Arabian Journal for Science and Engineering (2024) 49:3285–3312
https://doi.org/10.1007/s13369-023-07899-2

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

Blockchain Based n-party Virtual Payment Model with Concurrent Execution

Sujit Sangram Sahoo[1] · **Aravind R. Menon**[1] · **Vijay Kumar Chaurasiya**[1]

Received: 14 June 2022 / Accepted: 9 April 2023 / Published online: 10 June 2023

Abstract
Blockchain technology has limitations in terms of scalability and throughput of transactions compared to other payment methods such as VISA and smart cards. But, Off-chain payment solves this issue by performing micro-transactions without communicating with the ledger. Similarly, Ethereum uses smart contracts to complete Off-chain transactions by constructing state and virtual channels. In this model, we develop an n-party payment system that uses state and virtual channels. The proposed model simultaneously executes different contract instances while they belong to other independent channels. Again, a virtual channel having n-intermediaries takes linear time to resolve payment channel disputes by transferring to the Blockchain. We introduce a global contract GSCC that solves the disputes in constant time in the worst case without including Blockchain. The dispute-solving process does not affect any intermediary node and ensures the intermediary is not losing coins. GSCC stores all the prior information of the active participants. But, GSCC has no communication overhead to other parties and ensures the execution of disputes with total conflict amount. The resultant model guarantees the execution of every contract, fair dispute resolution, and security balance. Our proposed theorems and zero-knowledge proof ensure the security of the n-party model.

Keywords Blockchain · Smart contract · Off-chain payment · Payment channel dispute · Concurrency

1 Introduction

Blockchain technology was introduced in 2008 by Satoshi Nakamoto [1] for the transfer and use of cryptocurrency, and it is a decentralized, peer-to-peer append-only ledger. It contains a database between many users and is distributed in nature with a consensus to manage. Various applications have used blockchain over the past decade, including data storage and transaction handling. It started as a platform for exchanging cryptocurrencies but expanded to other domains. At the same time, Ethereum uses self-executable programs called smart contracts to avoid the limitations of earlier blockchain mechanisms such as Bitcoin. In 2014, the Ethereum blockchain had introduced [2]. Smart contracts enable a user to store, update and execute the state of any program. Implementing smart contracts on the blockchain allows multiple users to perform transactions, play games, vote, and many other applications. In this paper, we propose a system that can run smart contracts off-chain by preserving the security and immutability of data. It improves the system's performance without compromising data and user integrity. Before describing the entire model, we have to discuss the scalability issue, which is the leading cause of the whole problem.

1.1 Scalability of Blockchain

Blockchain technology has faced issues like Scalability and efficiency from the time it introduces. Credit cards perform transactions at a much higher rate than the block creation process in the blockchain. VISA can complete 70,000 transactions per second, whereas on-chain transaction execution in Ethereum and bitcoin is much slower [3]. It is only due to the block generation procedure of blockchain that involves achieving consensus, mining, and validation of transactions.

✉ Sujit Sangram Sahoo
rsi2018005@iiita.ac.in

Aravind R. Menon
mit2020049@iiita.ac.in

Vijay Kumar Chaurasiya
vijayk@iiita.ac.in

[1] Department of Information Technology, Indian Institute of Information Technology Allahabad, Prayagraj 211015, India

So Scalability and throughput of transactions have been one of the biggest challenges in blockchain over the past decade. Although blockchain guarantees immutability, anonymity, and transparency, it comes at the cost of extensive execution time [4, 5]. There are many models use applications of blockchain that are also in scalability problems [6, 7]. Much research has focused on solving this problem, and the problem mainly consists of two types of solutions: on-chain and off-chain-based. Sharding the database [8] is an example of an on-chain solution, where the data is divided into multiple nodes across the ledger to improve computation and search. The off-chain transaction mechanism introduced in bitcoin is called the bitcoin lightning network [9]. The main idea of the lightning network is to execute transactions off-chain, so there is no excess time on block verification, consensus, and creation. Raiden [10] and sprites [11] are also other off-chain solutions. The sprites discuss a model that helps reduce the time for the payment routing process in the bitcoin lightning network. They have introduced the concept of state channels for payment. But Perun [12] describes the ledger state channel and virtual state channel protocol in detail. The perun protocol is written in solidity language and uses Ethereum smart contracts to create channels, lock coins, handle disputes, and close channels. In the same way, the general state channel paper [13] extends the work done in the perun paper by defining the state channel and virtual channel processes for more extended channels. They have described how new virtual channels build over existing state or virtual channels. Again the multiparty [14] improves upon the dispute time of the general state channel by introducing the dispute board. Our problem statement depends on the state and virtual channel networks that are part of off-chain payment solutions. We discuss both mechanisms in the next section with dispute handling. Table 1 compares existing schemes to scalability issues. Figure 1 explains the summary of our proposed model.

Table 1 Different existing schemes relating to scalability issue

Network name	Blockchain technology	Payment channel	Dispute mechanism	Intermediary security issues	Limitations
Lightning network	Bitcoin	Multi-signature	NA	Yes	Intermediaries can see all the transaction details for routed payments
Raiden	Ethereum	Smart contract	NA	Yes	Supports only simple payments, has not provided mechanism for longer channels
Sprites	Ethereum	Smart contract	Yes	No	Does not formalize the protocol for different types of channels, such as state and virtual
Perun	Ethereum	Smart contract	Yes	No	Does not provide details on various types of disputes
General state channel	Ethereum	Smart contract	Yes	No	Dispute solving time is linear to number of intermediaries
Multiparty virtual state channel	Ethereum	Smart contract	Yes	No	Low concurrency in multiple parties executing contract instance on the same channel

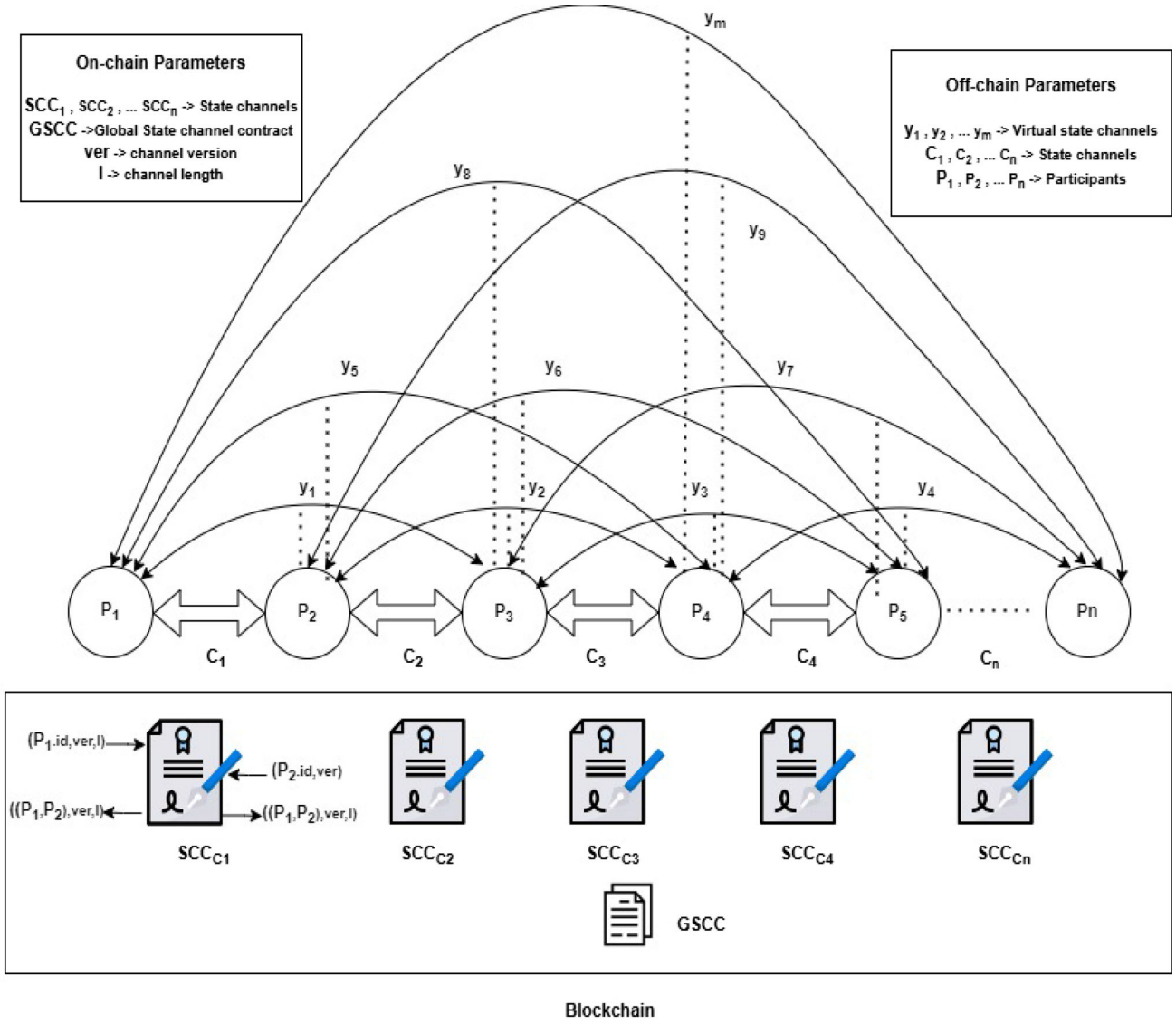

Fig. 1 Summary of the proposed n-party virtual state channel network

1.2 State Channel Networks

The main focus is the off-chain channels: state channel networks and virtual payment channels. The virtual payment channels are built over state channel networks. The primary state channel contract (SCC) protocol, shown in Fig. 2, is defined in perun [12]. The protocol establishes the channel creation process, contract execution, and closure. In channel creation, Alice and bob lock coins in an SCC, deploy them in blockchain and hold a contract instance G_0. Suppose one party wants to perform a transaction. In that case, he sends the updated contract instances, the following version number, and the updated function with his signature for the other party to verify. But in the transaction execution phase, both create the next contract instance for version w by updating locally and passing w+1 to the other party. If the other party verifies with G_w (state of the contract with version number w) and the function obtained, they sign and send it back. If both parties sign, it is a valid contract instance. Any party can close the channel using the contract instance to call the close method and send it to the blockchain. According to the latest contract instance state, the blockchain releases funds, and both parties get their coins. The state channel forms the basic building block for virtual state channels, and the proposed solution entirely depends on the virtual channels and the disputes solving mechanism. The contract reaction time defines by the term Δ, which is the worst-case reaction time of a contract from its participants.

Virtual Payment Channel The virtual payment channel network shown in Fig. 3 is created on top of two existing state channels. Therefore, state channels must complete first to

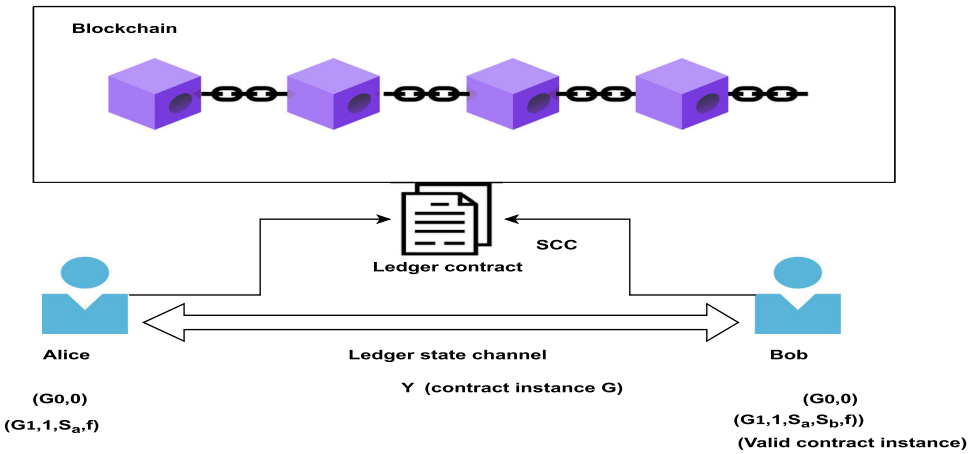

Fig. 2 Ledger state channel

build a virtual channel. Alice and Bob want to create a virtual channel and utilize the ledger state channels SCC1 and SCC2. Since these are already deployed on the blockchain, intermediary Ingrid utilizes a special contract instance on both channels, VSCC (v_a and v_b). In v_a, Alice and Ingrid lock coins so that Ingrid plays the role of bob. Similarly, in v_b, Ingrid plays the role of Alice. Now, Ingrid and VSCC's v_a and v_b play the role of "judge" and channel creator and closer, respectively. The transaction execution phase is similar to the ledger state channel, where both have an initial contract instance and version number (G_0,0). One party updates the instance, signs it, and sends it to the other party along with the executed version number and function. The other party verifies locally and signs it to create a valid contract state (G_1,1). When a party requests the VSCC to close the channel by running its instance close, the latest version determines as before, and funds are released so that Ingrid is financially neutral.

