IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 17, 2021, accepted November 20, 2021, date of publication November 30, 2021,
date of current version December 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3131419

Blockchain State Channels: A State of the Art

LYDIA D. NEGKA[1] **AND GEORGIOS P. SPATHOULAS**[2]
[1]Department of Computer Science and Biomedical Informatics, University of Thessaly, 351 31 Lamia, Greece
[2]Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), 2802 Gjøvik, Norway

Corresponding author: Georgios P. Spathoulas (georgios.spathoulas@ntnu.no)

This work was supported in part by the European Commission under the Horizon 2020 Programme through the Project LOCARD (Lawful evidence collecting and continuity platform development) under Grant 832735.

ABSTRACT Blockchain technology has been quite popular during recent years and it finally seems to present a significant rise with respect to its use for real-world applications. This advancement has brought up a critical challenge that public blockchain systems face, which is scalability. Most of the currently deployed systems fail to cope with increasing usage. In order to provide the promised security guarantees, large delays and high usage fees are imposed for submitted transactions and thus widespread adoption of the technology is hindered. A number of different approaches have been proposed to increase the capacity of blockchain systems with respect to processing transactions. The present survey focuses on one of the most popular ones, that of state channels, and to the extent of our knowledge constitutes the first collective survey of research in this field. An extensive analysis of relevant publications is conducted and a general view on the domain is provided. We have identified the limitations discussed through all relevant research efforts along with the various features that differentiate proposed designs. A comparison between retrieved papers is carried out on the basis of those limitations and features. Finally, future research directions are analysed while the role of state channels in the general public blockchain ecosystem is also discussed.

INDEX TERMS Blockchain, layer 2, scalability, state channels, survey.

I. INTRODUCTION

It is an undeniable fact that blockchain technology's popularity has recently seen a wild surge [1]. Even though this was predicted, it is still evident that most of the public blockchain systems are not ready for widespread use, since they suffer from very low capacity with respect to transactions' processing rate (the number of transactions that can be processed per second). The first blockchain system to be proposed, Bitcoin [2] has a transaction rate of around 4 tx/sec as of September 2021, while in the case of Ethereum [3], the second most popular blockchain platform, the transaction rate is slightly better, approximately 30 tx/sec. The main idea behind the proof of work blockchain consensus mechanism requires that every transaction is processed by every network node before it is confirmed and published. On top of that, because of security reasons the block generation rate and the upper size limit for blocks are restricted, thus a hard upper limit has to be imposed on the transactions rate. While this approach is efficient in terms of security, it directly clashes with blockchain systems scalability [4], as it hinders the use of such systems by a large number of users.

The associate editor coordinating the review of this manuscript and approving it for publication was Lorenzo Mucchi.

Hence, the matter of addressing scalability issues in blockchain systems has popped up as an issue of unprecedented urgency, since it is causing immense hindering in the widespread application of blockchain technology in multiple fields that could greatly benefit from it. The latency of transactions and the imposed high usage cost (transaction fees) that permissionless blockchains suffer from usually outweigh the benefits of using them and many possible applications become irrelevant.

More than one approach has been explored to remedy those issues. It is a prevailing practice to divide such efforts into two large categories. Layer 1 solutions aim to directly enhance the blockchain functionality and employ alternative protocols [5], [6] or sharding [7], [8]. Those approaches do come with their own set of drawbacks since they require fundamental adaptations of core blockchain systems' components to be applied. Especially for deployed systems that are already in use, the transition to a modified layer 1 is a very challenging process.

Layer 2 approaches are developments on top of an underlying blockchain without requiring significant modifications of that [9]. The most popular proposals in this category are sidechains [10], [11], plasma [12], rollups [13] and state channels. They are generally preferred to Layer 1 solutions,

as they are easier to implement directly on top of existing platforms and they do not come to the expense of fundamental concepts like decentralisation and security.

The focus of this paper is on the State Channels approach [14]. It is one of the leading solutions proposed against the scalability problem and allows the trust-less communication of parties that wish to mutually execute a contract while allowing them to dodge high cost, high latency and any privacy issues that often come up through using a public blockchain system. However, there presently has been no extensive study focusing on this solution. State channels have only been studied along with other Layer 2 solutions and this has led to overlooking their unique characteristics mostly in favour of their close counterparts, Payment Channels. Therefore this survey is solely focused on the State Channels approach and the benefits it can bring when to any existing system by minimizing the transactions' load for any given application.

The main contribution of the present survey is to study, record, analyse and compare all existing state channel efforts, found either in related literature or in the general blockchain ecosystem. The present survey:

- Records all existing state channels approaches
- Analyses how those have eventually contributed to the ecosystem
- Identifies the main requirements those protocols set to operate
- Identifies the main features those protocols offer
- Compares existing approaches
- Sets the main focus points for future research

By making all existing contributions readily available and by providing a concentrated reference point, we aim at facilitating researchers that wish to contribute to this domain, as anyone that wants to be introduced to the state channels' ecosystem will be provided with a good starting point. The commentary and classifications that can be found in this work make it significantly more feasible to pinpoint the weaknesses in the current state of the art and further needs to be addressed in future research.

The rest of this paper is organised as follows: Section II introduces the notion of scalability, how it affects existing blockchain technology and the existing solutions. Section III presents an overview of the State Channel concept as well as an abstract definition. Section IV organises all the included proposals in subcategories and present a detailed analysis of each one. Section V contains comparative reviews of the aforementioned designs across different factors. Finally, in Section VI all conclusions and results from the present survey are discussed.

II. BLOCKCHAIN SCALABILITY

A. BLOCKCHAIN SCALABILITY PROBLEM

The unforeseen popularity of blockchain technology has highlighted the major scalability issues of the original design. Public blockchain platforms have been unable to cope with the exponential increase of transactions, that has been triggered by the increase in their usage. A limitation known as the blockchain trilemma [15], has been extensively discussed in recent years. Three desirable properties for a blockchain system are security, decentralization and scalability, but it is very difficult, at least given existing approaches, to maximise all three of those for a given system. Various factors combine to cause major issues and current systems have to prioritise on at most two of the mentioned properties and make a compromise with respect to the level on which the third property is supported. The common approach that has been followed in the ecosystem is to retain a high level of decentralization and security and limit the scalability of the public blockchain systems.

The problem stems from the root of the design. The data verification process that gives blockchain the robustness it is famous for allows the network to be only as fast as its slowest node, since blocks need to be propagated at the very least to the vast majority of nodes before the generation of the next block.

Additionally, the consensus mechanism most commonly used, called Proof of Work (PoW), with the intensive, expensive calculating process it demands from the miners, plays a big role in terms of network latency. A block can contain a finite number of transactions, and miners will prefer to include in their prospective blocks transactions that offer higher fees. Hence, those that do not provide as high a profit get pushed back in this transaction confirmation process. The obvious solution to this matter, increasing block size, is not optimal in many aspects. Beyond being just a temporary measure that would have to be re-adopted every time latency shows up again, a bigger block size also decreases decentralization and increases the chance of forks occurring in the chain, since block size affects block propagation time. Another action with the same goal that has been proposed is to reduce the size of transactions so that more of them can fit in a block. Such approaches have been implemented in Bitcoin Unlimited [16] and by Bitcoin's Segregated Witness [17], but neither of those methods has brought significant improvements to the situation.

B. BLOCKCHAIN SCALABILITY MEASURES

Various approaches have been proposed to increase the capacity of existing blockchain systems with respect to transactions processing. Those are categorised in two broad categories as layer 1 or layer 2 solutions.

1) LAYER 1

Proposals on this layer refer to fundamental alterations to the design of the blockchain system under consideration. A change in the consensus mechanism [18], [19], like the case of Ethereum 2.0 replacing Proof of Work with Proof of Stake, can measurably improve scalability, but changes of this scale are risky and difficult to implement and enforce. Moreover, the alternative consensus mechanisms may come with compromises with respect to security or decentralization.

Another popular approach on the same level is sharding [20]–[22]. This concept refers to replacing the requirement for all peers to sequentially process transactions. Instead, it suggests breaking up the blockchain network into individual segments (or shards). Each shard would hold a unique set of smart contracts and account balances. Nodes are assigned to individual shards to verify transactions and operations, instead of each being responsible for verifying every transaction on the entire network, while secure communication between shards is enabled to provide a global view of all data.

2) LAYER 2

Solutions that do not require fundamental changes to blockchain systems' design, but rather enable increased scalability through supporting mechanisms that function on top of the systems' chain are broadly labelled as layer 2 scalability solutions [23].

a: SIDECHAINS

Sidechains are schemes on which additional chains operate in a parallel and in an independent way to the main-chain. They do not have to share the main-chain's consensus protocol but enable secure assets transfer back and forth to the main-chain. Even though sidechains are a tried and tested approach, they demand a trade-off between scalability and security.

b: PLASMA

Plasma is an Ethereum specific solution that can be described as a separate blockchain that operates beside it. The plasma chain is secured through periodic commits of its state to the main-chain. It has increased transaction throughput and reduced costs, but is limited in what transactions it supports, as it is not compatible with generic smart contracts. Additionally, it brings on other kinds of delays to enable challenges and ensure fund security.

c: ROLLUPS

Rollups enable transactions to be executed off-chain and have their data stored on-chain. They require a stake deposit and a contract that can monitor off-chain execution when need be, and inherit the security guarantees of the underlying blockchain. They are further divided into two subcategories: Zero Knowledge (ZK) and Optimistic Rollups. Their main difference lays in the transaction validation process. ZK rollups require each one to be accompanied by a fraud-proof bond, while Optimistic rollups assume all transactions are valid until a challenge is issued. ZK rollups offer instant finality, but demand intense calculations for the validity proofs and are not always EVM compatible. Optimistic rollups are prone to delays due to challenges but offer full smart contract support.

d: PAYMENT AND STATE CHANNELS

Payment and State channels are based on the same fundamental concept, performing the majority of transactions completely off-chain in the case of no malicious activity. Payment channels apply this logic for the purpose of conducting payments and materialize an unlimited series of such through a pair of deposit and withdrawal transactions. State channels extend this functionality to include the execution of code and more arbitrary state transitions. Both require an initial fund deposit from participants to operate, and in the optimistic case communicate with the blockchain system only during the channel's opening and closing. The present survey focuses on existing approaches for state channels, the general concept of which is sufficiently described in the following Section.

III. STATE CHANNELS

A. STATE CHANNELS OVERVIEW

The main reason behind the aforementioned scalability issues of blockchain systems is the fact that each update of the state of a smart-contract is required to go on-chain and be executed by all nodes of the network, even if it is relevant to a limited subset of users. State channels effectively counter this phenomenon, by limiting the on-chain operations through the means of a session of off-chain interaction between the interested users, and thus reduce the corresponding overhead.

