, whitepaper 0003. [Online]. Available: https://github.com/wormhole-foundation/wormhole/blob/main/whitepapers/0003_token_bridge.md

[16] J. Kilpatrick, W. Moglia *et al.* (2023) Gravity bridge. [Online]. Available: https://github.com/Gravity-Bridge/Gravity-Docs

[17] "Axelar network: Connecting applications with blockchain ecosystems," 2021, whitepaper Draft 1.0. [Online]. Available: https://axelar.network/axelar_whitepaper.pdf

[18] J. O. Chervinski, D. Kreutz, X. Xu, and J. Yu, "Analyzing the performance of the Inter-Blockchain Communication protocol," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Porto, Portugal: IEEE, Jun. 2023, pp. 151–164.

[19] J. Kim, M. Essaid, and H. Ju, "Inter-Blockchain Communication message relay time measurement and analysis in Cosmos," in *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*. Takamatsu, Japan: IEEE, 2022, pp. 1–6.

[20] M. Vines, "Simple payment and state verification," 2019. [Online]. Available: https://docs.solana.com/proposals/simple-payment-and-state-verification

[21] "Create an alternative endpoint similar to getStorageAt that also includes a Merkle proof," 2023. [Online]. Available: https://github.com/tronprotocol/java-tron/issues/5359

[22] S. Lanlege, "NEP-384: Host function for fetching block hashes in contract runtime," 2022. [Online]. Available: https://github.com/near/NEPs/pull/384

[23] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology – CRYPTO '87*, C. Pomerance, Ed. Heidelberg, Germany: Springer, 1988, pp. 369–378.

[24] H. S. de Ocáriz Borde, "An overview of trees in blockchain technology: Merkle trees and Merkle Patricia tries," Feb. 2022. [Online]. Available: https://researchgate.net/publication/358740207

[25] J. Albert and S. Chaumette, "On the impossibility to forge illegitimate proofs of membership in Merkle (Patricia) trees," in *Web3Sec – Workshop Encouraging Building Better Blockchain Security*, Austin, United States, Dec. 2023. [Online]. Available: https://acsac.org/2023/workshops/web3sec/WEB3SEC2023-albert.pdf

[26] C. Goes, "ICS-4: Channel and packet semantics," Aug. 2019. [Online]. Available: https://github.com/cosmos/ibc/tree/main/spec/core/ics-004-channel-and-packet-semantics

[27] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-Stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85 727–85 745, 2019.

[28] G. A. Pierro and R. Tonelli, "Can Solana be the solution to the blockchain scalability problem?" in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Honolulu, HI, USA: IEEE, 2022, pp. 1219–1226.

[29] "Solana documentation," Solana Foundation. [Online]. Available: https://solana.com/docs

[30] "Programs are unaware of non-default heap sizes." [Online]. Available: https://github.com/solana-labs/solana/issues/32607

[31] M. Nazarewicz, "Solana transaction size limit," https://mina86.com/2025/solana-tx-size-limits/, Feb. 2025. [Online]. Available: https://mina86.com/2025/solana-tx-size-limits/

[32] ——, "Solana signature count limit," https://mina86.com/2025/solana-signatures-count-limit/, Feb. 2025. [Online]. Available: https://mina86.com/2025/solana-signatures-count-limit/

[33] ——, "Mutable global state in solana," https://mina86.com/2025/solana-mutable-global-state/, Mar. 2025. [Online]. Available: https://mina86.com/2025/solana-mutable-global-state/

[34] ——, "Solana IBC evaluation dataset," 2024. [Online]. Available: https://codeberg.org/mina86/be-my-guest/src/branch/stats

[35] (2024) Low latency transaction send: What are bundles? Jito Labs. [Online]. Available: https://docs.jito.wtf/lowlatencytxnsend/

[36] V. Buterin, "On security of cross-chain applications," Jan. 2022. [Online]. Available: https://www.reddit.com/r/ethereum/comments/rwojtk/comment/hrngyk8/

[37] R. Belchior, P. Somogyvari, J. Pfannschmid, A. Vasconcelos, and M. Correia, "Hephaestus: Modelling, analysis, and performance evaluation of cross-chain transactions," Nov. 2023. [Online]. Available: https://techrxiv.figshare.com/articles/preprint/Hephaestus_Modelling_Analysis_and_Performance_Evaluation_of_Cross-Chain_Transactions/20718058

[38] M. Baricevic, "BFT time," Sep. 2019. [Online]. Available: https://github.com/tendermint/spec/blob/master/spec/consensus/bft-time.md

[39] Y. Gao and H. Nobuhara, "A decentralized trusted timestamping based on blockchains," *IEEJ Journal of Industry Applications*, vol. 6, no. 4, pp. 252–257, 2017.