It solves certain drawbacks of the state channel network. A state channel requires a direct channel between two parties, but payment between two parties overlaps many intermediaries in the virtual channel. The creation and closing of a channel in a virtual channel network are off-chain in contrast to a state channel network, which is on-chain. It helps to reduce the involvement of the blockchain. The virtual channel creation over an intermediary party performs the channel creation, dispute handling, and closing procedure, and this intermediary plays the role of the dispute solver in the virtual system. A dispute arises when two parties disagree with the current state of their channel or if at least one party is dishonest. If all parties involved are honest, the optimistic case, and if any party is dishonest, the pessimistic case.

Handling Disputes Suppose Alice creates a dispute, such as Bob is not responding or Bob is trying to validate an older version of the contract instance. In that case, she registers by calling the contract instance of v_a called "register(G_w,w,s_b)" and sends the latest version of (G_w,w). v_a sends this to Ingrid, who registers this with v_b by calling v_b contract instance "register(G_w,w,s_b)", where s_b is the private key of Alice. Bob responds to v_b by executing a contract instance of v_b called "register(G'_w,w',s_a)", where s_a is the private key of Bob. v_b sends this to v_a via Ingrid, registering it in its instance. Finally, version numbers w and w' are compared, and the latest version is validated. Alice goes directly to the blockchain and raises the dispute to validate the contract instance if Ingrid is malicious.

1.3 Advantages of Virtual Channel

The virtual channel does not involve the blockchain in contract execution. State channels require the blockchain for channel creation, closing, and handling disputes. In the case of virtual channels, intermediaries are involved in all three processes, whereas state channels require the use of blockchain. It helps reduce the construction time and dispute handling cost for virtual channels. The intermediaries are not involved in the transaction execution process, thus providing privacy to the participating parties. Virtual channels are made longer by building new channels over the existing state or other virtual channels and help to resolve the privacy loss and time consumption involved in payment routing methods. The virtual channel between two parties allows direct fund transfer instantly without interacting with intermediaries.

1.4 Disputes in Payment Channels

In a payment channel, A party is in dispute with another party when they disagree on a valid state of a contract instance. It happens when one party refuses to sign an instance or tries to execute an older instance. Such scenarios handle in various ways by the previous payment channel models. The disputes handled directly by the blockchain are called direct disputes. In the case of a model that uses direct dispute, the dispute initiating party registers the contract instance with the

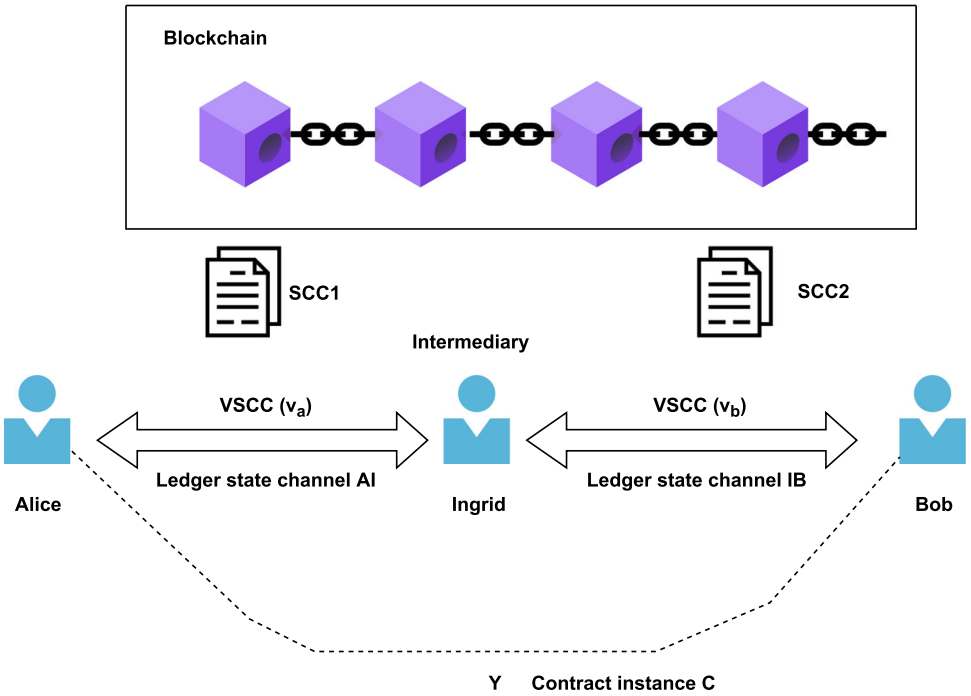

Fig. 3 Virtual state channel

blockchain. The blockchain takes the help of various miners to perform the dispute handling process. Ledger state channels described in perun [12] and general state channel [13] use the method of direct disputes. An Indirect dispute model is where all disputes are first handled by an intermediary party, failing which the blockchain is involved. Virtual channels described in perun and general state channel use an indirect dispute model, where the intermediary tries to resolve first before going to the blockchain. The multiparty [14] has a direct dispute model, which uses a dispute board instead of the blockchain. Therefore, in the model [14], the blockchain is not involved in the dispute process in any scenario.

Dispute Handling in Virtual Channels A dispute occurs when a party does not cooperate by responding on time or when parties disagree with the current valid state of the contract instance. The protocol has two ways of handling disputes: direct and indirect disputes.

1. Direct Dispute: Assume that two parties, Alice and Bob, are having a dispute and cannot solve it by themselves. In a direct dispute, one of the parties contacts the blockchain directly to intervene and solve it. The blockchain takes the latest valid responses from the parties. It validates the most recent instance by choosing the highest version number of the contract and is executed directly on the ledger. The paper [14] uses another type of direct dispute called the dispute board to perform this task.
2. Indirect Dispute: Assume that two parties, Alice and Bob, are having a dispute and cannot solve it by themselves. In the indirect dispute model [13], one of the parties contacts the intermediary by sending their current contract state to special contract v_a. The contract v_b gets a similar response from the other party. The intermediary plays the role of the blockchain by validating the most recent contract instance in v_a and v_b. If any of the parties is dishonest, the dispute is taken to the ledger and executed, similar to a direct dispute.

The paper [13] proposes a method to extend the indirect dispute method to a chain of channels. When virtual channels are built on top of other virtual channels and ledger channels, disputes that are unsolved at the higher level are taken to the next lower level until it reaches the blockchain.

1.5 Dispute Handling Time

This section will discuss the dispute handling time for each of the existing off-chain models. Sprites has a model that handles disputes simultaneously using a pre-image manager contract and the blockchain. Perun has proposed direct disputes in state channels, where a party in dispute goes directly to the blockchain, which settles the dispute. This method is time-consuming since the blockchain involves miners mining a block to solve the dispute. Hence, block creation takes approx 10 min or more. Virtual state channels use indirect disputes, where an intermediary tries to solve the dispute first, failing which the parties approach the blockchain. Here, the worst-case time is similar to the direct dispute scenario. The general state channel model has a dispute time linear to the number of intermediaries involved. Therefore, the worst-case

time is $n\Delta$ (Δ is the maximum time it takes for a contract to respond to a request) + the block creation time.

Our model proposes a dispute-handling mechanism by a global contract deployed by participating parties on the blockchain. The contract uses zero-knowledge proof, append-only board, and cryptographic signature to ensure the dispute-solving process is secure and authentic. The worst-case time complexity for our dispute model is

Definition of Terms The symbol Δ represents the reaction time of a contract in the worst case,i.e., the time it would take to react to the farthest request from a participant in a channel. All the activities done by contracts are executed in the order of Δ. It measures more accurately in terms of "rounds." If party P_1 wants to start a virtual channel with P_3 using P_2 as an intermediary, in the first round, P_1 sends the updated contract instance by invoking the create method using parameters. Contract instance in the channel informs P_2 about this. In the next round, P_2 waits for confirmation of channel creation from P_3. Similarly, all parties respond to contract requests within Δ time to perform dispute handling and closing of the channel.

1.6 Research Gap

The virtual channel protocol described in perun is for two parties and an intermediary. In general state channel networks, virtual channels are made longer by recursively building new ones over existing ones. However, the dispute handling in a pessimistic case becomes linear to the number of intermediary nodes ($n\Delta$). The concept of a dispute board has improved the dispute handling time. But, there are issues in the dispute board mechanism that affect its performance.

Dispute board The dispute board is a particular contract deployed on the blockchain to resolve any disputes occurring on the channel. It maintains a set of tuples containing unresolved disputes (D_{aux}) and completed disputes (D). The incoming disputes are handled using these tuple values. It helps the board quickly identify if any disputes are duplicates or whether some parties are malicious. It provides all parties with Δ time to respond with their contract instances.

Drawback of Dispute board The dispute board has the data about disputes between parties. It does not have information about which parties are a part of a particular contract instance. Channel creation and contract information are registered with the overall channel contract called mpVSCC. Therefore, whenever a new dispute arises, the dispute board must contact the mpVSCC contract to check whether the parties are part of that contract instance. It creates an overhead in terms of computation time. If the number of disputes increases very fast at a particular time, then the dispute board might fail. There is no prior information with the dispute board regarding the disputes, and the dispute board may be working slowly; this could fail to Guarantee execution.

1.7 Our Contribution

Our model addresses the issues highlighted in Table 1. The proposed system aims to make an n-party system fully concurrent and solve any disputes using a global state channel contract (GSCC) in $O(\Delta)$ time instead of $O(n\Delta)$. There are channels between every pair of parties from 1 to n, constructed using the state and virtual channel protocols. The main idea is to create a system under a single contract, GSCC, with independent channels. The GSCC contract is a global contract deployed by any party on the blockchain before channel construction and game registration(discussed in Sect. 3). The single contract ensures that the GSCC controls the data of all activities done by all parties. It is aware of all involved parties and their channels at any time. When a party raises a dispute, the GSCC contract does not have the overhead of contacting another contract for information. The independent channels in our model give the system high concurrency since any party can access channels freely and simultaneously. We represent the idea by showing the execution of a concurrent two-player game in an n-party system. At the time of game registration with the GSCC, all n parties agree to some basic parameters (discussed in Sect. 3). If parties P_1 and P_2 play the game, one makes the first move and sends the updated state to the other party. The other party validates this state, makes its move, and sends the latest state to the first party. The process continues until one party refuses to or is unable to respond. The system allows games to execute in parallel, i.e., the game between P_1 and P_2 will not impact game execution in the P_3 and P_4 channels. GSCC keeps track of all channels' status and the results of every game and resolves disputes between parties. All parties trace each game by accessing the GSCC to ensure fairness. It helps participating parties conduct n-player tournaments without a trusted third party. GSCC also helps handle disputes much faster than the general state channel model. The resultant model is also helpful for participants to use as a simple payment system.

2 Literature Survey

This section discusses different research models relating to off-chain payment systems. The main issue is the scalability problem in the basic blockchain models; later, many schemes solve it. The on-chain and off-chain solutions are based on parameters such as channel opening and closing mechanism, dispute handling method, scalability, and execution time. Table 2 explores on-chain and off-chain models.

On-chain Scalability Solutions Research on the scalability problem resulted in various on-chain solutions, namely sharding. [15] proposes a model named RapidChain, which is resilient to Byzantine faults and performs communica-

Table 2 Comparing on-chain and off-chain solutions

Network name	Blockchain technology	On-chain or off-chain	Scalability	Opening and closing channel	Dispute handling time
Lightning network [9]	Bitcoin	Off-chain	Highly scalable. Intermediaries are involved in routing	Performed on-chain	Handled by blockchain. Hence, time taken is block creation time (10 min approximately)
Perun network [12]	Ethereum	Off-chain	Highly scalable. State channels enable thousands of transactions per second	Performed on-chain for state channels and off-chain for virtual channels	Does not provide details on various types of disputes
General state channel network [13]	Ethereum	Off-chain	Highly scalable. Long virtual channels enable thousands of transactions per second	Performed off-chain	Dispute solving time is linear to number of intermediaries
Multiparty state channel with direct disputes [14]	Ethereum	Off-chain	Highly scalable. Multiparty channels enable thousands of transactions per second	Performed off-chain	Disputes are solved in constant time with a dispute board
Bitcoin network [9]	Bitcoin	On-chain	Low scalability. Block creation process takes place for each new transaction	Does not require payment channels	Block validation is done by miners
Ethereum framework [2]	Ethereum	On-chain	Low scalability. Block creation process takes place for each new transaction	Does not require payment channels	Block validation is done by miners

tion between parties and storage without a trusted setup. It solves other existing sharding techniques' linear communication and computation time. The proposed system performed 7300 tx per sec on a network of 4000 nodes. The literature [16] provides a complete study on existing sharding protocols such as RSCoin, OmniLedger, RapidChain, and ChainSpace in terms of dealing with cross-shard transactions, transaction models, pipelining, and so on. The model [17] describes fastpay, a blockchain scalability solution that aims to prevent the double-spending problem. The scheme [18] is another sharding-based scaling solution. These models are not discussed any particular solution for virtual payments.