State channels were first mentioned by Coleman [24], but the concept was first tested in a chess game implementation around the same time [25]. Even though they have proven to be an undoubtedly useful mechanism for complex interactions on the blockchain, they have been greatly inspired by the simpler idea of payment channels [26], that only focus on payments. Payment channels were the first approach that introduced the idea of keeping blockchain users' interaction off-chain, when this is feasible without undermining the security of the said interaction. Payment channels are confining in terms of possible use cases, as they emerged to solve Bitcoin's significant scalability problem [27], and only focus on conducting off-chain payments. Payment channels have been successful in general [28], and have been applied to other blockchain platforms with different functional requirements (e.g. Raiden in Ethereum [29]). Despite the restricted application domain of payment channels, they are one of the main focus points of current blockchain scalability research. As a result, they have been already analysed much more extensively [30]–[32] than state channels, their more broadly applicable counterparts. The present analysis focuses on state channels but it has to be noted that any advancement observed in the payment channel domain does indirectly benefit state channels, since many ideas can be carried over from payment channels and adjusted to the state channels' requirements.

State channels approach is essentially the extension of the payment channels approach, to go from simple payments to arbitrary state transitions. This extension aims at allowing the off-chain execution of smart contracts of generic functionality (without interacting with the blockchain system) while still reaping all of its security guarantees.

When two or more users interact with a smart contract, they sequentially update the contract's state. The underlying

blockchain guarantees that the content of those updates (also noted as state transitions) is common for all participants and that the order under which those transitions are submitted is strictly defined. Because of the consensus mechanism used, it is not feasible (or it is at least extremely hard in practice) for any user to claim that a state transition has not been applied, that a state transition has different content or that past state transitions should be in a different order than the one those actually appeared in.

The main idea behind state channels is to achieve the same security guarantees but at the same time minimise the number of the required on-chain transactions. In practice, there are two distinct cases with respect to state channels operation; (a) the optimistic one in which users do not intend to cheat others and off-chain exchanging of state updates between users is sufficient and (b) the pessimistic one in which there are users that may attempt to cheat others and so the communication with the blockchain system is required to resolve disputes over malicious users' behaviour.

State channels aim at reducing to minimum on-chain interactions between participants, but they do have a great dependency on the liveness of the underlying blockchain system, since, to remain secure, they have to assume any transaction happening on the channel can always be published on-chain when required. The blockchain system is necessary to take on the role of a trusted arbitrator that will function as a guarantee for honest parties and settle any occurring disputes. To that end, state channel protocols require participants that want to collaboratively execute an app, to bind (lock) a predetermined set of assets (funds, tokens) on-chain. Those assets can be retrieved (with a different distribution for users) through the finalisation of the state channel. Through this constraint, the state channel can satisfy the requirement for establishing trust between users and it keeps parties incentivised to operate according to the protocol, a condition that enables the off-chain collaborative execution of the app. The main factor that forces participants to behave according to the rules is the fact that they have an amount of assets put in as stake on-chain, and that any misbehaviour will result in them losing their amount of assets.

The activity in a state channel can be described as a session of interactions between participating actors and a blockchain system (usually through a coordination smart contract). Participating actors mainly interact with each other, while they sporadically interact with the blockchain system to (a) initiate the state channel (b) terminate the state channel and (c) resolve any disputes that may come up. A state channel can be in one of the following four phases throughout its operation:

1) OPENING PHASE

At this phase participants have to go through a protocol to initiate the state channel. Usually, it involves the funding of the channel, during which participants commit assets on-chain that will function as a stake for the duration of the channel existence. Additionally, all participants sign an initial state for the channel, to which they all agree upon, to set the starting point of the state channel.

2) UPDATE PHASE

This phase is the one in which the core functionality of the state channel takes place. Participants typically communicate with each other (and not with the blockchain system) through cryptographically signed messages. Through such messages, they propose updates of the state channel and they also accept status updates proposed by others. The main concept is that a state update that has been signed by all participants is of equal finality with submitting this state on-chain. The channel remains in the update phase as participants behave according to the protocol.

3) DISPUTE PHASE

If users' behaviour deviates from the protocol then the state channel transitions to the dispute phase. Participants need to be in full consensus regarding the state channel's state updates. If this is not the case, because of a participant remaining silent, or proposing invalid state updates then a dispute arises and participants revert to the communication with the blockchain system to ensure a fair outcome. The dispute phase is the phase for which most differences amongst the various state channel implementations can be spotted. As the dispute is resolved the state channel may return to the update phase or progress to the closing phase.

4) CLOSING PHASE

This is the phase during which state channel finalises and locked assets are returned to the participants. In the optimistic case, closing occurs when participants have concluded their interaction and all agree to publish its outcome to the blockchain system, to finalise the app execution and release locked funds accordingly. In the pessimistic case, the state channel might be forced to transit to this phase through an unresolved dispute. In that case, to protect the assets of the honest parties, the state channel is closed and the locked assets are distributed to them according to the last agreed-upon state and/or in favour of the honest parties.

The main requirements for a functional state channel could be summarized in a few straightforward and fundamental concepts that have been carried over from traditional blockchain systems and are still the pillars of the whole scheme: trustlessness and finality. Trustlessness relates to the guarantee that a participant is not significantly endangering their stake regardless of the behaviour of other parties, and finality relates to the fact that a given channel state has the same validity as an on-chain state. Of course, state channels are also expected to serve their main purpose, which is to reduce the number of times that participants need to go on-chain and thus improve the load of usage a traditional blockchain system can serve.

A significant improvement on the state channels ecosystem was introduced with the development of state channel networks [33], [34], which enable the establishment of a virtual

state channel on top of multiple existing state channels, without requiring any on-chain transaction. Through state channel networks interaction with the blockchain system is further reduced, while a higher level of privacy is achieved since the on-chain publicly available footprint of state channels is even smaller. Moreover, further optimisation of state channels has taken place to handle multiple state updates in parallel and also support multiple virtual channels on top of a single existing channel without imposing any latency. Those are the main directions to which research in the domain has been targeted during recent years and the corresponding results are analysed in the present survey.

B. ABSTRACT DEFINITION OF STATE CHANNELS

In the present Subsection, an abstract definition of state channels is given and the main concepts that exist in most of the current state channel implementations are analysed. State Channels can only be relevant in the context of a Turing complete blockchain, therefore let's assume a blockchain platform that supports smart contracts' execution and a set of users (accounts) that want to interact with a deployed smart contract. For every interaction with the contract that alters its state, the users have to submit an on-chain transaction. This creates significant implications with respect to time delays, as the transaction rate of public blockchain systems is limited, and with respect to finality time, as even after the transaction is processed, the users have to wait for a couple of blocks to be mined to be sure that the transaction is not reverted. On top of that, the use of public blockchain systems requires the payment of fees for each transaction. The level of the fees is highly affected by the usage of the network and this is one of the main factors that limit public blockchain systems wide adoption by the users.

A large portion of deployed smart contracts are used by a limited number of users and in such cases, the aforementioned scheme is very inefficient. The imposed time delays and financial costs are disproportional to the volume of interactions, the security of which is ensured. An alternative approach proposed by the state channels ecosystem is to limit the main part of the interaction to a user to user communication (called state channel) and require users to go on-chain only when there is a substantive reason to do so. This mainly happens during setting up or closing a state channel or when one of the users behaves maliciously or goes offline.

Users interacting with a smart contract deployed on a traditional blockchain system, call its functions to update the state of the contract. The main reason behind state channels introduction is that the established workflow for transactions creates highly problematic time and cost overheads. In return, it ensures the secure update of the contract's status, but this is of interest only to the interacting users. State channels propose that the same level of security can be ensured by off-chain communication between interacting users in the optimistic case (when users are not malicious) and limited on-chain interaction in the pessimistic case (one or more users are malicious).

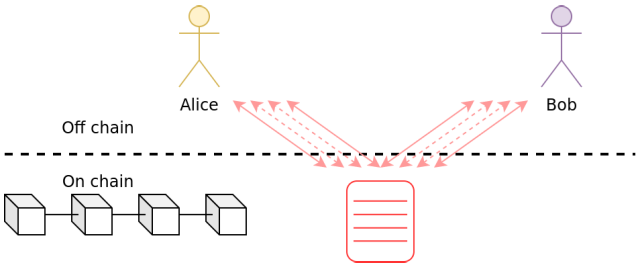


FIGURE 1. Traditional smart contracts workflow.

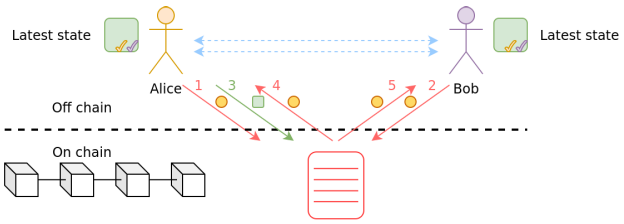


FIGURE 2. State channels workflow (optimistic case).

The main component that enables this interaction is a smart contract deployed on the blockchain that governs the operation of the state channel. While this is implemented in various formats, in each of the research efforts analysed in the next Section, its mission is standard and two-folded:

- It serves as an on-chain account to which participants deposit (or lock) funds. Those funds are eventually redistributed to participants according to the final state of the channel and ensure that users interact with the channel properly, because otherwise, they may lose their deposited funds.
- It implements functionality that validates transitions between two states of the channel. This is the basic mechanism that enables the secure operation of a state channel. It lives on-chain and it is employed by users that have been victims of the malicious behaviour of other users, to ensure a fair advancement of the state channel. On the other hand, the existence of the transition validation mechanism limits the applications that can be deployed to a state channel environment, as there is a strong requirement for an explicit description of all possible states of the application.

Figure 1 presents the workflow followed in the traditional context. Alice and Bob interact with a smart contract, which is deployed on-chain. Every such interaction that alters the state of the smart contract corresponds to an on-chain transaction that brings in all the time and cost issues discussed above.

Figure 2 presents the general workflow that is followed by most state channel protocols and specifically shows what happens in the optimistic case. Alice and Bob need to execute the same application as previously but this time they use a state channel instead of interacting with an on-chain contract. For the sake of simplicity, we assume that there are only two users that operate a single state channel and that they deploy on it a single app. Let's assume that the state channel app is

noted as *app* while its random state i (i stands for order) is noted as $state^i_{app}$. The smart contract that supports the state channel is noted as *SC*. The first step requires both participants Alice and Bob to fund the state channel by making an on-chain transaction of funds from their personal account to the *SC* contract (interactions 1,2). Those funds are locked in *SC* and will be redistributed back to the users according to the final $state^i_{app}$ of the app and under specific circumstances. The funding of the state channel brings the app to its initial state $state^0_{app}$.

Then comes the main operation of the app, during which the users may submit transactions to the state channel (off-chain communication between them) that update the state of the app. These transactions include a nonce i, the previous state, the updated state and an action/operation that triggers this transition and are signed by the users. The transaction is in the following format:

$$tr_i = (i, state^{i-1}_{app}, state^i_{app}, operation, sig_{Alice}, sig_{Bob}) \quad (1)$$

There are implementations that in two-party interactions only one of the parties, the one that makes the state transition signs it, but the general case is that all participants should sign the latest app state. At the end of the transition, each participant holds the signed transition tr_i. Based on that, they can prove on-chain that this is the valid current state, unless another participant holds a more recent signed and valid state transition $tr_j, j > i$. In the optimistic case, users keep on operating the state channel and running the *app* off-chain until its execution reaches an end. Then one of the users submits the final state transition to the *SC* (interaction 3). The *SC* allows a time period for challenges, that are mainly related to the existence of a later valid state transition and then releases the locked funds back to the users according to the final state (interactions 4,5).