Off-chain Scalability Solutions The scalability of blockchain can be improved drastically by having parties execute transactions away from the main blockchain network, called off-chain execution. Eberhardt, Jacob, et al. [19] have proposed an off-chain solution called ZoKrates, to improve the scalability and privacy of the system. ZoKrates is a toolbox used to conduct off-chain computations such as zero-knowledge proof using smart contracts to reduce on-chain involvement. Cheng Xu et al. [20] have proposed a system called SlimChain to improve scaling and storage in the blockchain. SlimChain's stateless design commits changes on-chain and maintains storage and transaction execution off-chain. Randhir Kumar et al. [21] propose a distributed off-chain storage mechanism to prevent unauthorized access to patient's confidential medical information and provide consistency, availability, and integrity of data. These solutions have been improved over the years to involve the blockchain only when necessary. All the models are not in the way to solve the issue, probably in an efficient manner that can appear more accurate in the future.

Lightning Network The off-chain solution to the scalability problem introduced systematically in bitcoin is called the lightning network. Poon et al. [9] introduce the bitcoin lightning network(BLN) concept. The protocol enables parties to perform any number of transactions by executing them off-chain after locking a certain number of coins in a multi-signature wallet. These are called microtransactions, which can complete in a few microseconds. It is achieved by updating the balance of the coins after each transaction rather than the on-chain process of coin exchange, consensus, and block creation. Any party can stop participating in off-chain transactions by going to the ledger and, unlocking the wallet, distributing the funds. The paper also introduces Hashed Timelock Contracts (HTLC), which uses a routing mechanism to perform off-chain transactions between parties that do not have a common communication channel. Lee et al. [22] discuss the robustness of the BLN. DDOS attacks and their effects on the lightning network are explored. A simulation of various attack and defense mechanisms includes random, high-centrality, community-based, and high-degree attacks. The defense mechanisms utilized for these attacks are random, preferential, and balanced defense.

The effectiveness of the attack and defense strategies are discussed and shown in the simulations. These simulations were performed on the BLN mainnet and testnet. George et al. [23] present an analysis of the privacy of BLN. The author runs various attacks on the BLN simulator and testnet to check the routing path and balance discovery for different network topologies. The main idea of this paper is to identify how attackers can gain secret information about networks and exploit them by simply connecting to peers and routing payments. Finnegan et al. [24] explain the availability and reliability of BLN in terms of the amount that can transfer at a time, availability of nodes and payment routes, the success rate of off-chain transactions, and possible reasons for transaction failure. Yuwei et al. [25] explain the BLN routing process success rate, graph construction for various topologies, and impact of network structure on lightning network performance. BLN has an intermediary insecurity issue, which is only for the bitcoin network. We discuss in ethereum model by avoiding the intermediary insecurity issue.

Payment Channels Ethereum blockchain has introduced payment channels for off-chain execution, which can be open and closed between any two parties. Smart contracts have enabled Ethereum blockchain to perform this task seamlessly. They can execute functions automatically when conditions are met and guarantee the same level of security and integrity as blockchain. It helps provide many new features such as state registration, dispute handling, function execution, etc. Stefan et al. [12] describe the protocol for the primary payment channel, multistate payment channel, and virtual payment channel. The protocol is called perun, and it uses solidity language.

Stefan et al. [12] explain the perun payment channels for the two-party system and there is no intermediary insecurity issue. The sprites [11] paper defines channel construction using a global contract called the preimage manager. Perun protocol is an improvement on sprites since it involves the blockchain sparingly. In the case of a non-virtual or ledger state channel, a routing path with multiple intermediaries between sender and receiver has been established before performing transactions. The purpose of the virtual payment system is to enable parties to perform multiple microtransactions in a few seconds without using a routing path or multiple intermediaries between the parties. A single intermediary is used to create a virtual channel between any two parties, even if they do not have a direct ledger channel. The paper clearly defines protocols for virtual channel creation, contract instance update, contract instance execution, and channel closing. Security and efficiency goals are also defined for each of the processes. Gudgeon et al. [26] discuss another layer-2 scaling solution called systematization

Table 3 Summary of existing models relating to virtual channel

Network model	Channel Creation	Dispute handling	Channel closing	Execution
Lightning network [9]	Performed on-chain. Each party locks a certain number of coins with the blockchain. Their balances are updated each time a transaction takes place	Performed on-chain. The blockchain is involved in this process	Performed on-chain. The blockchain releases the funds after receiving signature from both parties	Performed off-chain. After locking their coins in the blockchain, both parties can update balances without involving the blockchain
State channel network [12]	Performed on-chain. Multi-sig smart contracts are utilized to save the state of a transaction between parties and open a payment channel. The process involves deploying contract on the ledger and block creation in Ethereum, which takes 10 min	Performed on-chain. Disputes are run directly on the ledger	Performed on-chain. The payment channel can be closed by the ledger after obtaining signature from the parties involved	Performed off-chain. The state of the contract is updated and signed by the party. The other party can locally compute the contract state and sign after verification
Virtual State Channel Network with Dispute Board [14]	Performed off-chain. The channel construction mechanism is similar to the virtual state channel network discussed above, taking $O(\Delta)$	Performed on-chain. A dispute board contract is set up before constructing payment channel. All disputes are managed by it directly in $O(\Delta)$	Performed off-chain. The channel closing mechanism is similar to the virtual state channel discussed above	Performed off-chain. Intermediary plays the role of each party in their respective state channels
Recursive Virtual State Channel Network [13]	Performed off-chain. Virtual state channels can be built over other virtual state channels or state channels recursively to avoid routing. The construction time of such channels is linear to the number of intermediate parties, $O(N\Delta)$	Performed off-chain. Disputes are handled by the intermediary first. If any party is dishonest, the dispute is resolved by the party in the next layer. The process is repeated and resolved by the blockchain if none of the parties are able to resolve	Performed off-chain. The intermediary closes the channel after receiving signature of latest contract instance from participating parties	Performed off-chain. Once the channel is created, the participating parties can perform transactions off-chain
Multiparty state channel with direct disputes [14]	Performed off-chain. The multiparty protocol enables parties to create channel over existing state and virtual channels	Performed on-chain. The channel does not contain intermediaries. Therefore, the dispute board solves all disputes	Performed off-chain. Once a party initiates the closing procedure, all channels will be closed and funds will be released	Performed off-chain. Transactions can be performed off-chain any number of times while the channel is open
Virtual state channel network [12]	Performed off-chain. Channel is built on top of two existing state channels. The intermediary can obtain contract instances and lock coins independently without involving the ledger. If it takes Δ time to get response in worst case, total time is 2Δ	Performed off-chain. Dispute between participants is first resolved by the intermediary. If any participant is dishonest, the blockchain is involved and execution continues from there	Performed off-chain. Virtual channel can be closed by the intermediary after receiving signature of latest state from the parties. The intermediary remains financially neutral in the entire process	Performed off-chain. The intermediary plays the role of both parties in each of the state channels to carry out transactions

of knowledge(sok), providing a comparison between protocols and research on layer-2 transactions. Zhong et al. [27] have designed an off-chain payment system called the secure large-scale instant payment (SLIP). The system locks coins to be circulated among parties and uses an aggregate signature scheme to connect the channels. The efficiency and privacy of the SLIP system are analyzed, and a comparison with the BLN. Perun has discussed a single intermediary, and there is insufficient information regarding payment channel disputes.

General State Channel Network [14] extends the perun protocol for state channel networks and virtual state channel networks to construct a general state channel network model. A virtual channel using perun protocol creates over a single intermediary between the two participants. If multiple intermediaries involve between two parties, the proposed model uses a recursive approach to construct virtual channels over existing and virtual state channels. It is carried out until there is a direct virtual channel between the participants, and it requires $O(\Delta)$ time for dispute solving. Lisa et al. [14] have extended the previously proposed two-party virtual network by creating a system that supports multiple parties concurrently. They have also proposed a model called a virtual channel with direct disputes, making use of a dispute board to handle all the disputes. The worst-case time complexity for dispute handling and complex channel protocols has improved. A four-round protocol has been proposed to ensure that the parties can execute contract instances concurrently. The dispute board is not working concurrently and has no information about the involved parties. It creates a problem at the time of dispute solving. Table 3 shows a comparison between existing schemes.

2.1 Other Related Work

Vitalik et al. [28] discuss another layer2 scalability solution of Ethereum called plasma. It also consists of off-chain transaction execution by creating replicas or duplicates of the original blockchain and sharing funds from the main chain. The child blockchains form more child chains, resulting in a tree-like structure. Changes in the child chains are updated periodically to their parent chains until they reach the main chain. Tairi et al. [29] propose a payment channel hub(PCH) to resolve scalability in blockchain and allow off-chain payments. It is a three-party protocol involving an intermediary that provides a cryptographic challenge to the participating parties.

Security is an essential part of blockchain technology. Providing privacy to users, authenticity to data, and transparency is vital. The concept of cryptographic proofs such as zero-knowledge proof [30, 31] has excellent future scope in blockchain security. Zero-knowledge proof is a mechanism where one party can convince another that he knows certain sensitive information without revealing it. Yang et al. [32] propose an identity management system model using blockchain and zero-knowledge proof. They have identified threats in existing identity management systems such as a single point of failure, privacy leakage, and centralization. They have utilized smart contracts and zero-knowledge mechanisms such as zk-SNARKS to create a challenge-response protocol to allow users to have privacy. Wanxin et al. [33] propose a traffic management and privacy-preserving model using blockchain and zero-knowledge proof. The author has identified data integrity and privacy in the traffic network. A model is proposed by integrating blockchain and non-interactive zero-knowledge proof to improve security and performance on the hyperledger fabric platform. The author explored the advantage of Zcash, which utilizes zero-knowledge succinct non-interactive arguments of knowledge (ZK-SNARK) to improve privacy. We provide zero-knowledge proof to prove our model to the global contract.

3 Proposed Methodology

The idea is to propose a model that helps an n-party virtual system execute contracts concurrently and solve disputes constantly. It means that parties can run and update contracts on their channel irrespective of the traffic created by other parties simultaneously. The system comprises a global contract GSCC that handles the dispute handling process. The summary of the proposed scheme has described in Fig. 1, but the Fig. 4 describes the easy working of all the functionalities.

1. All n parties have to be able to perform transactions or update contract states with any other party in the system concurrently.
2. Disputes have to be handled constantly by the global contract.
3. Time has to be optimized for all processes by the global contract by setting a limit on each process.

The channels from C_1 to C_n are ledger state channels which exist between the parties P_1 to P_n. The channels Y_1 to Y_m are virtual state channels constructed above ledger state channels and other virtual state channels. The construction has multiple levels, and each channel level is constructed over the lower levels. The first and lowest level is the state channels C_1 to C_n. The dotted line represents the intermediary over which the virtual channel is constructed.

The secondary objective is to design the system in such a way that there is consensus among all the parties to perform construction, dispute handling, execution, closing, and so on. All the intermediary parties and channel participants

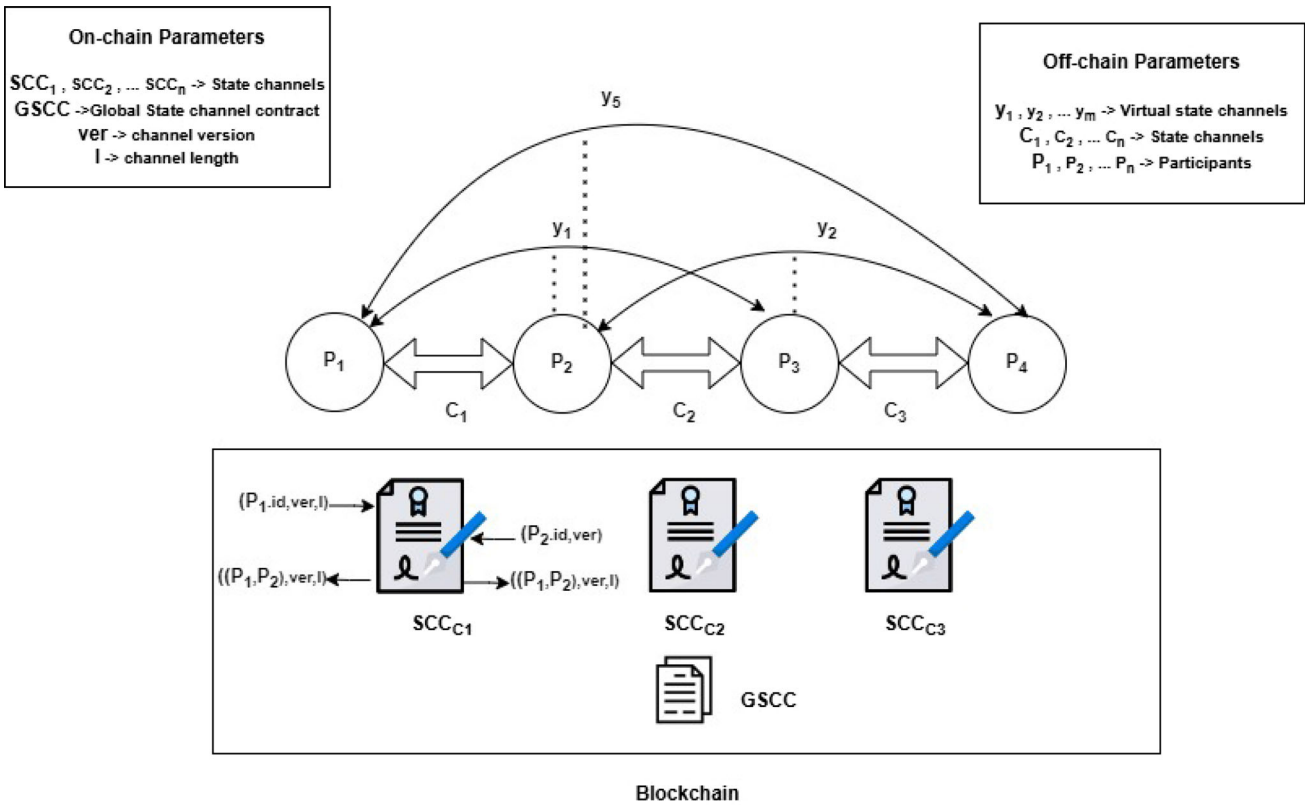

Fig. 4 Proposed four-party diagram for methodology explanation

Fig. 5 Global state channel contract for channel construction

should not lose their coins and each off-chain activity has to be protected against malicious parties. It is essential to explore how the proposed solution improves upon some existing off-chain solutions. The main idea is to extend the application of payment channels to design an n-party concurrent game execution mechanism. In the working procedure, we explain the channel construction, game registration, coin locking mechanism, game execution procedure, dispute handling, and channel closure procedure in detail. There are five main phases in the working process of the model:

1. Channel creation
2. Game registration and coin locking phase
3. Execution procedure
4. Dispute handling
5. Channel closure

Following this, we identify and analyze the security and efficiency goals. After this section, we explore some theoretical proof definitions and their descriptions. Off-chain transactions are performed in Ethereum blockchain by using multistate smart contracts and constructing payment channels. Various protocols have developed over the years, such as Raiden, sprites, perun, etc. Our model uses the perun protocol

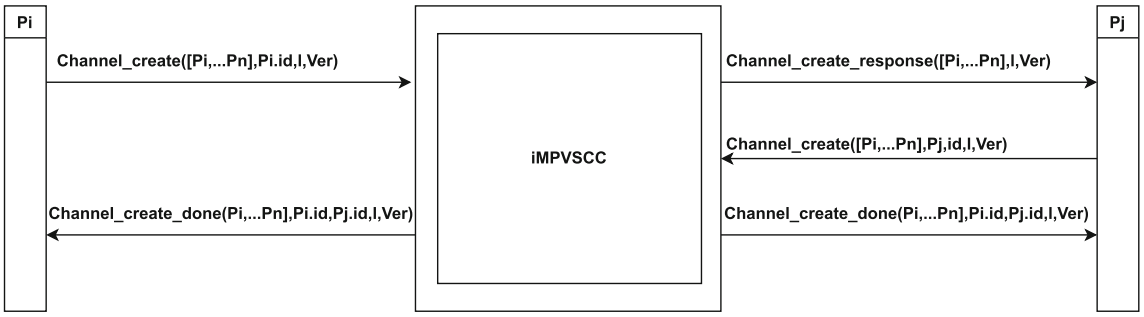

Fig. 6 iMPVSCC contract for channel creation

for state and virtual channel construction. This paper proposes two innovative special contract called the iMPVSCC and GSCC.

The proposed model uses the state updating and signature functionality of the Ethereum smart contract. The idea is to utilize the model to play two-player games amongst all the n parties, concurrently and simultaneously. All parties have independent channels connecting them, enabling concurrent execution of multiple contract instances. The global contract GSCC uses an append-only contract board to settle disputes and decide the winner of each match played in the system in each round. After all the games of all the rounds are completed, the GSCC calculates the final balances of all the parties, uses channels to transfer funds correctly and releases these funds.

3.1 Total cost of GSCC Contract

Channel creation, game registration, coin locking, and closing processes are one-time tasks that GSCC runs. These processes consume more time and computation resources. However, we obtain a trade-off where the execution time and dispute handling time reduce significantly. In our n-party system, the GSCC holds the data of all n^2 channels, the balances in each channel, the append board containing details of the game, and the final contract state of each channel. Every time a dispute arises between parties that participate in a game, the GSCC gets involved in resolving it. GSCC has the authority to add data to the append board since it has all the game details. GSCC approves the final valid channel instance for the coins to exchange in the channel. Now, we explore the creation of channels, state execution, dispute handling and closing of the game in detail.

3.2 Channel Creation

Suppose parties $P_1, P_2, ...P_n$ want to construct a n-party channel with functional/design requirements explained previously. We first make use of a global contract manager to help facilitate the construction mechanism.

Algorithm 1 Channel Construction

Require: Input: $length$,$number$
1: **while** $length < number$ **do**
2: i=1
3: **for** i =1 to $number$ **do**
4: **if** $i + length > number$ **then**
5: break from loop
6: **end if**
7: construct(i,i+length)
8: **end for**
9: $i = i + 1$
10: **end while**

The global state channel contract shown in Fig. 5 is a contract that is triggered at the time of channel creation by party P_1. Party P_1 deploys the GSCC contract on the blockchain. Any party can perform the initialization process in the set of parties in the n-party network. The instance channel initialization is called by passing the name of the parties and the identification of sending party to the global contract. The global state channel sends the channel to create a contract instance to the party by assigning a version number(ver) and length of the channel (l). This ver ensures that the channel of a particular size is made precisely once by the party. Any party verify it by accessing the GSCC. The party uses these details obtained from the GSCC to create a channel on the iMPVSCC, as shown in Fig. 6.

The iMPVSCC takes input from one party and triggers a response from the other party for channel creation. The other party responds with its identification number and sends the length and version with its signature. The iMPVSCC can construct the virtual channel of length l and lock funds from both parties. GSCC provides each party with data on the fund to be locked in each channel. GSCC encrypts all the data it sends to parties with its key σ_g. Finally, a channel creation message is sent with the identification of both parties. It is sent to the GSCC by one of the parties informing that the channel has been constructed. The GSCC then moves on to the next channel creation with the next party, and this continues until all pairs of parties have a channel with each other. The GSCC is a global contract that handles the channel creation protocol

of the system. If the length value + party number exceeds the total number of parties, the contract resets the party number to one, increments the length by one, and continues the process. It ensures that the channel of length two is constructed first (initial value of l is 2), then length three is built, and so on. It must not be that the GSCC validates one contract creation process at a time. Therefore, a party can only construct the channel with contact with GSCC. The channel construction algorithm discusses in Algorithm.1. The channel can be constructed after all the n parties have connected ledger state channels. Creating a state channel takes time to deploy a contract on the Ethereum blockchain and create blocks.

As discussed above, the value of l initializes to 2 to construct a virtual channel between 2 parties over an intermediary. We keep constructing channels for all parties until channel number$(i) + l$ becomes greater than the total parties. If so, we go back to the first party, increment channel length by one, and continue till all channels are constructed. The construct method in the above algorithm uses the perun protocol to construct a virtual channel between parties i and $i+l$. Since all lengths from 1 to $l-1$ will be constructed before constructing channel of length l, intermediary is optimally chosen to be $(i+i+l)/2$ party. Hence, virtual channel is constructed in time Δ between existing channels $(i, (i+i+l)/2)$ and $((i+i+l)/2, i+l)$. So a virtual channel is to be constructed as in [12] between parties i and $i+l$ with party $(i+i+l)/2$ as intermediary. If there are n parties, the two loops will create n^2 channels.

Protocol for n-party Channel creation

To create an off-chain channel, party P_i performs the following procedure:

Channel Creation Initialization Phase

Procedure *channel_initialize*($[P_i, P_n]$, $P_i.id$) is run by P_i on GSCC:

Party P_i runs this instance of GSCC to start the channel creation procedure with Party P_j.
$Channel\_initialize(P_i, P_n, P_i.id) \xleftarrow{\tau_0} P_i$

Procedure *channel_create*($[P_i, P_n]$, $P_i.id$, l, ver) is run by GSCC to P_i and P_j:

The GSCC generates the latest version number(ver) and initial channel length(l) with value 2.
P_i, ver=0
P_j, ver=0
l=2
$Channel\_create(P_i, P_n, P_i.id, l, P_i.ver) \rightarrow P_i$, and $Channel\_create(P_i, P_n, P_i.id, l, P_j.ver) \rightarrow P_j$

Protocol for n-party Channel creation

Local Contract Channel Creation Phase

Procedure *channel_create*($[P_i, P_n]$, $P_i.id, l, ver$) is run by P_i on iMPVSCC:

After obtaining channel creation parameters such as length, participants, version number from GSCC, P_i runs this instance to initiate channel creation. These can be checked by iMPVSCC with GSCC.

$Channel\_create(P_i, P_n, P_i.id, l, P_j.ver) \leftarrow P_i$

Procedure *channel_create_response*($[P_i, P_n]$, $P_j.id, l, ver$) sent by iMPVSCC to P_j:

The iMPVSCC contract awaits for a response from party P_j for creating channel. If party responds, channel creation is done.

$r{=}Channel\_create(P_i, P_n, P_i.id, l, P_j.ver) \xrightarrow{\tau_1 \leq \tau_0 + \Delta} P_j$

if $r == \perp$ then stop. Else,

$Lock\_coins(P_j, \sigma_g(P_j, a)) \leftarrow P_i$

$Lock\_coins(P_i, \sigma_g(P_i, a)) \leftarrow P_j$

$Lock\_coins\_done(P_i, P_j, a) \leftarrow iMPVSCC$

$Lock\_coins\_done(P_j, P_i, a) \leftarrow iMPVSCC$

Channel creation acknowledgement Phase

channel_create_done$([P_i, P_n], P_i.id, P_j, id, l, ver)$ **is run by iMPVSCC on P_i and P_j:**

Funds are locked and channel creation validation is provided by iMPVSCC.

$Channel\_create\_done(P_i, P_n, P_i.id, l, P_i.ver, P_j) \rightarrow P_i$

$Channel\_create\_done(P_i, P_n, P_i.id, l, P_j.ver, P_j) \rightarrow P_j$

channel_create_done$([P_i, P_n], P_i.id, P_j, id, l, ver)$ **is run by P_i on GSCC:**

Party P_i runs channel creation done instance on GSCC. After this, GSCC increases length of virtual channel by one and continues channel creation process with next party.

$l{=}l + 1$

$Channel\_create\_done(P_i, P_n, P_i.id, l, P_i.ver, P_j) \leftarrow P_i$

$Channel\_create(P_i, P_n, l, P_i.ver) \rightarrow P_i$, and $Channel\_create(P_i, P_n, l, P_{j+1}.ver) \rightarrow P_{j+1}$

3.3 Game Registration and Coin Locking Phase

Game Registration Procedure

The following methods are executed on global contract GSCC to register a game between N parties:

Contract Registration Phase

At time t_0, party P_i runs the $game\_register(c, r, n, a)$ method of GSCC contract.
GSCC waits till time $t_1 = t_0+\Delta$ for all N parties to respond with the set of opponents.
Party P_i sends response as r_i=(P_a, P_b) to GSCC. σ_g is the key of GSCC.
if r.length()<=c and P_i not in r then,
message = (P_a,$\sigma_g(P_a$,a))+','+(P_b,$\sigma_g(P_b$,b))
return message to P_i
else,
message = 'invalid response'
return message to P_i
if P_i receives valid message from GSCC, P_i splits message at ',' and provides each tuple (P_a,$\sigma_g(P_a$,a)) and (P_b,$\sigma_g(P_b$,a)) to their respective channel, ie., channel contract (P_i,P_a) and (P_i,P_b).
$Lock\_coins(P_a, \sigma_g(P_a, a) \rightarrow iMPVSCC$
Channel contract will be able to decrypt using GSCC's key to obtain value 'a' and lock 'a' coins in that channel.
$Lock\_coins\_done(P_i, P_a, a) \leftarrow iMPVSCC$

This section will discuss the game registration and coin locking procedure in detail. This phase happens during the local channel creation phase, where funds are generated and locked, as discussed in the last section. While constructing an n-party channel, any party P_i can execute the $game\_register(c, r, n, a)$ procedure of the GSCC contract. There are four values passed in the function: the concurrency c, number of rounds r, number of parties n, and the amount for each game a. For this game to have sufficient funds in each channel, every party must lock at least $r * a$ coins in each of their $n - 1$ channels. The value $c * r$ is the total number of games a party P_i can play. In each of the rounds, a party can play concurrently c different players. c is always less than or equal to $n - 1$, since a party cannot play against more than $n - 1$ players at a time. However, in one channel, each party can lose a coins in each of the r rounds. All n parties agree upon these parameters before the party P_i executes this on the global contract.

$$T_{coins} = r * a * (n - 1)$$

In the above equation,the total number of coins (T_{coins}) deposit in all off-chain channels by one party.

After this execution, the GSCC provides all the n parties at most Δ time to respond with a list of players that they would be playing against(The number of players have to be less than or equal to the value c that was passed at the time of function call) and a zero-knowledge proof that they have sufficient number of coins in their main blockchain account. The global contract waits for Δ time and verifies the values received from all parties. If no party responds to the GSCC, they are considered not playing in that round. In the next step, the global contract maintains an append-only contract board which consists of all possible pairwise matches between n parties.

The board can only append by the GSCC. The values added to the board cannot alter, even by the GSCC. It ensures that the outcome of all the matches describes precisely once. After receiving the list from all the parties, the GSCC marks all the pairwise matches not taking place in that round as 'No Result' (N); after the outcomes of each game are obtained, the append board will mark it as 'W' for a win and 'L' for a loss.