On the contrary, there are cases where one of the participants does not behave according to the protocol because the valid advancement of the app's state is not aligned with their benefit or interests or because of communication failure (e.g. the user goes off-line). Such a user may:

- Become unresponsive with respect to proposing a state transition in their turn
- Become unresponsive with respect to signing a valid state transition another participant proposes
- Propose an invalid state transition

Most state channel implementations include a challenge-response scheme, under which users are protected from counterparts the behaviour of which deviates from the defined protocol. The result of such mechanisms is that the misbehaving user ends up losing all or part of the funds they have transferred to *SC* during the initial funding of the channel.

The pessimistic use case is depicted in Figure 3. Initially, both participants fund the channel (interactions 1,2) and then start exchanging app state updates. Let's assume that at some point in time, Bob becomes inactive and does not respond with a state transition in his turn (interaction 3). Alice can challenge Bob by submitting the last valid state transition tr_i

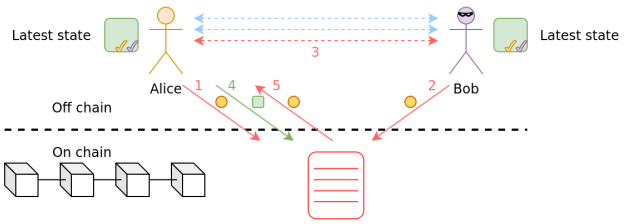


FIGURE 3. State channels workflow (pessimistic case).

to the *SC* (interaction 4).

$$tr_i = (i, state^{i-1}_{app}, state^i_{app}, operation, sig_{Alice}, sig_{Bob}) \quad (2)$$

This includes Bob's signature and thus proves that Bob has agreed upon it. Then a time period is allowed for Bob to respond by submitting the next state transition tr_{i+1} to the *SC*.

$$tr_{i+1} = (i+1, state^i_{app}, state^{i+1}_{app}, operation, sig_{Bob}) \quad (3)$$

If this happens, then the state transition is checked with regards to its validity and the execution of the app can be continued (either on-chain or off-chain according to the design of the implementation). If Bob does not respond in the predefined time, then the app execution ends and the state channel is closed, with the funds being redistributed in favour of Alice (interaction 5).

Let's also assume another scenario in which Bob proposes to Alice (requesting her to sign it) for an invalid state transition for the app:

$$tr_i = (i, state^{i-1}_{app}, state^i_{app}, operation, sig_{Bob}) \quad (4)$$

Alice can revert to the *SC* contract again and report the invalid transition Bob is proposing. According to the implementation, *SC* may directly close the channel or again provide a time period for Bob to come up with a valid state transition.

In any case, the main concept is that when a user holds a valid state transition, signed by all participants, and also knows that it is the latest state they have signed, then they are provided with the same finality guarantees they would get on-chain. There is an additional step to be taken, that is to go on-chain and challenge the counter-parting entity, but it is definite that there may be no other result than finalising the latest signed, valid state they have access to.

IV. STATE CHANNEL IMPLEMENTATIONS

In the present Section, significant research efforts in the state channels domain are analysed. Those are discussed in four distinct groups, formed with each publication's focus as a criterion. Early state channel implementations may have a payment channel design as their main contribution but are credited with the earliest significant state channel implementations. General state channel designs have the development of a widely applicable state channel framework as their focal point. General state channel networks implementations have extended the idea of state channels to networks and virtual

channels. Finally, there are application-specific implementations that while being centred around a specific domain and its needs, still present interesting findings.

A. EARLY STATE CHANNEL IMPLEMENTATIONS

In this Subsection, we present the research efforts that conceptualised the state channels approach and made the very first steps towards implementing such schemes. Those designs paved the way for the evolution of the idea of payment channels and moved from updating only the balances of participants to updating the state of smart contracts while staying off-chain. The efforts described herein have repeatedly been leaned on by later proposals that refined those very first efforts.

1) SPRITES AND STATE CHANNELS: PAYMENT NETWORKS THAT GO FASTER THAN LIGHTNING

Even though the main contribution of Sprites [35] is focused on a payment channels protocol, that protocol is designed in a modular way, based on a generic state channel abstraction. Sprite payment channels are modelled on state channels as they require an adaptable connection between on and off-chain processes. They incorporate a digital signature exchange scheme through which processes handle off-chain communications.

State channels are primarily used by Sprites protocol to implement its dispute handling process and to enable the adding and withdrawal of funds from an application contract without referring to the blockchain. They are straightforwardly described as a state machine that is constantly replicated between two parties and can progress through a transition function that is application-specific to implement. In the channel environment, a guarantee for liveness is provided, as all participating parties are guaranteed that the most recent state is identical for every other party and can be eventually finalised on-chain at any time.

An open channel will go forward in rounds, during which parties will submit inputs containing signed messages that include the current round number, the state to which the channel transitions, and the resulting blockchain output if applicable.

A dispute can be raised in the case that a party tries to submit an invalid response or becomes unresponsive, which will result in a state lacking one or more signatures. In either case, an honest party can trigger the dispute process that includes providing evidence that the previous round has already been validated and informing the rest of the participants of the dispute process's commencement. The dispute can then be cleared off-chain if evidence that contributes to the validation of the controversial round or evidence that support a later valid round is submitted. In case of insufficient evidence, resolving can happen on-chain by any party and will result in the channel exiting with the most recent committed valid state.

The layout described in the Sprites protocol with regards to the dispute process is commonly encountered in other efforts since the same or a similar approach has been also used on many later proposals on state channels.

2) PERUN: VIRTUAL PAYMENT CHANNELS OVER CRYPTOGRAPHIC CURRENCIES

The work of Dziembowski *et al.* [36] and others introduced an innovative modification on payment channels, as they were the first to present the concept of virtual channels, where intermediary parties enable payments between ''side'' parties they are already connected to without being involved in the payments. Perun virtual channels paved the way for other efforts with respect to virtual channels and their main contributions were the minimisation of interaction with the blockchain and the increased privacy for conducted payments. The same concept has been extended later on by some of the authors [34] to be applied to state channels as well. Perun protocol describes an implementation of payment channels that are built recursively over a custom, novel design of a state channel that is called multichannel. Multichannels were constructed to fit into the role of supporting virtual payment channels, but they can also be used beyond that aspect, for generic state channels.

A multichannel is identified by a unique identifier that can never be repeated. It can be created in one of two ways: (a) parties cooperatively agree on the channel parameters and the round execution begins, or (b) an initiator begins the process without consulting the other prospective participants. For every created multichannel a corresponding smart contract is deployed on the blockchain, and the multichannel remains active as long as the said contract is up and running.

A distinguishing feature of multichannels is a delicate conditional state update mechanism. This essentially means that the user is given the option to only accept some of the proposed updates, in contrast to a standard state channel in which such updates have to be treated as a group. This feature has been developed to support the functionality of creating and running in parallel many contract instances, called nanocontracts, while a mechanism that prevents parties from initiating many concurrent processes that will lead them to overspend has been put in place.

In the optimistic case, participants can update the contract state after locally executing it, without contacting the blockchain system. An on-chain transaction is required in the case of a nanocontract's registration, which happens for every contract separately, in the case of a dispute between parties, and during multichannel close. To ensure the safety of parties' funds, and also that participants will pay their due in every case, they must lock a stake in the nanocontract. In case a dispute occurs, nanocontracts are deactivated while it is ensured that no funds are blocked. Funds cannot be added to the contract after execution has begun, although authors state that such a functionality would be possible through a global double-spending mechanism for all registered nanocontracts.

A multichannel can be closed by any party, but only if all nanocontracts supported by it are inactive. Its state can be modified if all participants agree on the outcome.

Otherwise, in case of a dispute, the protocol leaves it up to the channel's contract to handle malicious or inactive behaviour. This is done through an application-specific mechanism that terminates the channel by distributing nanocontracts' funds to participants if a certain amount of time lapses with no response after the dispute has been submitted.

The authors went on to further improve their work through a more recent publication [37] that extends PERUN and also includes a security analysis.

B. GENERAL STATE CHANNEL DESIGNS

The majority of state channels focused research efforts aim to realise adaptable, generic designs that satisfy as many requirements as possible. There are cases in which research efforts focus more heavily on specific aspects or requirements. Collectively such implementations are the backbone of the state channels ecosystem and offer various outlooks on building protocols that enhance the efficient operation of blockchain systems.

1) FORCEMOVE: AN N-PARTY STATE CHANNEL PROTOCOL

Authors of the Force-Move protocol [38] identified that conflicting state submissions, external states dependence and inactivity are the three main factors that can lead to disputes between participants in a state channel. The Force-Move protocol was designed to combat the last one, namely the unresponsiveness of participants that threatens to lock all assets in a channel indefinitely. The authors have mainly focused on turn-based applications for which information required for dispute resolution is always fully embedded into the state of the application. In this way, they eliminated the external state dependency and conflicting states issues and focused only on providing an efficient solution for dealing with the unresponsiveness case. They primarily focus on applications that are or have similarities to, games, though any other application that complies with the Force-Move specifications, like payment channels, can be implemented through it.

The main component of the protocol is the Adjudicator: a most often, but not always, an on-chain smart contract that is integral to the functionality of this design. A generic interface to be implemented by the Adjudicator component of each different application is provided. This includes methods to interpret the different types of states as defined by the protocol, to facilitate off-chain communication and to validate states. A method called ValidTransition is arguably the most important method to implement, as it is the core of the protocol, giving the ability to separate a valid state transition request from an invalid one, and eventually produce the outcome of a dispute. The functionality of this method strongly depends on the content of the application and has to be implemented accordingly. Additionally, the Adjudicator is funded with the assets that participants are putting in as stake during their interaction. Those locked funds are the main mechanism that forces participants to act according to the rules of the application, as the Adjudicator releases and distributes those funds as indicated by the game's (app's) outcome.

A Force-Move channel's id has to be unique to prevent replay attacks, and it is the participants' responsibility to define it and double-check that the uniqueness requirement holds. State structure in Force-Move protocol is configurable according to the specific application that is to run, but there are some standard attributes, such as the number of the current turn and those that define how assets would be distributed to participants if the channel was to close at the specific state.

The key countermeasure that the authors applied against the inactivity of participants is the force-move method. It is triggered by one of the participants to force an unresponsive or non-complying opponent to act. It must be noted that in this protocol an invalid move is treated as a non-existent one. The force move process will result in either the smooth continuation of the game off-chain or its immediate termination and distribution of staked assets. This depends on the validity of the response of the challenge, or the absence of one. There are four valid ways for the challenged party to reply to the force-move request; respond with a move, prove a response to an alternative move, refute challenge, provide conclusion proof. If the challenged party responds with one of those four before the expiry of the dispute time window, the game progresses normally. Otherwise, it concludes based on the last accepted state.