Further, the GSCC sends a tuple to each party with two fields: the names of the virtual contract and the number of coins in that contract, encrypted with its key, σ_g. Each party

provides this tuple to each of its payment channel contracts by running the $lock\_coin$ procedure, as shown in Fig. 7. The contract will use σ_g to decrypt and get the amount. Each channel will lock coins. After all coins are locked, each party receives a $Lock\_coins\_done$ message from their channel contract. All parties will submit their confirmation messages to the GSCC to start playing the game.

3.4 Execution Procedure

One game consists of two players, where each player chooses to make their first valid move. Once a player P_i makes his move, he signs the new state with his signature σ_i, and sets the initial version number ver as 0. P_i sends the signed state and version number to the party P_j, who signs the instance with his signature σ_j. Signing a state by party P_j means that he agrees upon the move made by P_i, making it a valid executable state. Now, party P_j makes his move, signs the new state with his signature, sets the next version number as $ver + 1$, and sends both of it to P_i.

coins for the winner and loser of that game. If these two parties behave maliciously and continue to play a new game in the same round, the append-only board prevents them from validating this new game. After all the games in a round are completed, the next round starts, and the board is cleared. Concurrency and multiple rounds of games give all participants the flexibility to play during a particular round.

Game execution example: Consider a scenario where four parties P_1, P_2, P_3 and P_4 decide to play a two-player game tournament with each other simultaneously and concurrently. All parties form a chain of ledger state channels connecting each with their neighbor. One party belonging to a channel deploys SCC on the ledger. All players agree upon basic parameters such as the number of rounds, number of concurrent games per player, and the amount to be locked per game. Player P_1 deploys GSCC on the ledger to start the channel creation, game registration, and coin locking procedure. All parties have to cooperate with the GSCC to complete the above three processes by responding to the GSCC contract within each round of information exchange.

Protocol for n-party channel execution

The function explains the execution of contract between two parties P_i and P_j. Let s and c be the current state number and contract instance respectively.

Contract Execution

c_{s+1}=$Update\_instance(c_s) \leftarrow P_i$
$\sigma_i(c_{s+1}) \leftarrow P_i$
$(c_{s+1}, \sigma_i(c_{s+1})) \rightarrow P_j$
$\sigma_j(\sigma_i(c_{s+1})) \leftarrow P_j$ (State s+1 becomes valid since it has signature from both parties)
c_{s+2}=$Update\_instance(c_{s+1}) \leftarrow P_j$
$\sigma_j(c_{s+2}) \leftarrow P_j$
$(c_{s+2}, \sigma_j(c_{s+2})) \rightarrow P_i$

The above process continues until one party is unable to make the next move. When the contract execution halts, the GSCC resolves the dispute and validates final instance.

This process repeats until one party wins, making the other party unable to send the next valid state of the game, as shown in Fig. 8. There is a dispute when one party refuses to respond to another party's move or a party does not reply within Δ time. Details of the dispute handling process by the GSCC will discuss in the next section. The GSCC settles the dispute and assigns the winner and loser to the append-only board, which finishes that game between the two parties. The append-only board ensures that no one, including the GSCC, can modify the result of a completed game. The GSCC creates the latest valid state of channel balances by updating the

All parties decide the four initial values as 2,6,3 and 0.1, which are the number of concurrent games (c), rounds (r), parties ($n-1$)(since a party cannot play with itself), and amount for each game(a) respectively. Each party locks a total amount equal to the product of three terms. In the worst case, any party can play a maximum of two parties concurrently for all six rounds and lose every match. Each channel has to have sufficient coins locked in it in this scenario. Therefore, each party locks a total of 6*3*0.1, which is 1.8. Each

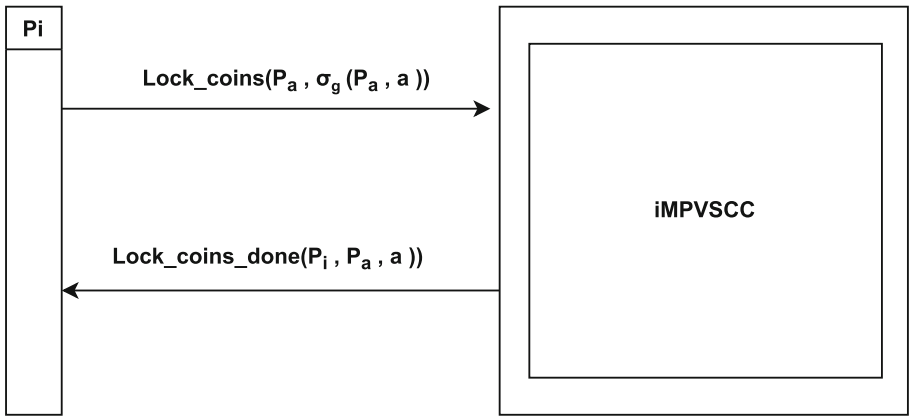

Fig. 7 iMPVSCC contract for locking coins after game registration

party has to lock a 0.6 value in each of the three channels. Figure 9 shows the initial number of coins locked by each party in their respective channels and the final amount after completing their game. In this scenario, we have shown the first round where party P_1 playing against P_2 and P_3, party P_2 playing against P_3, and party P_4 is not participating. GSCC marks the games that are not taking place as "No result (N)."

After locking coins, all the games can occur concurrently, and any game can start or stop in their independent channels. In Fig. 9, we assume that party P_1 has won against both P_2 and P_3, party P_2 has won against party P_3. Hence, the final distribution of coins is shown, where the winner gains 0.1 coins, and the loser has to pay that amount. Tables 4 and 5 shows the initial state(left) of append board and final state(right) after completing all the games in that round. If the index (i, j) mark as 'W' by GSCC, the index (j, i) is 'L.' The 'W' at (row 2,column 3) means that party P_1 has won against party P_2. The same convention is followed throughout by the GSCC while marking the result. Let us explore the game execution between parties P_1 and P_2, and they have the initial state of the contract instance c_0. Assume that P_1 makes the first move.

1. Party P_1 updates the state of contract to c_1.
2. P_1 signs c_1 with its key σ_1 to result in $\sigma_1(c_1)$.
3. P_1 sends the state c_1 and the signed instance $\sigma_1(c_1)$ to P_2.
4. Party P_2 validates the signed state $\sigma_1(c_1)$ by signing with his key σ_2, resulting in $\sigma_2(\sigma_1(c_1))$. This makes instance c_1 valid.
5. P_2 makes the move in c_1 and generates next state c_2.
6. P_2 signs c_2 with its key σ_2 to result in $\sigma_2(c_2)$.
7. P_2 sends the state c_2 and the signed instance $\sigma_2(c_2)$ to P_1.

The above steps continue until one party cannot make his next move or if someone refuses to sign an instance. In this case, any party can raise a dispute with GSCC, which solves

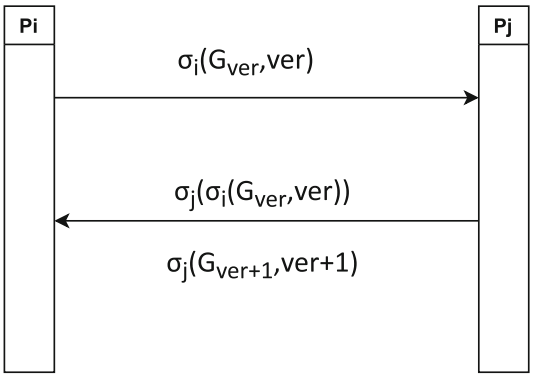

Fig. 8 Execution process of two parties exchanging signed versions of current and next instances

it in constant time. Once the dispute is solved, the game between those two parties ends. GSCC updates the game's result on the append board and provides the parties with the final contract instance with updated balances after completion of their game. The parties go to their channel contract iMPVSCC to set the balance. The final balance calculation and fund release happen after all games and rounds finish.

Other examples: The model also performs as a payment mechanism between n parties. Instead of sending the number of rounds and concurrency to the global contract, party P_1 sends value zero in both fields, indicating that the parties want to make payments. Parties decide the coins per channel. Therefore, the GSCC does not need to maintain an append board. The GSCC sets up each channel to lock the number of coins that parties decide, and the remaining registration and coin locking procedure remains the same. If party P_1 wants to make a payment to P_2, the party P_1 can create a new contract instance by updating the current balances of both parties. Further P_1 signs with his key and sends it to P_2. If P_2 agrees with the new state, he can sign with his key, making that instance valid. In this manner, P_1 and P_2 make many payments to each other using their state or virtual channel.

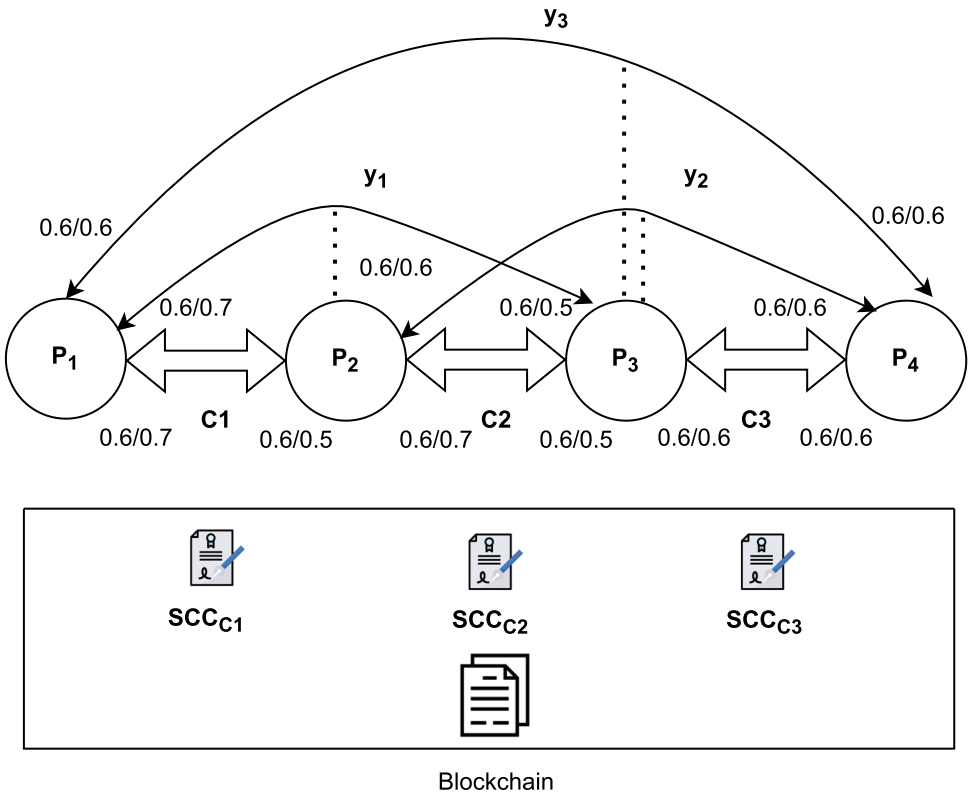

Fig. 9 Four player system explained using the state and virtual channels

To explore the payment application, consider the four initial values as 0.0,3 and 2, which are the number of concurrent games (c), rounds (r), parties ($n-1$) (since a party is connected to $n-1$ neighbours), and amount to be stored in each channel (a) respectively. The parties take help from GSCC to lock 'a' coins in each $n-1$ channels that a party is connected to. The coin locking phase is performed with the help of intermediary contract iMPVSCC. Once all parties lock coins, they are free to update balances of the channel any number of times. Parties obtain initial state of contract with 'a' coins from iMPVSCC, named c_1. If party P_1 wants to send 'x' coins to P_2, the below procedure is followed.

1. Party P_1 updates its balance from a coins to $a-x$, changing state from c_0 to c_1.
2. P1 signs c_1 with its key σ_1 to result in $\sigma_1(c_1)$.
3. P_1 sends the state c_1 and the signed instance $\sigma_1(c_1)$ to P_2.
4. Party P_2 checks if state c_1 balance is greater than or equal to zero.
5. If so, P_2 validates the signed state $\sigma_1(c_1)$ by signing with his key σ_2, resulting in $\sigma_2(\sigma_1(c_1))$. This makes instance c_1 valid.

Any of the two parties will initiate the state update at a given point of time. The above process continues until any party refuses to respond to a state change request from the other party. A party invokes the GSCC to resolve the issue. The GSCC asks for latest state from both parties. Initially, it verifies if the total balances of the valid state are equal to the total amount locked in the channel by the two parties. The GSCC has authority to verify this since all parties and channels register with it during registration phase. If it is valid, the GSCC validates the party's instance that has the higher version number. The parties provide the final valid instance obtained from GSCC to the channel contract iMPVSCC, which releases the funds and updates the final on-chain balances.

Table 4 Initial state of append board

	$P1$	$P2$	$P3$	$P4$
$P1$	N			N
$P2$		N		N
$P3$			N	N
$P4$	N	N	N	N

Table 5 Final state of append board

	$P1$	$P2$	$P3$	$P4$
$P1$	N	W	W	N
$P2$	L	N	W	N
$P3$	L	L	N	N
$P4$	N	N	N	N

Dispute Handling: A dispute arises when an instance of a contract does not have the signature or approval of both parties. In our model, this scenario arise because of two reasons:

1. A party P_j does not respond to the state change signed by the other party P_i in Δ time.
2. A party P_j cannot make its next valid move because of the game ending in that party's defeat.

We consider the above scenarios when the game comes to an end. After waiting for Δ time, the party P_i raises a dispute with the GSCC by running the $dispute\_handle(c_i, zkp, c_i.ver)$ method. While invoking the method, the party P_i has to provide the latest instance, version number and a zero-knowledge proof, proving that the instance execution is done by P_i and P_j. The GSCC checks the validity of the instance by verifying the signatures and proof provided.

If the instance is valid, the contract provides party P_j with Δ time to respond with its own latest valid instance, version, and zero-knowledge proofs. After obtaining and validating information from both parties, the GSCC signs the correct instance, thereby solving the dispute. The GSCC returns the correct instance to both parties. Figure 10 shows the dispute handling process execution sequence, with party P_i initiating the dispute with party P_j. Finally, the global contract updates the append board. Party P_i initiates the dispute with the GSCC in one round, and the contract waits for one more round for the other party to respond. In the optimistic case, when the other party responds before timeout, the dispute process requires 1 round with a time of Δ. In the pessimistic case, the contract waits for another round, making the total time as 2Δ. The time complexity of the dispute solving process is $O(\Delta)$. Once the dispute in a channel is solved, the GSCC creates the final contract instance for that channel and shares the data with both parties. The parties then go to the channel contract iMPVSCC to settle the final balances.

3.5 Channel Closing Procedure

The final stage is when all games and rounds are complete. Any party P_i initiates the channel closing method of the GSCC by passing the party number of each of the participants, shown in Fig. 11. The GSCC completes the validation of the last states of each game and provides this valid contract instance to all parties when their game ends. GSCC checks each game by navigating through the append board. If the game is complete, $unlock\_coins(P_i....P_n)$ is invoked.

Protocol for n-party channel closing

The function explains the working of GSCC in closing channel between two parties P_i and P_j.

Channel closing

Upon $Channel\_close(P_i, ..., P_n) \leftarrow P_i$

For i from 1 to n

For j from 1 to n

if $board[i][j] == \perp$, then stop.

Else,

$\sigma_g(unlock\_coins(P_i, ..., P_n)) \rightarrow P_i$

P_i sends $\sigma_g(unlock\_coins(P_i, ..., P_n)) \rightarrow iMPVSCC$

$\sigma_g(unlock\_coins(P_i, ..., P_n)) \rightarrow P_j$

P_j sends $\sigma_g(unlock\_coins(P_i, ..., P_n)) \rightarrow iMPVSCC$

$Channel\_close\_done(P_i, ..., P_n) \rightarrow P_i$

P_i sends $Channel\_close\_done(P_i, ..., P_n) \rightarrow iMPVSCC$

The two loops navigate through every game. GSCC ensures that only valid game participants can unlock their channel coins. In final step, GSCC acknowledges that channel is closed,

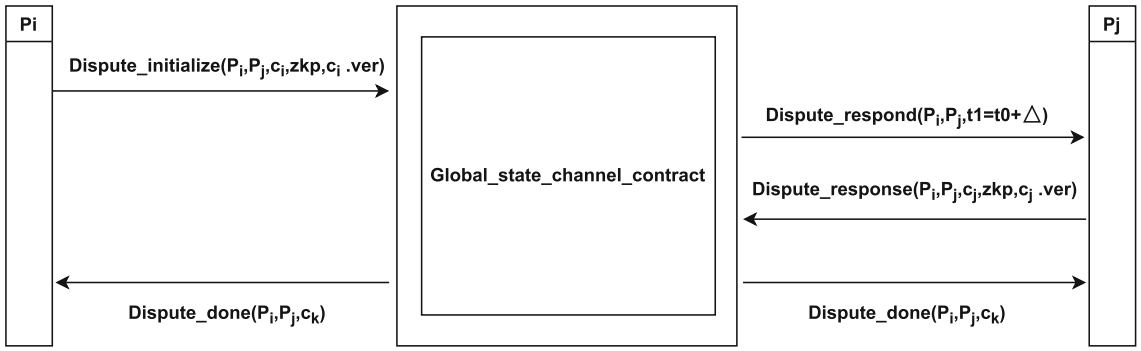

Fig. 10 GSCC dispute handling mechanism

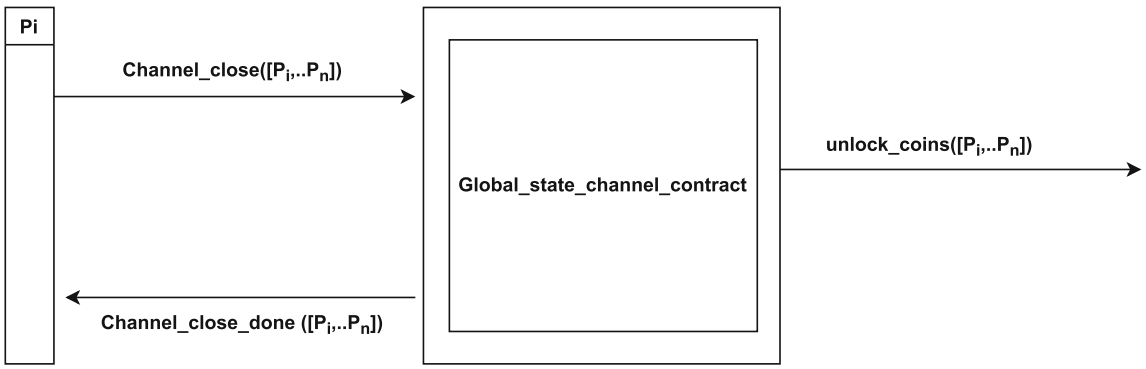

Fig. 11 GSCC channel closing mechanism

The parties provide their channel contract iMPVSCC with final state received from GSCC. iMPVSCC executes the final instances of each channel, releasing all the locked coins to their respective parties. After checking every channel, the $Channel\_close\_done(P_i, ..., P_n)$ method is run by GSCC to acknowledge process completion.

3.6 Working of GSCC

In this section, we explore the functioning of GSCC in resolving dispute. GSCC acts as the entity that resolves all disputes in the n-party network. A party P_i initializes a dispute with GSCC by running the method $Dispute\_initialize(P_i, P_j, c_i, zkp, c_i.ver)$ at time τ_0. In function initialize, party P_i provides GSCC with its latest instance and proof of its blockchain account. If the $zkp\_function(P_i)$ returns a true response, GSCC accepts it, otherwise rejects the dispute initialization. GSCC requests other party to provide proof of their account and their latest valid instance in τ_1 time, where $\tau_1 \leq \tau_0 + \Delta$. GSCC provides fixed time Δ for parties to give response to requests. If a party is unable to provide proof of their account, GSCC marks it as malicious. A party that does not respond with its instance in given time is also treated as a malicious party. If the $zkp\_function(P_j)$ returns a true response, GSCC accepts it. GSCC checks which party has a valid instance number that is larger. It assigns c_k as c_i if $c_i.ver > c_j.ver$. Otherwise, it assigns c_k as c_j. Finally, GSCC runs the $Dispute\_finalize(P_i, P_j)$ method. Both parties P_i and P_j are provided with message $Dispute\_done(P_i, P_j, c_k)$ at a time τ_2. This message contains c_k, which is the final valid state for the channel between P_i and P_j.

GSCC dispute handling

The function explains the working of GSCC in resolving dispute between two parties P_i and P_j.

Dispute handling

Upon $Dispute\_initialize(P_i, P_j, c_i, zkp, c_i.ver) \xleftarrow{\tau_0} P_i$

If $zkp\_function(P_i) \nexists$, then stop.

Send $Dispute\_respond(P_i, P_j) \xrightarrow{\tau_1 \leq \tau_0 + \Delta} P_j$

Upon $Dispute\_response(P_i, P_j, c_j, zkp, c_j.ver) \xleftarrow{\tau_1} P_j$

If $zkp\_function(P_j) \neq 1$, then stop.

If $c_i.ver > c_j.ver$, then c_k=c_i. else,c_k=c_j

Upon $Dispute\_finalize(P_i, P_j)$

Send $Dispute\_done(P_i, P_j, c_k) \xrightarrow{\tau_2} P_i$

Send $Dispute\_done(P_i, P_j, c_k) \xrightarrow{\tau_2} P_j$

4 Security Analysis and Efficiency Goals

In this section, we explore the security and efficiency goals of the proposed model. We discuss all the security measures and present some theorems to prove them.

4.1 Consensus on Channel Creation

In the previous section, we explore the role of the GSCC. The GSCC helps create a channel of length two starting from the first party and increments the length by 1 to repeat the process. Each party plays a role in channel creation with the iMPVSCC contract after obtaining parameters from GSCC. It proves that the entire multiparty model cannot construct without attaining consensus.

4.2 Dispute Handling with Corrupt Parties

The dispute handling process described previously has given each party the opportunity to prove that they have the correct state of the contract. The global contract performs this process by providing Δ time to both parties to respond with their latest valid instance.

4.3 Consensus in Execution and Contract Updation

A party updates the state of a contract and signs with his private key. The other participating party validates this locally and signs with his key, if validation is successful. A contract is valid after achieving signature from both parties, which can be verified by intermediary party using the virtual state contract. Hence, consensus achieve in contract execution and updation.

4.4 Concurrency in Execution of Contract Methods

The channel construction algorithm ensures that while constructing a virtual channel of length i between parties P_i and P_j, all channels from length 1 to $i-1$ exist between them. There exists a channel between each pair of parties. The execution of one contract does not impact the execution of another one simultaneously. Multiple parties' contract instances work simultaneously without affecting the system performance. This section will explore some theorems and their proof and analyze their complete description.

Theorem 1 *If there are n participating parties from 1 to n in constructing a channel, every pair of parties i and j are connected by a virtual channel built over the party then the intermediary must be (i+j)/2.*

Proof The n-party channel between parties 1 to n is constructed after attaining consensus from all participants. There also exists a state channel between all parties before construction begins. The party numbered 1 sends the initial request to global contract GSCC. The GSCC generates parameters for the virtual channel to be constructed between parties 1 and 3 with the help of local contract iMPVSCC. Following this, the GSCC moves on to build a virtual channel between parties 2 and 4. The GSCC continues the process until all parties up to n have constructed a channel length of 2. The GSCC now returns to party 1 and generates the parameters to build a channel between 1 and 4. The exact process continues until all pairs of parties have a channel connecting them.

Therefore, this proves that all channels between i and j can be built with the existing channel over party $(i + j)/2$.

A three-party channel construction protocol is described below. We assume that all parties P_1,P_2,P_3 are connected by two ledger state channels as explained above. n is the number of parties,which is 3. p is the party number, i is the intermediary. The below mechanism creates virtual channels.

The three party channel creation takes place as follows:

1. Initially, Party P_1 sends creation initialization procedure to GSCC contract.
2. GSCC upon receiving ($create\_initialize$,P_1), runs $generate\_parameters$, where parameters such as ver and i are calculated.

_generate_parameters_ function description:

initialize static variable l as 2
input : (N,l,p)
if p+l==N, then,
    set p as 1
    Increment l by 1
if p+l>N, then,
    exit
i=ceil[(p+l)/2]
end function

3. GSCC sends $channel\_create(p, l, ver, i, p + l)$ to P_1.
4. P_1 upon receiving $channel\_create$ from GSCC, sends it to intermediary P_i
5. P_i upon receiving $channel\_create$ from party p, waits for Δ round for response from party p+l.
6. Party p+l sends $create\_channel\_ok$ response to party i.
7. The three parties p,i and p+l lock funds on each virtual channel to open it.
8. Party i sends $channel\_create\_done(P_1, l, ver)$ to GSCC.
9. GSCC runs $generate\_parameters(P_1, l, ver)$

The process continues until all channels are constructed between n parties. The $generate\_parameters$ method calculates and assigns intermediary. The method terminates creation process when the party number p+l exceeds total number of parties P. □

Theorem 2 *If a party P_i tries to execute an invalid contract state G with version number w on the blockchain, it gets discarded.*

Proof All disputes are solved using the GSCC deployed by the parties on the blockchain. If two parties have a dispute, the GSCC invoke by the party raising the dispute. Then, the GSCC starts a procedure that gives Δ time to both parties to provide their latest contract version. The contract validates one of the party's contracts, making them the winner of the game. The contract on the append-only board appends the result. A malicious party cannot change the game's outcome by running an older instance of the contract at any later instance. It is because all older instances do not have a valid signature from the GSCC.

Consider a party P_i with contract instance G_v and another party P_j with contract instance G_w.

1. Party P_i wants to execute its instance G_v on the blockchain at a certain point.
2. The blockchain recognizes that the contract is from an off-chain channel and waits for response from other parties.
3. Party P_j sends its contract instance G_w to the ledger.
4. The blockchain runs the instances on the ledger.
5. The most recent instance of the contract will achieve consensus from other participants.
5. Finally, the blockchain creates block for the valid instance and discards the other ones.