There are five different phases for a Force-Move channel the PreFundSetup phase, the Collaborative phase, the Challenge phase, the Concluded phase and the Terminated phase. Force move challenges can be triggered only while the channel is in the Collaborative state, and triggers its transition to the Challenge state. The funding of the game happens externally, through an Adjudicator contract.

The authors themselves point out how their design is susceptible to various cases of malicious parties' behaviour and the causing of problems through firing invalid force move challenges, that still needing to be responded to.

2) COUNTERFACTUAL: GENERALIZED STATE CHANNELS

A "template" to make the adoption of state channels less of a burden for applications whose developers do not want to meddle with the blockchain is presented in this framework [39]. The design also allows the expansion of the template in an open-source style, providing an easy-to-use API so that added functionality can be developed for a channel and then used by any application that fits in a similar category. This concept has the added benefit of relieving developers of solely covering deployment costs and encourages sharing between different applications.

The counterfactual design stands upon four pillars. The Safety Criterion ensures the protection of participants' assets and trustlessness in the channel. The Finality Criterion sets the requirement for final state validity. Equally fundamental are the Responsiveness and Off-Chain Desideratums that stress the importance of encouraging participant behaviours that align with basic state channel functionality.

Similarly to most existing state channel implementations, the Counterfactual approach is dependant upon the liveness of the blockchain system it operates on top of, the availability of channel participants and the insurance that they have no incentives external to the channel, as well as the absence of errors in the application code.

Like an abundance of state channel designs, this one is also vulnerable to griefing attacks and stale update posters. The first can be halfway mitigated by practices that either sacrifice privacy or require a very large stake to be effective, while the second is the main reason most frameworks only address turn-based applications, since they depend on this concept to punish the posters.

The name of the protocol comes from the counterfactual concept that the approach is centred around. Authors take the idea of an event that, even though it has not happened yet, it can happen at any time and everyone can behave as if it already has, and apply it with slight variations to the three main components of a state channel construction:

- Counterfactual Outcomes: finalizable channel outcomes that have not yet been finalized but participants can act as if they have been.
- Counterfactual States: states that can be brought on-chain by any participant that is involved with them, but still only exist in the channel.
- Counterfactual contract Instantiations: a new process of instantiating a contract to a channel without involving the blockchain. It requires a registry that maps the counterfactual code to its Ethereum equivalent and also predetermines how its Ethereum address will occur.

Along with the aforementioned registry, the other functionality that has to be implemented on-chain is participants' deposits holding. Authors claim that such functionality can be adequately implemented by a multi-signature wallet as long as it is secure enough. On top of that, the authors state this approach brings in preferable characteristics, such as increased privacy and upgradability.

The proposal has been influenced by and made to fit on top of Ethereum's object-oriented design, resulting in high compatibility between the two layers. Every channel has a separate counterfactual state and functionality, and this approach facilitates the implementation of conditional payments and the enabling of instant on-chain fund deposits and withdrawals.

Channels that can be formed between users through a common intermediary are also implemented and are named Metachannels. Through this, the protocol supports the important functionality of virtual channels, which enables less interaction with the chain and has been consistently proposed in other research efforts since then. Also, popular third-party services like watchtowers [40], that aim at reducing the requirement for constant connectivity for participants, have been taken into account so that the framework is compatible with them.

3) CELER NETWORK: BRING INTERNET SCALE TO EVERY BLOCKCHAIN

The Celer Network [41] is an assortment of technologies aiming to improve scalability by any means possible. State channels along with sidechains are employed to that end, along with a value transfer routing method that increases throughput and a redesigned cryptoeconomic model.

The term network is used to describe the relationship between the various layers of the design, the base of which is the state channel and sidechain layer which is very relevant to the scope of this work. Celer uses state channels, called cChannels, to accomplish their goal of enabling fast off-chain interactions that can run on top of any blockchain system that supports smart contracts.

Because the design is destined to be blockchain-agnostic, authors identified common points between blockchain platforms that support the execution of Turing-complete programs, and incorporated such points in cChannels. One of those necessary components is an off-chain address translator that maps off-chain features to blockchain functions, through the use of a unique identifier dubbed as an off-chain address. Another key point, is the use of a Hash Time Lock Registry, which is implemented on-chain and is required for the implementation of atomic transactions that happen between multiple channels.

The concept of conditional updates and dependencies amongst states is greatly stressed in this work. The focus however seems to be on the more specific case of conditional payments. A detailed analysis is given for a General Payment Channel design as an example that could be of great use and fits the state channel specification. Based on common requirements that exist for state channels, authors have implemented several optimisations that are beneficial for their operation. There is an option for cooperative parties to sign every state as the final one and hence be able to close the channel in one transaction, and similarly, there exists a process that allows the opening of a channel in only one transaction as well.

Dependencies between states could bring great latency when claiming for a final state since it would be necessary to await the cooperation of every party that is involved with a depended-upon state. To avoid this overhead, a final state can be directly claimed by submitting a fraud-proof stake.

Celer also contains an alternative state channel model that is based on sidechains instead of the main chain of a blockchain platform. While it comes with added benefits concerning stake deposits and a reduced number of necessary on-chain transactions, it also has a set of drawbacks, namely the finality delay enforced by sidechains which will affect data availability and the absolute necessity of a fraud-proof bond to be deposited by the block proposer. Those disadvantages are recognised by the authors but no corresponding countermeasures are provided.

In general, no extensive security analysis has been done, and for the most part, the proposal seems focused on

a payment channel analysis, without providing sufficient details on the dispute process that is required for the secure operation of state channels.

4) TWO-PARTY STATE CHANNELS WITH ASSERTIONS

State Assertion Channels [42] were presented, in an effort to disburden honest parties from shouldering the costs of issuing a challenge in case of inactivity or of an invalid state submission. The design of State Assertion Channels proposes an alternate dispute settling process. A party issues a challenge when a counter-party is inactive. After that, the counter-party shall respond with a state assertion (state's hash) within a predefined time window. A response to that state assertion is a second state assertion issued by the first party. Either the first party responds with the next state assertion, hence stating that they accept the previous one, or they challenge it by submitting to the contract the full state in plaintext. If the assertion was indeed invalid, then the honest party gets the other's bond as a refund for the process.

Even though the requirement to stay connected to respond with assertions exists, the design is not compatible with outsourcing frameworks like Pisa [40]. Additionally, timers and countdowns are difficult to implement on blockchain applications without sacrificing security. The applications that can implement this state channel design are quite restricted since it only supports those that involve strictly two parties, interacting in a turn-based scenario that only allows single transition functions and never for an exception to be thrown.

5) YOU SANK MY BATTLESHIP! A CASE STUDY TO EVALUATE STATE CHANNELS AS A SCALING SOLUTION FOR CRYPTOCURRENCIES

Kitsune [43] is another state channel design by the authors off [42]. It supports channels of n parties regardless of the application they intend to run and combines features from previous implementations [35], [36] in an effort to integrate their features.

The authors provide an application contract template to facilitate the deployment of any app within a state channel. This allows the application contract to cease its operations on-chain, transitioning to a ''locked'' state, given the fact that participants' consent to move the app to a state channel. The state contract is instantiated and the list of participants and duration of the dispute period are defined. While the app is running off-chain, any participant can submit a state update that will only become valid if signed by all parties. Close can occur optimistically, if all parties sign a closing state, or through a dispute.

Once a dispute is initiated, which can be done by any party, a dispute timer starts to force participants to submit the latest valid state they hold along with state identification information and corresponding signatures. The channel contract will store the most recent valid state and after the time-out of the dispute timer, the channel is set to *OFF* state. The dispute's duration and its outcome are stored on-chain, while the execution of the process returns on-chain. The application contract can also be deactivated if the state channel remains inactive for long time periods.

The authors attempt to analyse the performance of their approach, along with any challenges faced and also estimate the actual cost of running a battleship application on it. They conclude that the proposed state channels scheme is an optimal solution only for applications without a strong dependency on liveness. Also in the case of participants that are not cooperative by default, the usage costs outweigh any of the benefits.

The framework allows non-turn-based submission of states but does not address how a simultaneous submission is handled. Additionally, it is common for a party to start a dispute once they do not receive all requested signatures on time (with respect to a local clock) and end up moving the application execution back on-chain. This approach seems like it will complicate and delay the progression of the channel. The dispute processes only possible outcome seems to be closing the channel, and return the application on-chain, even in cases where disputes could be resolved and the application execution could have been allowed to continue off-chain.

6) HYDRA: FAST ISOMORPHIC STATE CHANNELS

In this proposal [44] the concept of isomorphic state channels is introduced, to replace the sequential state processing that is predominant in state channel approaches. Hydra channels address applications that are run by multiple parties but require those parties to remain connected throughout the process, having liveness as an important condition. The protocol's main advantages are funds' security, high performance and preservation of the smart contract capabilities on the state channel.

Hydra makes use of the Extended UTXO model [45] to provide support for general state machines and therefore make Bitcoin Turing complete and able to support state channels. This enables the Hydra channels, called heads, to make use of the underlying blockchain's state representation without any translations or modifications being required. Thanks to the EUTXO use, heads can boast faster confirmation times, simultaneous transaction processing and full asynchrony in the optimistic case, along with smaller round complexity. The authors provide simulation results to assess the efficiency of their design and compare it to baseline approaches, to lay out the performance related characteristics.

Creating a head follows a commitment process similar to other state channels. Any party can take on the initiator role and announce the identities that are invited to be head members. Public key information is exchanged between parties through authenticated channels to be used for on-chain transaction authentication and off-chain multi-signature based validation of state updates. An initial transaction establishes the head, initialises the state machine and forges the participation tokens for every party, which will be integral to the state verification procedure.

Since the channel progresses and transactions are processed, it is common for parties to have conflicting views

of the head state. Snapshots are an effective conflict resolution mechanism that also improves on storage load since any transaction included in them can be discarded. Snapshot leaders are tasked with creating and getting snapshots signed by all participants, in order to create points that can resolve conflicting state cases.

Any head party can procure a certificate for the current UTXOs in the channel at any time and use it to initiate the head's closing. Members are given a contestation period to upload UTXO certificates and the most recent valid one gets finalised on-chain with the end of the contestation period.

Beyond the basic protocol, Hydra functionality has been extended to incorporate, among other features, the support of committing and decommitting UTXOs in the head without blockchain interaction and the ability to execute an optimistic close without a contestation period.

7) PISA: ARBITRATION OUTSOURCING FOR STATE CHANNELS

Pisa [40] builds upon the concepts first expressed in the Monitor [46] and Watchtowers proposals, with the same end goal, eliminating the requirement for state channel's participants to be constantly online and synchronised with the chain, in order not to be vulnerable to disputes with unfortunate timing and execution fork attacks by malicious parties. Pisa introduces the concept of bringing in a third party, named custodian, to prevent the above incidents from taking place while a participant, dubbed customer, is unable to do so themselves. The customer is also provided with evidence in the form of a receipt that can be used to punish the custodian in case they do not fulfill their obligations.