The involvement of blockchain in this process increases the overall time of solving the validity of contract. □

Theorem 3 *If an n-party system is constructed off-chain, each member has to be able to prove their legitimacy to any other party by using their own secret number or value.*

Proof Security is an important feature of blockchain, since it ensures that malicious entities cannot corrupt sensitive data. We have used the universal composability (UC) model to prove the channels and contracts are secure. Canetti et al. [34] explores the UC model for two-party and multiparty systems using zero-knowledge proof mechanism. Their protocol allows any subset of parties to perform correctly, independent of the activity of the rest of the network.

Consider an N-party system where each party knows a secret value x_i, where i is the party number. Therefore, the secret values are x_1,x_2,...x_n. The idea is to compute and store a value y=f(x_1,x_2,...x_n) in the global contract GSCC after receiving all the x_i values. We will be exploring a three-party system with secrets x_1,x_2, and x_3 for this proof.

1. The $security\_check(P_1, P_2, P_3)$ instance of GSCC is invoked by party P_1.
2. GSCC provides computing function $f(x_1,x_2,...x_n)$ to party P_1 and waits for response.
3. Party P_1 inputs its x_i locally and sends value y to GSCC.
4. GSCC provides computing function $y = f(x_1,x_2,...x_n)$ obtained from P_1 to party P_2 and waits for response.
5. Party P_2 inputs its x_i locally and sends value y to GSCC.
6. GSCC provides computing function $y = f(x_1,x_2,...x_n)$ obtained from P_2 to party P_3 and waits for response.
7. Party P_3 inputs its x_i locally and sends value y to GSCC.

8. GSCC stores the final y value and terminates.
9. GSCC waits for 1 round for each party to respond. Hence, total time complexity for n-party system is $O(n\Delta)$.

The value y cannot alter by any party once it computed and added in the contract. This means each party can prove that they know their secret value x_i each time without revealing what it is. □

4.5 Zero-Knowledge Proof

The zero-knowledge proof verifies the participating parties that they have sufficient coin to involve in the n-party virtual model. Party P_1 wants to prove to GSCC that it knows the secret value x to verify him as he has the required no. of coins. The GSCC is a global contract that knows the party after participating in this model, but it has no information regarding the contract account as the user may have already used some coins. The GSCC chooses one miner who has the complete details of the contract account of the participating party see Fig. 12. We use the property of the Elliptic curve discrete logarithm to define the zero-knowledge proof and g∈E(F_p). p is a large prime, and the Elliptic curve(E) is the koblitz curve, i.e., secp256k1. The P_1 chooses $x \xleftarrow{\$} \mathbb{Z}_{n^*}$. g is the generator of F_p and n is a prime number. Ethereum uses the SHA-3 hashing algorithm, i.e., Keccak-256; hence in this proof, we use it for blockchain convenience. GSCC verifies the public address with the respective number of coins available.

***zero_knowledgeproof* function description**

P_1 calculates value V=$x.g$, and sends it to P_2.
P_2 generates challenge $c \xleftarrow{\$} \mathbb{Z}_{n^*}$ and sends it to P_1.
P_1 generates value $z \xleftarrow{\$} \mathbb{Z}_{n^*}$ and calculates e=$z - x * c$.
P_1 calculates Y=$e.g$, and h=Keccak-256(Y).
P_1 sends tuple (e, h) to P_2.
P_2 calculates r=$z.g$+$c.V$.
P_2 calculates d=Keccak-256(r).
If d==h, then the party P_1 has proven to party P_2 that it knows its secret value x.

5 Discussion and Results

We analyze the resultant work and all the smart contracts. We use solidity programming language and truffle suite along with Ganache with Metamask. The contracts deploy on the ethereum framework. We discuss the time consumption and contract execution cost for state update and dispute handling process in each two-party game. We assume equal response time from all participants during each transaction in our simulations. We provide a proof of concept representation of update and dispute handling mechanisms. The successful execution of simulations tells us the proposed system's fundamental concepts. Table 6 calculates the gas consumption and time taken in seconds for each transaction between parties P_1 and P_2. Gas cost can be represented in gwei or eth units. The conversion equation is shown below:

$$1\ eth = 10^9\ gwei$$

From the simulation, we can infer that contract gas cost and execution time varies for update and dispute transaction types. Table 7 divides update and dispute transactions to calculate efficiency for each type. The efficiency of a transaction is calculated using the below equation.

$$\text{Efficiency} = \frac{\text{Gas Cost}}{\text{Execution Time}}$$

Figure 13 compares the execution times for update and dispute transactions. In this scenario, two parties perform transactions by transferring coins off-chain using state update. When a dispute happens, a party invokes the GSCC. The GSCC checks with both parties for the latest instance and transfers coins correctly based on the latest valid state. We observe that update transactions consume more time and execution costs than dispute handling. Figure 14 Compares the update and dispute process efficiency for five transactions. The efficiency signifies how much gas costs a particular type of transaction uses per unit of time. In this case, the update process has high efficiency, making it more cost-efficient than the dispute-handling process and comparing the efficiency of update and dispute transactions. Figure 15 shows the total cost of each module. The initialized module is the coin-locking phase. The getBalances module provides a party's current balance at any point. The setBalance module is used to update the state/balances of any party. The DisputeHandle module compares the version numbers from both parties to check the latest one and validates it. The total cost is obtained by adding the cost of each transaction.

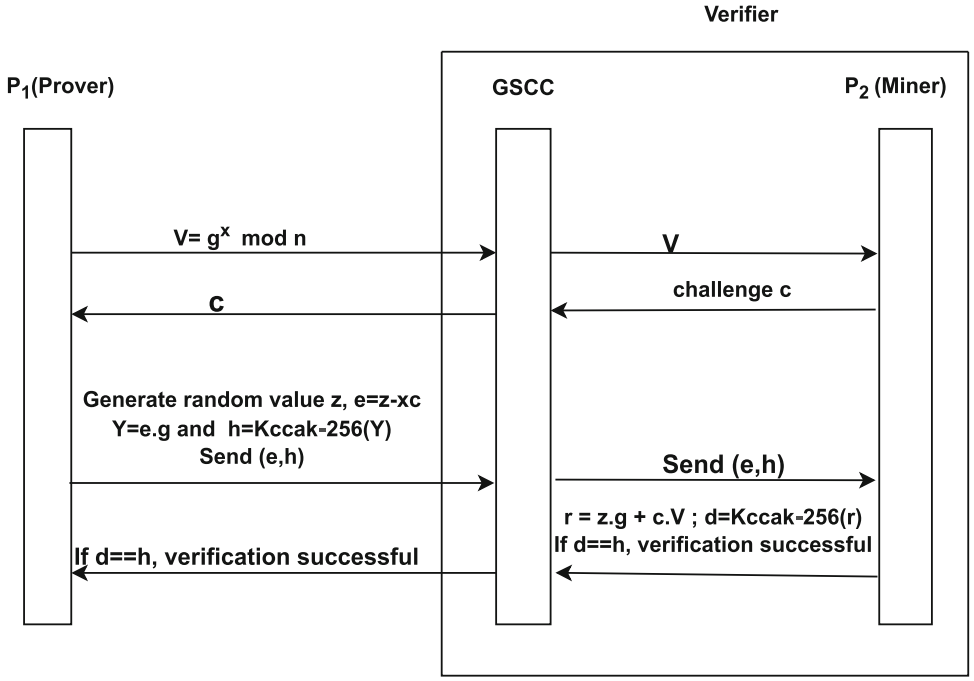

Fig. 12 Zero-knowledge proof mechanism

Table 6 Simulation results 1

Transaction number	Transaction type	Gas cost (gwei)	P_1 initial balance	P_2 initial balance	P_1 updated balance	P_2 updated balance	Time (s)
1	Update	42688	10	10	8	12	3.693621
2	Update	42688	10	10	7	13	3.298053
3	Dispute	21624	10	10	–	–	2.511392
4	Dispute	21624	10	10	–	–	2.013312
5	Dispute	21624	10	10	–	–	2.525194
6	Update	42688	10	10	6	14	3.047172
7	Dispute	21624	10	10	–	–	2.009664
8	Update	42688	10	10	5	15	3.292367
9	Update	42688	10	10	9	11	2.890422
10	Dispute	21624	10	10	–	–	2.156778

Table 7 Simulation results 2

Transaction number	Update cost (gwei)	Dispute cost (gwei)	Update time (s)	Dispute time (s)	Update efficiency (cost/-time)	Dispute efficiency (cost/time)
1	42688	21624	3.693621	2.511392	11557.22257	8610.364292
2	42688	21624	3.298053	2.013312	12943.3941	810740.51116
3	42688	21624	3.047172	2.525194	14009.05495	8563.302463
4	42688	21624	3.292367	2.009664	12965.7477	410760.00764
5	42688	21624	2.890422	2.156778	14768.7777	110026.06666

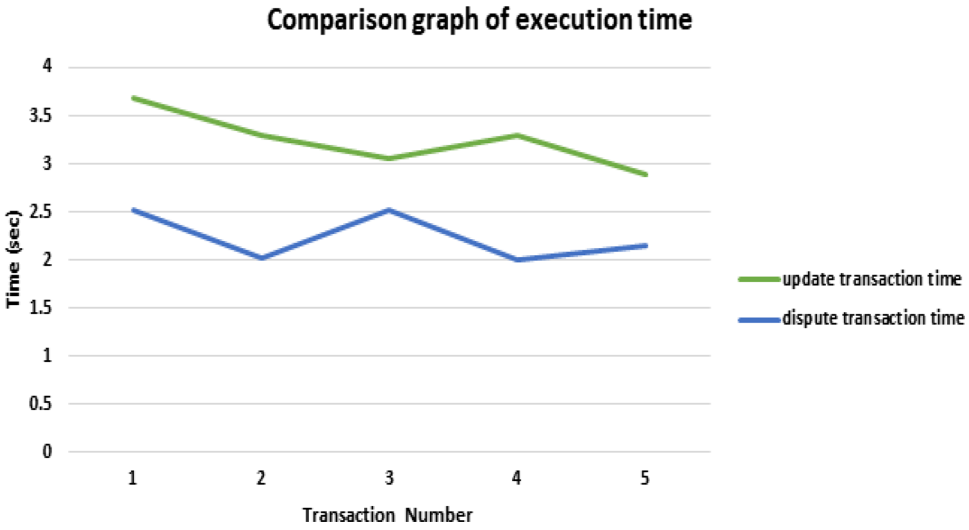

Fig. 13 Comparing update and dispute handling time(sec)

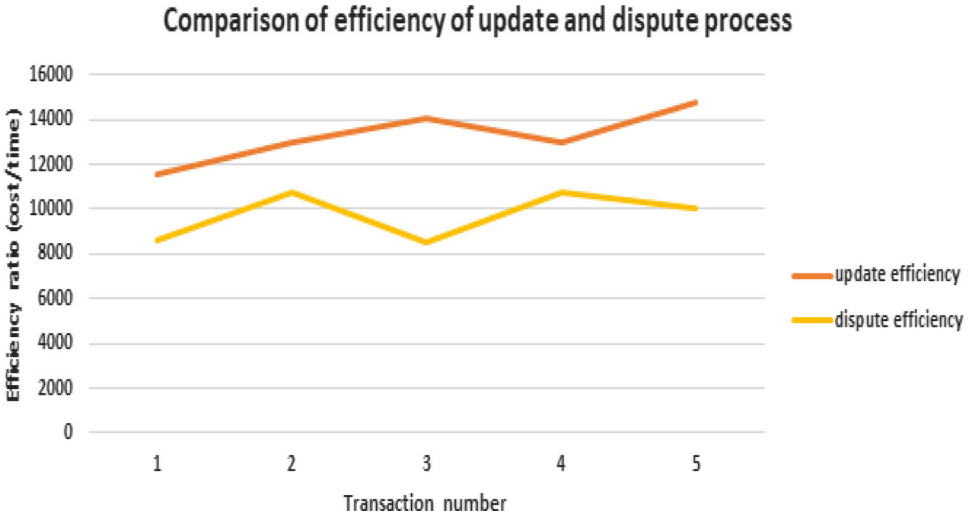

Fig. 14 Comparing efficiency of update and dispute processes

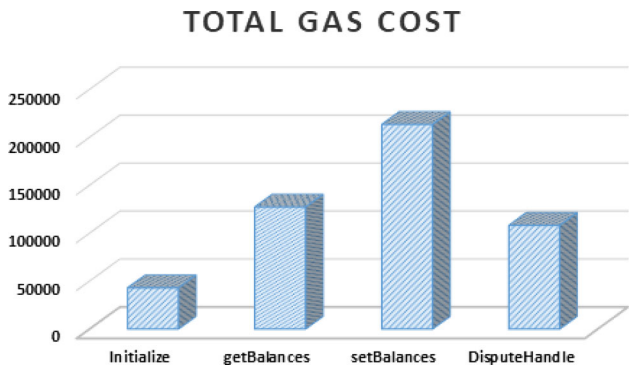

Fig. 15 Gas consumption of each module

Figure 16 shows the execution time for non-concurrent transactions. We assume that four events are happening at the same time. Therefore, the total time for each transaction gets multiplied by four in the case of the non-concurrent method. Our model can conduct state updates/transactions off-chain. The on-chain payment cost and time in the real-time scenario are very high because of consensus requirements, cryptographic challenges, and the involvement of miners. Figure 17 shows the real-time ethereum on-chain average transaction cost. When we compare the cost with Table 6, the update execution cost for our method is much less than the on-chain block creation and consensus cost.