The implementation is supported on three contracts, that of the channel, that of the custodian, and one based on a simpler variation of the Sprites model [35]. Any channel participant wanting to hire a custodian to watch over their channel can submit payments in real-time to the custodian's smart contract, wired through a payment channel. In return, the customer receives a receipt that counts as evidence of the appointment and the accountability of the custodian. Custodians receive payment for every hash of state they are provided with, to be used to prevent an execution fork attempt. Note that the custodian gets a salted hash of the state and not the state itself to ensure channel members' privacy. Additionally, the nature of payment channels protect the customer from having their payment stolen, and the receipt protects them from a misbehaving custodian. However, this receipt has to be ratified before it is received, and the custodian will only do so after checking the validity of the conditional transfer sent by the customer, guaranteeing fair exchange on both sides. Protocol design protects the custodian from being framed by parties or any efforts by them to increase the storage load. It does not, however, offer an efficient way to protect a customer from a custodian colluding with the rest of the participants if the benefit offered is enough to make the custodian willing to waive their stake.

This solution only works if the participant knows and has planned to be offline for a while, not in the case of an unexpected crash unless they are proactively paying a custodian. Also, it does not secure against a challenge addressed to the customer while they are online, only against the attempt for an execution fork attack.

8) BRICK: ASYNCHRONOUS STATE CHANNELS

The authors of this work [47] have correctly identified the assumption of a synchronous network as a security weakness that most state channel implementations suffer from. Malicious entities are provided with valuable information, like the duration a network would need to be under attack for a successful disruption, and it is an obvious motive to attempt to censor honest parties during dispute periods so that they cannot respond to or receive challenges and hence timeouts can be manipulated to go off to the attacker's benefit.

The authors of Brick implement a system that needs to make no assumptions regarding message delivery to claim that it is secure. By redesigning the dispute handling process to involve a committee of third-party members, they have introduced intermediaries that enable the architecture to work on an asynchronous network and removed the necessity for parties to remain online.

In Brick, a valid state is defined as a state that has been signed by all members, is the most recent state (freshest) and has not been invalidated by the committee. A committed state has to have been signed by a sufficient number of committee members or be part of a block on the chain.

The phases of a Brick state channel's life cycle are divided into various sub-protocols. The initial phase is the Open phase, during which the channel is funded, its closing fee is determined, hashes of members' public keys are stored, and the collateral for committee members is deposited. The channel then moves on to the Update phase, during which the members create the announcement for the state that they agreed upon to follow. The sub-protocol that follows is the Consistent Broadcast phase when said announcement is sent by every party to the committee members in the form of a hash. Finally comes the Close phase, instantiated as the Optimistic Close, in which all members sign and publish the freshest state or the Pessimistic Close, for which a party requests committee signatures on the freshest state.

The authors of the paper execute an extensive analysis to prove they live up to the conditions they claim Brick meets. Security is achieved through the guarantee that a channel can only close in the freshest state, and liveness is assured since every valid operation will eventually be committed if not invalidated. To maintain privacy, there is no way for unauthorised external entities to gain information about the state of the channel before the initiation of a close protocol.

Incentives are provided to encourage honest behaviour assuming rational entities that aim to increase their profit. Committee members, apart from locking an initial collateral into the contract, also receive guaranteed fees for providing signatures and participating in the closing of channels.

Therefore, honest behaviour of rational committee members is guaranteed, and that also guarantees the security of the network.

Brick aims to get rid of the necessity for state channel participants to remain online for the entirety of the procedure. By extent, the risk of parties losing their funds or otherwise finding themselves susceptible to attacks that would interfere with their connectivity is also eliminated. The authors lay down the groundwork by re-imagining the dispute handling process so that a committee is involved in it, guaranteeing network activity without relying on time windows. Apart from the intended result, this has the additional benefit of eliminating the requirement to measure time in the blockchain system, a process that is known to be problematic. A possible negative point of this design is the financial rewards that have to go out to responsive committee members that weigh on the channel participants.

C. GENERAL STATE CHANNEL NETWORKS DESIGNS

Research efforts analysed in the previous Subsection have established the building blocks for developing state channel designs. To enhance the benefits of those designs, there have been subsequent attempts that aim at combining multiple state channels into networks. The concept of virtual channels has been integral to the state channel environment since it commenced the advancement beyond mere channel designs to channel networks that could further benefit the scalability prospects of blockchain systems. State channel networks have attracted various efforts to develop robust and secure protocols for off-chain set up of state channels between blockchain users on top of already established state channels.

1) GENERAL STATE CHANNEL NETWORKS

In this work [34] the authors set a course to provide a formal definition for a state channel network, along with security specifications, that had been missing from the environment until that point. They present a design for virtual channel constructions, that are built on top of existing channels, funded on-chain and noted as ledger channels. Through this approach, it is feasible to operate a state channel without the requirement to commit on-chain transactions during its opening and closing and also to potentially resolve disputes through intermediaries.

Virtual channels are recursively built on top of multi-contract ledger channels, which enables every ledger channel to support many virtual ones simultaneously and every party to involve themselves in the off-chain execution of more than one contract at a time. There is no limit on the number of intermediaries a channel can span across, while there is an assumption for a synchronous communication network.

The described implementation allows the building of virtual channels even between seemingly incompatible cryptocurrencies, under the single requirement that they support smart contracts. Virtual channels also increase privacy as they function on a peer-to-peer communication scheme. The existence of an intermediary improves security, adding an extra step before forcing participants to resort to the blockchain.

The intermediary is contacted and if they agree to host the virtual channel, they optimally need to be contacted only during open and close phases, while they are not required to interact with the blockchain. In the pessimistic case, when a dispute occurs it is handled differently than on a ledger channel. Since the intermediary, contrary to an on-chain contract, cannot be trusted, consensus has to be reached through the execution of contract instances on the underlying ledger channel of each participant with the intermediary. This enables the parties to settle on the last state signed by both. The intermediary is required to lock funds for every virtual channel they support for the channel to be functional, and those funds are protected by the contract instances shared with the parties that make use of the channel.

In terms of efficiency, the creation and optimistic execution of channels happen in a constant number of rounds that does not depend on the channel's length. The execution of the channel in the pessimistic scenario requires a number of rounds that are proportional to the length of the channel.

The design is limited in that it only accommodates 2-party state channels, and that execution delay may be increased in comparison to on-chain execution since the execution happens in rounds and not in real-time and that triggers communication delays. Additionally, the requirements placed on an intermediary, locking coins for the duration of contract execution and the responsibility to be available to mediate disputes, may discourage a party from undertaking that role. Authors suggest that this shall be balanced with a service fee, but this would have to be sufficient to motivate the party while at the same time still provide a cheaper alternative than running the application on-chain.

2) MULTI-PARTY VIRTUAL STATE CHANNELS

Two major additions in the state channel environment were introduced by the same authors [48] in a later stage. They expanded the 2-party virtual state channels design to support multi-party virtual state channels, while they also redesigned the dispute process, so that the time complexity of the channel, in the worst-case scenario, is independent of its length.

The implementation of multi-party virtual state channels happens by layering them on top of networks, formed out of 2-party ledger channels, that connect every participant to all other participants. That allows any party connected to the network to easily set up a state channel with any subset of the rest of the network participants. Those multi-party channels can execute off-chain contracts that concern more than two parties. To guarantee that the outcome of those contracts' execution will be reflected on the sub-channels, on which the multi-party channel is built upon, corresponding contract instances have to be instantiated on each one of the sub-channels. Those instances also protect the safety of the funds provided by the intermediary regardless of the behaviour of the rest of the participants.

The dispute board is the integral component of this redesigned dispute process. This process requires honest parties to rely on the ledger every time malicious behaviour is detected, in contrast to relying on the channel intermediaries. The dispute board functions as a state registration point on which all parties can register their latest valid view. In case of a dispute, the honest party detects and uploads a valid state to the board while it allows for other parties to respond in a given time frame, measured in rounds. Submitted states are then compared and the valid one is finalised. In case of a malicious party not responding or trying to enforce a false view, all other participants should register their own instances on the dispute board. This new approach can significantly decrease the worst-case time complexity, even if it creates a larger overhead for the underlying blockchain. Also, because the dispute process results are published, they can be securely used by other processes/contracts.

Virtual channels that support this model are dubbed ''hybrids'' since they adopt the same update and execution process as ledger channels but open and close as virtual ones. Channels that utilise direct and indirect disputes can cooperate seamlessly, in a balance that is according to the demands of each respective application.

The protocol makes some assumptions about the network, such as synchronous communication that guarantees the delivery of messages within a single round. This has as a result that adversaries, while able to see and reorder communications sent in the same round, cannot alter, delay, drop or add any. In terms of security, channel creation and update can only happen in total consent of participants and as long as a single honest party exists, they can always ensure the execution of a state. Regarding the framework's efficiency, the creation, as well as the optimistic update and close of a channel happen in a limited number of rounds, while in the pessimistic case those suffer a delay that is not dependant on the length of the channel. The authors claim that their design can guarantee high levels of fairness and efficiency in cases where a single malicious participant exists.

The proposed method is strongly coupled with the authors' previous work [34]. This is the reason why they recursively build long channels on top of two participants channels and do not opt for the creation of channels with greater length from scratch.

3) NITRO PROTOCOL

The Nitro protocol [33], extends an earlier scheme, Force-Move [38] and mainly relies upon three fundamental concepts; namely finalisation, redistribution and unbeatable strategies.

The first two are directly related to the process of channel termination, which should invoke the distribution of assets locked in the channel to its participants. Finalisation represents the storing of the state channel's outcome on-chain, while redistribution is an on-chain process (succession of operations) that enforces the redistribution of assets, that are locked in the channel according to this finalised outcome.

Participants operate on the basis of unbeatable strategies, which stand for the cases in which a participant can be sure that they can force the finalization of the channel to a specific state, given the data they hold, irrespective of the actions other participants may make. The series of actions the participant has to make is called an unbeatable strategy and is theoretically equal to a final event on-chain. Unbeatable strategies are defined with a slight difference between Turbo and Nitro, the two protocols analysed by the authors.

The simpler Turbo protocol allows for a single type of channel outcome and operation. Allocation outcomes essentially define the asset distribution between participants at every point in the channel. Turbo introduces manually set priority orders that govern the distribution of channel funds. The implementation of a channels' network is mainly based on Ledger channels, which are funded on-chain, and their sub-channels which operate completely off-chain and are funded by Ledger channels. All Ledger channels implement the Consensus Game, an operation that essentially enables participants to progress the state of the channel through finalisable outcomes they all agree on (called universally finalisable outcomes).

Nitro protocol builds on the foundations set by the Turbo protocol. It introduces guarantee channel outcomes that define the priority order for an allocation outcome of a channel. It is possible to have more than one guarantee addressing an allocation, and therefore obtaining an unbeatable strategy is a more intricate process, as there are not only different possible outcomes for every channel, but also different ways to distribute value for every outcome, thanks to the guarantees.

Proper virtual channels can be created by the use of the Nitro Protocol. Any number of participants can create a virtual channel as long as there is a common intermediary. That requires the creation of a single allocation channel, and a guarantee channel per participant targeting that allocation.