Consider the Fig. 9, where concurrent games between parties (P_1, P_2), (P_2, P_3), (P_3, P_4), (P_1, P_3), and (P_2, P_4). The first

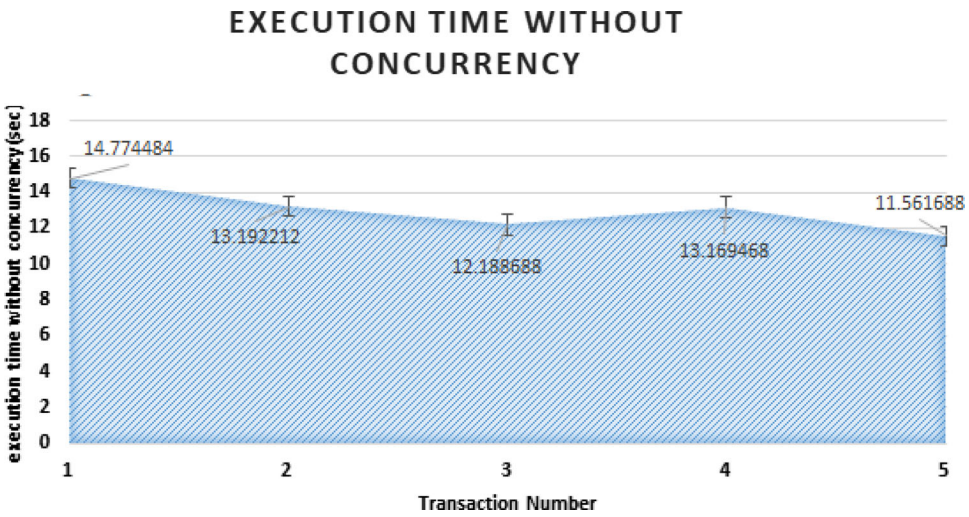

Fig. 16 Execution time of events without concurrency

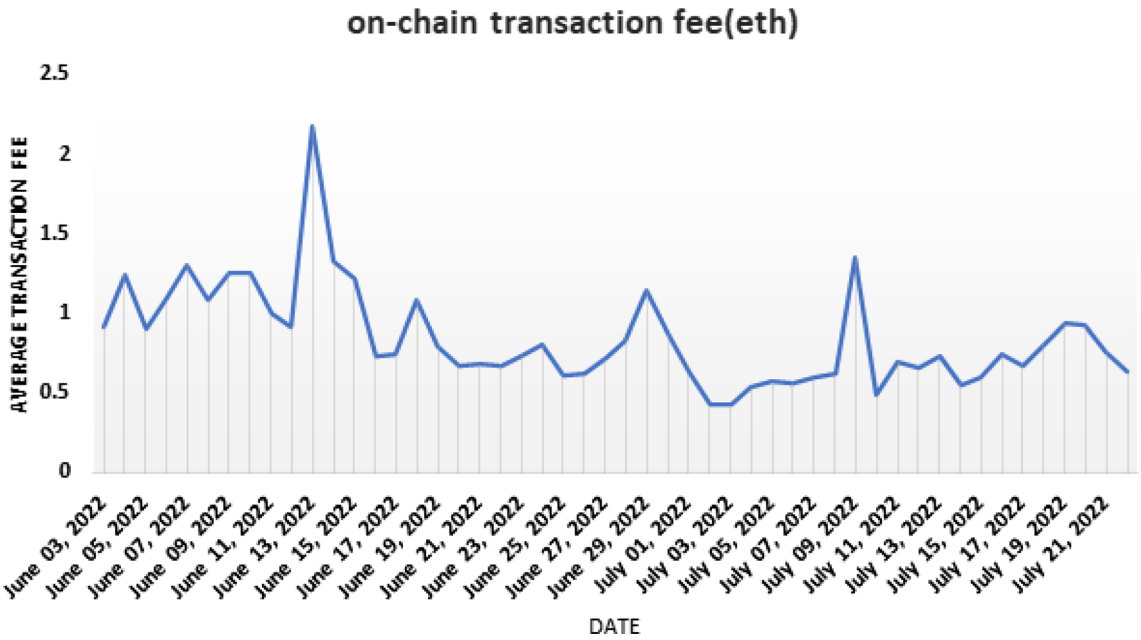

Fig. 17 Real time average transaction cost in ethereum

transaction is an update. Therefore, it indicates that the game between P_1 and P_2 has begun, with party P_1 making a move. During the update, P_1 sends the next state of the game with P_1 signature to P_2. Then, P_2 validates the move by giving its signature and reply to P_1. Transaction 2 is an update by P_2 in the game (P_2,P_3). Here, P_2 makes a move and sends an updated signed instance to P_3 for validation. Transaction 3 is a dispute in the game (P_3,P_4) since the party P_4 did not initiate the game within time. Therefore, the GSCC instance gets triggered by honest party P_3, and the game's result is decided, according to which party committed unfair play. Transaction 4 is a dispute between parties in the game (P_1,P_3), where party P_1 fails to respond in time to P_3. The GSCC gets called again to resolve the dispute by penalizing the unfair player and ending the game. Transaction 5 is an update in the game (P_2,P_4), with party P_4 making the first move. P_4 makes a move, signs the instance, and sends it to P_2 for validation.

After the above transactions are complete, the games (P_3,P_4) and (P_1,P_3) are no longer considered active by the GSCC. Hence, the results update in the append board, and fund distribution steps initiate. From the next transaction, the GSCC considers the remaining three games valid. Since all five transactions are conducted by different games belonging to independent channels, these execute concurrently, resulting in seamless execution and efficiency. Table 8 shows the transaction details for conducting the games, and details such as transaction cost, game details, and current player.

Table 8 Simulation results 3

Transaction number	Transaction type	Transaction cost (gwei)	Game details	Current move party
1	Update	21624	(P_1, P_2)	P_1
2	Update	21624	(P_2, P_3)	P_2
3	Dispute	42688	(P_3, P_4)	P_4
4	Dispute	42688	(P_1, P_3)	P_1
5	Update	21624	(P_2, P_4)	P_4

6 Conclusion

The research conducted in this work has shown that virtual off-chain computations have significant advantages over on-chain or state channel networks since they provide better throughput, scalability, and efficiency without compromising security, integrity, and privacy. The previous model has problems simultaneously providing the same channel to different parties for execution. Our construction solved this problem by giving independent channels to all pairs of parties, allowing concurrent execution. Dispute resolution is linear to the number of intermediaries in more extended virtual channels. The dispute board helped make dispute solving time constant but has an overhead in computation time. The GSCC dispute model described in this work solves disputes in constant time and contains stored data of each channel. It eliminates the communication overhead which exists in the previous method. A detailed view of each protocol construction, dispute handling, execution, and channel closing has also been given. The resultant model ensures the security of participant parties and does not lose the coins as an intermediary or payment initiator. Resolving disputes with intermediary parties before involving the contract provides an additional layer of dispute resolution before the contract execution. Therefore, an indirect dispute model for n-party concurrent execution is a possible future research problem.

References

1. Nakamoto, S.: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin (2009)
2. Buterin, V.: A next-generation smart contract and decentralized application platform. White paper, 3(37) **23** (2014)
3. Blockchain payment network in 2021. https://finance.yahoo.com/news/japan-banking-giant-mufg-plans-090014888.html (2021)
4. Sahoo, S.S.; Menon, A.R.; Chaurasiya, V.K.: Secure blockchain model for vehicles toll collection by GPS tracking: a case study of India. In: 2022 IEEE India Council International Subsections Conference (INDISCON), pp. 1–6. IEEE (2022)
5. Chaurasia, V.K.; Yunus, A.; Singh, M.: An overview of smart city: observation, technologies, challenges and blockchain applications. In: Blockchain Technology for Smart Cities, pp. 133–154 (2020)
6. Jabbar, S.; Lloyd, H.; Hammoudeh, M.; Adebisi, B.; Raza, U.: Blockchain-enabled supply chain: analysis, challenges, and future directions. Multimedia Syst. **27**(4), 787–806 (2021)
7. Tariq, N.; Asim, M.; Al-Obeidat, F.; Zubair Farooqi, M.; Baker, T.; Hammoudeh, M.; Ghafir, I.: The security of big data in fog-enabled iot applications including blockchain: a survey. Sensors **19**(8), 1788 (2019)
8. Hashim, F.; Shuaib, K.; Zaki, N.: Sharding for scalable blockchain networks. SN Comput. Sci. **4**(1), 1–17 (2023)
9. Poon, J.; Dryja, T.: The bitcoin lightning network: scalable off-chain instant payments (2016)
10. Network, R.: What is the Raiden network (2019)
11. Miller, A.; Bentov, I.; Kumaresan, R.; McCorry, P.: Sprites: payment channels that go faster than lightning. CoRR abs/1702.05812 **306** (2017)
12. Dziembowski, S.; Eckey, L.; Faust, S.; Malinowski, D.: Perun: virtual payment channels over cryptographic currencies. IACR Cryptol. ePrint Arch. **2017**, 635 (2017)
13. Dziembowski, S.; Faust, S.; Hostáková, K.: General state channel networks. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 949–966 (2018)
14. Dziembowski, S.; Eckey, L.; Faust, S.; Hesse, J.; Hostáková, K.: Multi-party virtual state channels. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 625–656. Springer (2019)
15. Zamani, M.; Movahedi, M.; Raykova, M.: Rapidchain: scaling blockchain via full sharding. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 931–948 (2018)
16. Wang, G.; Shi, Z.J.; Nixon, M.; Han, S.: Sok: sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, pp. 41–61 (2019)
17. Hao, Z.; Ji, R.; Li, Q.: Fastpay: a secure fast payment method for edge-iot platforms using blockchain. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 410–415. IEEE (2018)
18. Dang, H.; Dinh, T.T.A.; Loghin, D.; Chang, E.C.; Lin, Q.; Ooi, B.C.: Towards scaling blockchain systems via sharding. In: Proceedings of the 2019 International Conference on Management of Data, pp. 123–140 (2019)
19. Eberhardt, J.; Tai, S.: Zokrates-scalable privacy-preserving off-chain computations. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1084–1091. IEEE (2018)
20. Xu, C.; Zhang, C.; Xu, J.; Pei, J.: Slimchain: scaling blockchain transactions through off-chain storage and parallel processing. Proc. VLDB Endow. **14**(11), 2314–2326 (2021)
21. Kumar, R.; Marchang, N.; Tripathi, R.: Distributed off-chain storage of patient diagnostic reports in healthcare system using ipfs and blockchain. In: 2020 International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 1–5. IEEE (2020)
22. Lee, S.; Kim, H.: On the robustness of lightning network in bitcoin. Pervasive Mob. Comput. **61**, 101108 (2020)

23. Zabka, P.; Foerster, K.T.; Schmid, S.; Decker, C.: A centrality analysis of the lightning network. arXiv preprint arXiv:2201.07746 (2022)
24. Waugh, F.; Holz, R.: An empirical study of availability and reliability properties of the bitcoin lightning network. arXiv preprint arXiv:2006.14358 (2020)
25. Zhang, J.; Li, C.G.; You, C.; Qi, X.; Zhang, H.; Guo, J.; Lin, Z.: Self-supervised convolutional subspace clustering network. In: 2019 IEEE CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5468–5477 (2019)
26. Gudgeon, L.; Moreno-Sanchez, P.; Roos, S.; McCorry, P.; Gervais, A.: Sok: off the chain transactions. IACR Cryptol. ePrint Arch. **2019**, 360 (2019)
27. Zhong, L.; Wu, Q.; Xie, J.; Guan, Z.; Qin, B.: A secure large-scale instant payment system based on blockchain. Comput. Secur. **84**, 349–364 (2019)
28. Poon, J.; Buterin, V.: Plasma: scalable autonomous smart contracts. White paper, pp. 1–47 (2017)
29. Tairi, E.; Moreno-Sanchez, P.; Maffei, M.: A 2l: anonymous atomic locks for scalability in payment channel hubs. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1834–1851. IEEE (2021)
30. Walshe, M.; Epiphaniou, G.; Al-Khateeb, H.; Hammoudeh, M.; Katos, V.; Dehghantanha, A.: Non-interactive zero knowledge proofs for the authentication of iot devices in reduced connectivity environments. Ad Hoc Netw. **95**, 101988 (2019)
31. Unal, D.; Hammoudeh, M.; Khan, M.A.; Abuarqoub, A.; Epiphaniou, G.; Hamila, R.: Integration of federated machine learning and blockchain for the provision of secure big data analytics for internet of things. Comput. Secur. **109**, 102393 (2021)
32. Yang, X.; Li, W.: A zero-knowledge-proof-based digital identity management scheme in blockchain. Comput. Secur. **99**, 102050 (2020)
33. Li, W.; Guo, H.; Nejad, M.; Shen, C.C.: Privacy-preserving traffic management: a blockchain and zero-knowledge proof inspired approach. IEEE Access **8**, 181733–181743 (2020)
34. Canetti, R.; Lindell, Y.; Ostrovsky, R.; Sahai, A.: Universally composable two-party and multi-party secure computation. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 494–503 (2002)