D. APPLICATION SPECIFIC IMPLEMENTATIONS

State channels research has been focused on developing generic protocols that could benefit the blockchain and network designs and be used for every possible application. However, this Section focuses on the special case of more targeted state channel implementations, that have been developed to support a specific application.

1) ÆTERNITY BLOCKCHAIN

Æternity is a blockchain platform that aims to support decentralised applications, while it integrates oracle services and scalability enhancements [49]. State channels have been integrated into Æternity, to make the architecture highly scalable. The only transactions that are recorded on-chain are those concerning finalised channels or settling disputes, and since channels function independently from each other, those few transactions can be processed by the blockchain in parallel.

An Æternity channel's lifecycle follows a pretty standard pattern.

To open, participants submit an on-chain opening transaction with the amount of funds they will commit to the channel. The starting participant essentially leaves the counterparty with a free option to co-sign and activate the channel at any point, which leaves their initial deposit in a vulnerable state. To mitigate this security issue, channels are funded through a series of interactive steps to protect the funds of honest parties.

During the operation of the channel, the two parties exchange signed channel states that are accompanied by a nonce that represents states order and enables the identification of the most recent state in case of a dispute.

Channel closing can happen by cooperatively signing a closing transaction, or through a dispute that allows a time period for the other party to challenge it with a newer state. Disputes are charged to the initiator, which initially unburdens the party that is usually at fault, but authors expect this to be balanced by the redistribution of funds that will occur if the dispute is valid.

Æternity blockchain platform operates only on the basis of state channels. Distribution of funds, as well as oracle message interpretation, are handled by Æternity smart contracts that provide the basis upon which state channels are built. Those contracts do not store state, but only securely output it after a transition takes place. Contracts are maintained locally by channels' participants and are deployed on-chain only when a dispute occurs. This provides enhanced privacy for every interaction.

Hashlocks are used in Æternity to enable functionality that resembles virtual channels, allowing parties to interact with each other even if they do not have an active channel established between them, as long as they are connected by a path of state channels. However, this is a bothersome process for the intermediaries that will have to lock an immense amount of funds, and will probably need to be compensated by a fee, making this a sub-optimal option, especially if parties wish to interact more than once.

2) FUNFAIR TECHNOLOGY ROADMAP AND DISCUSSION AND A REFERENCE IMPLEMENTATION OF STATE CHANNEL CONTRACTS

Funfair technologies attempted to enhance the casino gaming experience, through decreasing costs and latency by the use of state channels [50]. Their state channel design, named Fate channels operate much like most other state channels, but focus mainly on serving the needs of online gambling and thus focus on actions/events that could potentially happen during such a game.

The lifetime of a fate channel is equal to a single game session, during which parties, usually a client and a server that is the house, can communicate in rapid succession through a channel they have placed a stake in. The user is charged with the opening fees of the channel. Updates happen while participants sequentially sign new states. The client can terminate the channel whenever they wish through a cash-out option, while the house covers the closing fees.

Fate channels were described in more detail [51] in a later stage. Funfair reintroduced their bidirectional 2-party channels and went on to analyse the technical processes through which they operate.

Funding is based on ERC20 tokens and it takes advantage of their built-in multi-sig capability. Participants co-sign a single transaction that transfers tokens from both of their accounts to a third account, which funds the channel. Participant balances are stored on the channel, but the channels total funds are kept on-chain to prevent inconsistencies while multiple channels are simultaneously running.

To open a channel, parties have to deposit funds and sign the opening commitment. A timestamp is used on the commitment so it cannot be maliciously exploited at a later date. The application's code exists on the state machine so that it can be accessible and verifiable. The first, or initial state of the channel has to be signed by both parties before the opening of the channels because at least one reference point is necessary for a dispute or the channel's close.

An action is the only way to advance the state, and only a single party can be taking one at a time. Once an action is submitted, the state machine checks that the suggested state is valid and the occurring balances non-negative before approving it.

The protocol's fundamental rules require that participants react in a reasonable time after receiving a state update. Upon receiving a state update, they have to perform necessary validation checks, sign it and send it back. A dispute can be raised by a party when the counter-party strays from that setup. Disputes can be resolved by a valid response by the counter-party, if they notice it, or by a timeout. Dispute initiators with malicious intent, that may challenge with a stale state or try to change their move through a dispute, are punished.

At the channel's close, parties send a signed close action on-chain with all the necessary parameters so that the state can be deemed valid and finalisable.

The State machine maintains a contract for each game or game type.

3) GAME CHANNELS: STATE CHANNELS FOR THE GAMBLING INDUSTRY WITH BUILT-IN PRNG

Game Channels [52] proposal also targets the implementation of gambling and casino games through state channels, with authors targeting 2-party fraud-proof channels. The process for game channel use is quite straightforward and along with the basic principles of most state channel frameworks.

Typically the two parties are a player and a dealer since everything is discussed in the scope of casino games. The player initiates the process to create a channel, but both parties have to agree and sign an initial state, and that has to be verified by the on-chain contract for the process to move forward. The player subsequently submits the tokens to be held in escrow to the dealer, the contract confirms both parties are in full consent for the creation of the channel and realises it.

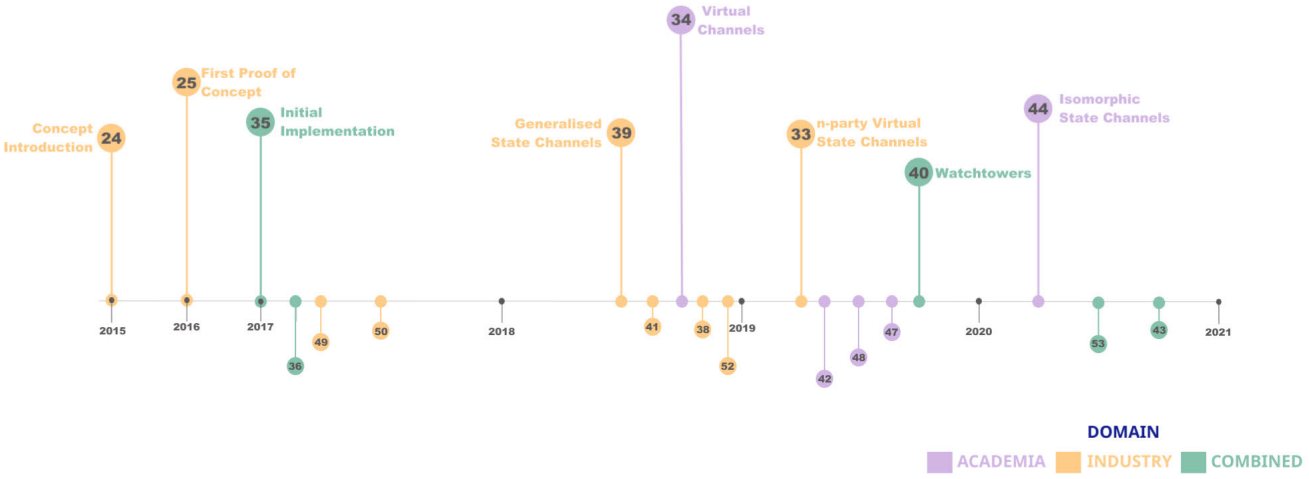


FIGURE 4. Timeline.

The state advancement process happens in rounds during which participants take actions regarding the game and communicate those to the dealer, who in turn provides an answer. As long as there is no cause for dispute, the channel keeps advancing to the next round.

Every time the state of a game channel is updated, a transaction with game-related information is sent to the blockchain, which partly hinders the supposed state channel functionality of reducing transaction load on the chain.

Either participant can place the call to close the channel and activate the protocol that corresponds to the situation. Specifically, the approach to closing differs whether it is a cooperative close, funds have run out, the channel has reached its expiration point or if a party is faced with a maliciously behaving (non-responding or submitting fraudulent data) party. In case of dispute, the validity of the data comprising the claim is checked by the smart contract.

Authors have modified their base protocol to expand its use cases and include channels between two players with no dealer involved, and also a setup where a third party can observe the game without participating in it.

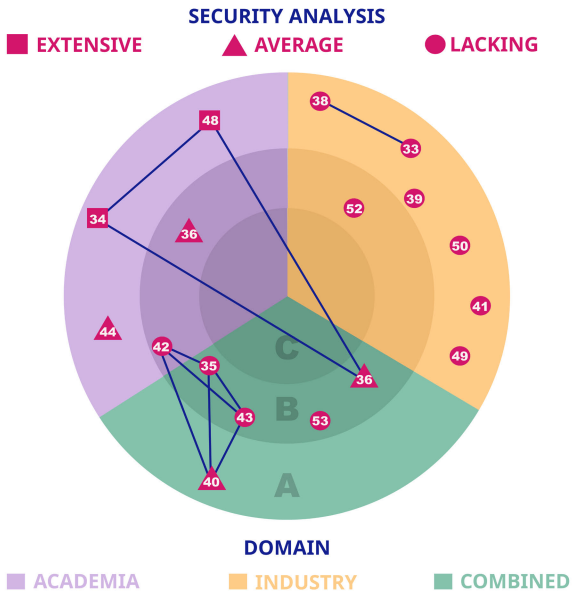


FIGURE 5. Radar depicting publication source, implementation quality, correlations and security analysis extent.

4) CRYPTOPOLY: USING ETHEREUM STATE CHANNELS FOR DECENTRALIZED GAME APPLICATIONS

Through the Cryptopoly implementation [53], the author aims to prove that developing a complex game like Monopoly on the blockchain, given that it is turn-based, is both feasible and cost-effective.

After choosing Ethereum as a platform, the necessity to address scalability issues led them to state channels as the optimal option with regards to latency and usage fees. The design adopts the dispute mechanism from Force Move [38], as well as the random number generation process used in Fate Channels [50].

A state machine with an approach that is as general as possible was designed since Monopoly is a highly customisable game. An Ethereum contract to handle the functionality of the channels (open, close, dispute) was also developed. Because validation processes differ per application, a generic applyChange method is used for its adaptability compared to hard coding the parameters. The specific effort has limited contribution as it only aims at applying already developed approaches to a specific application.

V. ANALYSIS OF IMPLEMENTATIONS

A. GENERAL ANALYSIS

Through the extensive study of the existing research efforts in the state channels' domain, various aspects to be commented on have emerged. Figures 4 and 5 depict observations with regards to the general landscape as this has been formed by all papers presented in Section IV. In both Figures, a division of the efforts according to their field of origin can

be discerned. The blockchain ecosystem is a diverse space and this classification provides an insight regarding which sector (academia/industry) motivated conducted research and also what is the level of interaction between the two sectors. Additionally, this distinction also provides data as to which designs are functional purely within an academic environment and which expand beyond that and into available and usable products. The categories are the following:

- **Academia:** includes papers that have been published by academic researchers, affiliated with a university/research institution.
- **Industry:** includes any research done by commercially oriented groups, e.g. startup companies, usually concerning a product.
- **Combined:** includes any work published either by authors affiliated with both aforementioned fields, or a group of authors that have a different field of origin.

It should be noted that only the focus of the present survey is state channels. Therefore, for schemes that partially included state channels [35], [36], [41], [49], any evaluation attempted only concerns the relevant parts of the overall designs.

In Figure 4 the categorization of the evaluated papers according to the divisions is especially useful since it enables observations regarding which sector was more active per time frame.

To visually illustrate the progression and evolution of the state channels ecosystem since the concept's introduction in 2015, Figure 4 depicts a timeline that shows all the major milestones and every state-channel related publication. It can be observed that for the first period from 2015 through the beginnings of 2017, the only noteworthy advancement is only a state channel proof of concept implementation [25]. In the first quarter of 2017 the first specification for a state channel design as a part of a larger payment channel scheme is analysed in Sprites [35], while Perun [36] follows the same year on a similar note. Counterfactual [39] provided a very detailed design, that was the first to be exclusively state channel related, in early 2018. In the second half of 2018, state channel networks and the concept of virtual channels is introduced by Generalised State Channel Networks [34]. At the start of 2019 comes the noteworthy extension of the concept to n-party channels, with Nitro [33] being the first protocol to build on the idea. In the second half of 2019, Pisa [40] focuses on resolving the common security assumption according to which all state channels participants are required to be constantly online, extending the concept of Watchtowers that had been introduced earlier. Another work focused on the same problem but with a different approach that comes around the same time is the design proposed by Brick [47]. The most recent noteworthy contribution regarding state channels is the Hydra [44] proposal that presents isomorphic channels, a very interesting enhancement.

Figure 5 also utilises the distinction of proposals according to their sector of origin introduced in Figure 4. This allows the comparative evaluation between this factor and every other aspect of the radar.

All designs analysed in this survey have been evaluated for their technical maturity. In Figure 5, efforts are depicted on a radar with those with a more well-rounded and advanced implementation in the outer ring. This allows the evaluation of the average quality of implementation per sector of origin. It can be seen that the vast majority of papers stemming from the industry and startup sector average a high level of technical maturity, while academic papers are split between the two most mature rings of the graph. Proposals that occurred from the combined efforts of academia and the industry sector tend to fall on the second ring. It is a fortunate observation that the works analysed herein score a high average on the quality and extent of their practical implementations, with all providing a more or less extensive technical analysis in their appendices or through a corresponding code repository.

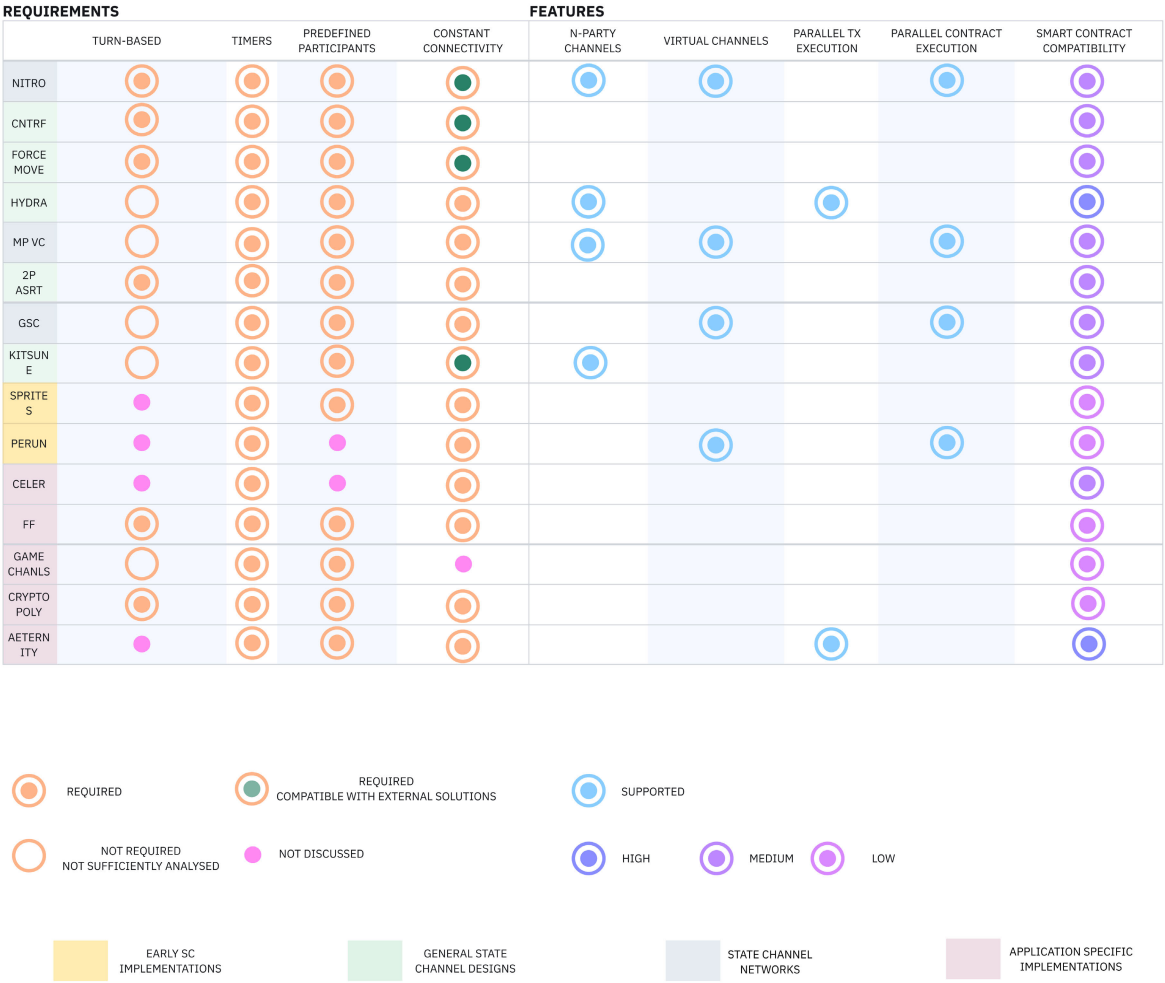
Another dimension in Figure 5 is the extent of the security analysis present in each research proposal. According to the thoroughness and detail of the analysis, three distinct ranks have emerged.

- **Lacking:** The majority of designs seem to find it sufficient to mention the security properties state channels inherit from the underlying blockchain system, or to claim security guarantees fundamental to the state channel, such as liveness and fund safety. They do not go as far as to analyse how those concepts are achieved, how the blockchain system's security extends to the protocol, and how security vulnerabilities occur because of the properties of the specific proposal. For example, more often than not, the way the dispute process is designed in ways that leave honest parties vulnerable to griefing attacks, or the channel's security becomes obsolete in the case of a software error in the contract. Methods to secure against those phenomenons are not provided by designs that fall in this category.
- **Average:** Designs that have been deemed to adequately cover the security analysis aspect by providing proofs for the security claims they make, have been placed in this category. Authors usually do that according to a defined threat model that seems to cover possible cases to a satisfactory degree.
- **Extensive:** There exist a couple of designs that provide very rigorous security analysis and use established protocols like the Universal Composability framework [54] or similar variants to evaluate the security of their design to the utmost extent.

It can be noticed that papers originating from the industry domain seem to be lacking a formal security analysis, while publications from academia tend to score a higher average on this front. Papers from the third domain appear to be falling in between.

Additional information that can be drawn from Figure 5 is the correlations between published works. The edges coloured in dark blue join papers that share at least one author. This enables evaluation of the diversity in the state

TABLE 1. State channel implementations comparison.



channel research field as well as observation of interconnections between works that seem to have a different sector of origin. It has to be noted, that there is significant interaction between academia and industry on the state channels domain.

While it may seem like an obvious categorisation in terms of supported blockchain platforms has been omitted, the vast majority of designs address Ethereum or Ethereum-like platforms that support smart contracts. The only exceptions can be found in Hydra [44] that means to be implemented on Bitcoin running EUTXO protocol and Aeternity [49] that is in itself a blockchain platform.

B. COMPARATIVE ANALYSIS

Through the analysis of the state of the art, common points between research efforts have been identified. Through this process, the main restrictions imposed by authors for their protocols to be functional along with the main features those offer have been summarized and are presented in Table 1. In the Table research efforts are categorized according to the Subsections presented in Section IV:

- **Early State Channel Implementations** papers were the first to implement a state channel infrastructure, even if that was not their main contribution. It is therefore expected that some characteristics were considered out of their scope.
- **General State Channel Designs** includes state channel designs of a generic nature that focus on being as broadly applicable as possible.
- **General State Channel Networks** proposals also have a wide area of applicability but focus on introducing the concept of virtual channels and the interactions within a state channel network.
- **Application Specific Implementations** covers any state channel related design that was developed with a specific use in mind, and therefore some features are often passed by in favour of others that better serve the implementations' goals.

Many proposals impose restrictions on the applications their state channel design supports. Those restrictions may be limiting but at the same time, they enable designs that are robust and can securely support a specific subset of apps.

On the other hand, it is not uncommon for proposals to omit to place such restrictions and then fail to adequately analyze how the issues that may occur from the allowed scenarios are handled. The first section of Table 1, entitled Requirements, depicts said restrictions that are present or absent in each of the works analysed. The Table indicates which are the specific conditions each research effort requires to hold. When this is not the case the Table indicates whether or not each scenario has been explored to a satisfactory degree.

The second section of Table 1, entitled Features, presents positive characteristics supported by each design. It is desirable for each effort to have as many checks as possible on this end of the Table, as that corresponds to offering a wide variety of features and covering an abundance of scenarios. All in all, an ideal design would score zero marks in the Requirements section and all marks in the Features one. Of course, no such proposal exists, but a detailed comparison between existing proposals can be conducted based on Table 1.

1) REQUIREMENTS

In this Subsection, the different requirements set by state channel implementations are discussed.

- **Turn-based Applications:** To facilitate the handling of the dispute process and the assignment of blame, most designs limit their applicability to strictly turn-based applications. This has the added benefit of not getting caught up in trying to handle simultaneous state update proposals and facilitating the process of punishing stale update posters. There exist a number of applications that do not impose or do not address this parameter. However, most of those designs do not explain how to efficiently and securely handle applications that do not adhere to the turn-based paradigm.
- **Timers:** In the great majority of cases, dispute handling is based on challenge-response schemes and thus requires a time measuring operation, to manage a time out period. This is necessary to allow an interval during which the dispute can be contested, while also ensuring the channel's progression will not stall indefinitely. However, there still is no provably secure way to measure time on the blockchain. Using block time is sub-optimal as it is subject to change and also bounds the granularity of time measurement.
- **Constant Connectivity:** This assumption is derived from the conditions related to dispute resolution. Since there tends to be a time limit on the ability to respond to/contest a dispute, a party that gets offline for any reason at an unfortunate moment can face severe consequences. Because an online connection can be unpredictable, having such significant stakes depending on it is a major security issue. Pisa [40] and Brick [47] recognise the importance of this matter and focus their designs on the effort of mitigating it. Because those two efforts are rather specific and aim to diminish only this specific requirement, they cannot be directly compared to other solutions and they have not been included in the comparison of Table 1.
- **Predefined Participant Sets:** A requirement that has been present in every one of the works discussed in this survey is that the set of participants forming a state channel is predefined and remains unchanged through the channel's entire lifespan. The reason this is such a widespread assumption is that changes to the participant set would complicate the funding process and require communication with the blockchain for the addition or removal of a party. While this assumption simplifies the development of the protocols, it substantially limits the applications that can be served by such designs.

2) FEATURES

- **Multi-party Channels:** The initial payment and state channel designs could only accommodate interactions between two predefined participants. The same is true for the initial virtual channel designs that could only be formed between two parties. Proposals further down the road enabled communication via state channels, virtual or not, amongst an unlimited number of participants, given that those were predefined. Such a feature allows a wider set of applications to be deployed on top of state channels.
- **Virtual Channels:** Virtual channels enable the creation, progression and closing of the channel without interaction with the underlying blockchain system. Depending on the design, disputes can even possibly be resolved without any on-chain interaction. Virtual channels are practically synonymous with the concept of a state channel network as they rely on the existence of underlying channels that have been directly funded on-chain (ledger channels). A common intermediary that maintains such a channel with both participants can enable the creation of a virtual channel between them, while even n-party virtual channels can be built through such ''paths''.
- **Smart Contract Compatibility:** An important point to consider when deploying an application to a state channel is related to developing a smart contract to support the application. The installation of the application has to cope with the requirements set by the state channels design used in terms of the contract that needs to be running on-chain. The least the development effort required to port an application from on-chain to a state channel the better. Low compatibility refers to the requirement for a contract to be written from scratch to be state channels compatible. Medium compatibility corresponds to the majority of the cases, where a contract is expected to go through some alterations (make use of a library, implement a specific function) to be state channel ready. High compatibility is only achieved in a handful of cases that allow the deployment of an existing contract to a state channel with no development requirements.
- **Parallel Transaction Processing:** Transactions on state channels are often dependent on each other, especially

since the turn-based scenario is heavily favoured. However, if a design minimizes the dependencies between transactions, then those can be concurrently processed and thus latency is reduced. While this is a desirable feature it requires specific conditions to hold with respect to the underlying blockchain protocol.

- **Parallel Contract Processing:** Virtual channels are built on top of ledger channels. Ledger channels need to have a contract instantiated in them for each virtual channel they support. Therefore, a channel with the ability to process more than one contract at a time also enables the support of more than a single virtual channel at a time. This feature enables more efficient use of deployed ledger channels as it provides a greater magnitude of application capacity for a given set of established channels.

3) EFFICIENCY

Since the main goal of state channel designs is to reduce the transaction load and keep the, often hefty, fees associated with on-chain transactions to a minimum, it would be optimal to evaluate every proposal according to the degree of success regarding the aforementioned goals. However, an assessment for fees is impossible, since fees have an absolute dependency on factors beyond the state channel implementation (blockchain application, blockchain system, state of contract etc.) and cannot be consistently calculated for all approaches.

Table 2 represents the closest to a fair efficiency evaluation between the state-channel proposals by calculating the number of on-chain transactions necessary in each major phase of the state channel as described in Section III. Those phases are Opening, Update, Dispute, and Closing and in most of the cases the number of transactions is expressed in relevance to the number of channel participants denoted on the table as p.

a: OPENING

The processes that are included in this phase are typically identical for every state channel: (a) the deployment of the channel contract (b) the funding of the channel. Variations in how the funding takes place and the occasional, design specific, additional actions cause diversity in the number of necessary on-chain transactions. The most significant differences can be seen between designs that refer to ledger channels, that are directly funded and those that refer to virtual channels. The latter, since funded by pre-constructed ledger channels, require no on-chain operations at this stage.

b: UPDATE

It is generally expected, for a state channel to be beneficial, that during optimistic state progression there will be no contact with the blockchain. However, it is made obvious through this analysis that this is not an absolute rule and that there is a single implementation that requires an on-chain transaction for each state transition.

c: DISPUTE

Even though this is the phase where designs diverge the most from one another, it is most often in the form of the dispute transaction submitted rather than the number of transactions necessary. To better analyse the Dispute phase we have split it into two sub-phases on Table 2, Dispute Initiation and Dispute Resolution.

d: CLOSING

Similarly to the opening phase, closing also has a universally adopted form. It signals the distribution of assets after either a dispute (resolved or unresolved) or a cooperative update to a finalisable state. The diversity once again is mostly observed between the ledger and virtual channels, with the second category being able to close entirely off-chain.

Many of the designs allow users to withdraw and deposit funds even after the channel has begun operating. Those scenarios have not been taken into account since they always involve an on-chain transaction.

A noteworthy effort to eliminate the correlation between fees and the several factors it relies on is made by the authors of [42]. This is achieved through introducing a static form for the dispute submission irrespective of the nature of the application and eliminating the need for the valid state to be computed by the contract. However, this still only applies to the optimistic case of a dispute where the initial request is obviously valid and not challenged further.

It must be pointed out that the information we were able to extract for setting up Table 2 was very disproportionate from one work to another. Some designs gave enough detail to depict both a best and worst-case scenario, while others presented only a general idea. The values on the table marked with an asterisk (*) are our best interpretation from the limited descriptions and might not be entirely accurate. In a handful of situations, the data was insufficient or absent and those cases have been marked with a dash (-).

VI. DISCUSSION

The relatively short history of state channels makes it feasible to comment on the evolution of the domain from its initiation up to today. It is an interesting observation that the introduction and initial efforts in the field were pioneered by the industry sector, as discussed in Section V. The industry has also been very active overall, claiming the majority of state channel related publications. This points to the fact that the need for this scalability-enhancing technology was quite apparent in real-world blockchain deployments. The first major state channel implementations were products of joint efforts between industry and academia, and this collaboration went on to contribute a number of publications later on. The introduction of virtual channels marked the first proposal of fully academic origin, with academic researchers being increasingly active ever since, as state channel designs have evolved and progressed.

TABLE 2. Comparison of state channel designs regarding on-chain transaction load.

	OPENING	UPDATE	DISPUTE INITIATION	DISPUTE RESOLUTION	CLOSING
NITRO	0	0	-	-	0
CNTRF	p+1	0	1/2*	-	-
FORCE MOVE	p+1	0	1	1	1
HYDRA	1	0	-	-	1
MP VC	0	0	1	p-1	0
2P ASRT	p+1	0	2	1	1*
GSC	0	0	0/1	0/p-1	0
KITSUNE	-	0	1	p-1	1/p*
SPRITES	p	0	1	1/2	1/2
PERUN	p+2*	0	1	-	2
CELER	1	0	1*	-	1*
FF	4+p*	0	1	1	p*
GAME CHANLS	1	1	1	p-1	1
CRYPTO POLY	p+1	0	1	p-1	1
AETERNITY	0*	0	1	-	0/1
PISA	p+1*	0	1	p	3
BRICK	p	0	1	p	1

EARLY SC IMPLEMENTATIONS
GENERAL STATE CHANNEL DESIGNS
STATE CHANNEL NETWORKS
APPLICATION SPECIFIC IMPLEMENTATIONS

(-) = ABSENCE OF DATA
(*) = LIMITED DATA
p = NUMBER OF CHANNEL PARTICIPANTS
BEST CASE/WORST CASE

At the point that has been reached, it can be said that the state channel approach is quite effective when applied to applications that adhere to specific rules. However, as it stands currently, the application scope is rather narrow. It is easily discernible from Table 1 that there is a high number of requirements maintained amongst proposals, for a state channel design to be effective and beneficial.

Such specifications vastly limit the applicability of state channel technology resulting in a downplay of its effectiveness on the overall improvement of blockchain scalability. It is a dire priority that future research focuses on applications that can eliminate at least some of the most restrictive requirements, so that state channels can be used broadly enough to have their advantages fully exploited. Such requirements that it is critical to be rescinded are the predefined participants' set and the turn-based interaction. Constant connectivity seems to be partially resolved, while time management is a general issue in blockchain systems.

In terms of features, beyond those outlined in Section III, some others have also emerged, as the applications using state channels have increased. A design must prioritise keeping the storage requirements to a minimum by preventing the state from endlessly expanding. It is also positively viewed if a design is blockchain agnostic and can be painlessly used between different blockchain platforms. Going beyond necessities, there is a number of desirable characteristics for a proposal to have and those are outlined clearly in Table 1. What is currently missing from this research field is a proposal with a good balance, if not a perfect score, between the absence of requirements and the presence of features. The development of such a design is what future research should

focus on if state channel technology is to advance beyond its current point.

As a general outline of the state channels domain, it has to be noted that research outcomes are significant but not at the level the blockchain community requires. While the concept is sound and early implementations have paved the way for creating a robust layer 2 solution, it seems that efforts have slowed down from early 2020 and on-wards. Apart from that, it is evident that the number of publications is lower than the expected one, given the importance of the scalability problem of public blockchain systems. On top of that, there is also significant redundancy among proposed protocols. The main reason behind this limited research impact is mainly due to the existence of alternative layer 2 schemes that aim at providing similar solutions to the scalability problem. During the early state channels years, the side-chains approach was very popular, while recently optimistic and zero-knowledge roll-ups have emerged as better layer 2 solutions.

To our view, the scalability problem in public blockchain systems sources from two factors (a) actual low transactions processing capacity and (b) misuse of given capacity. While all other layer 2 proposals aim at improving the transactions processing capacity, state channels aim at minimizing the on-chain interaction. State channels attempt to minimize the utilization of the limited on-chain resources. While a blockchain system is the source of ground truth, it is inefficient to make use of it continuously. The state channels approach proposes to make use of it only when this is highly required (e.g. a user acts maliciously), given that it is feasible to operate off-chain in all other cases. In other words, state channels is the only protocol that aims at enhancing the way apps utilise the given capacity of a blockchain system. Because of that, state channels will remain relevant even if another layer 2 solution achieves to significantly multiply processing capacity. A combination of state channels with any other layer 2 solution, that increases processing capacity, is feasible and makes sense given that state channels will enable the best possible use of the increased capacity.

REFERENCES

[1] H. Treiblmaier and T. Clohessy, *Blockchain and Distributed Ledger Technology Use Cases*. Cham, Switzerland: Springer, 2020.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[3] V. Buterin, "A next-generation smart contract and decentralized application platform," Ethereum Found., Zug, Switzerland, White Paper, 2014, vol. 3, no. 37.

[4] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A survey of distributed consensus protocols for blockchain networks," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1432–1465, 2nd Quart., 2020.

[5] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 1159, Jan. 2016.

[6] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse, "Bitcoin-NG: A scalable blockchain protocol," in *Proc. 13th USENIX Symp. Netw. Syst. Design Implement. (NSDI*, 2016, pp. 45–59.

[7] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," 2017, *arXiv:1708.03778*.

[8] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 583–598.

[9] J. Stark. (2018). *Making Sense of Ethereum's Layer 2 Scaling Solutions: State Channels, Plasma, and Truebit*. [Online]. Available: https://medium.com/l4-media/making-sense-of-ethereums-layer-2-scalingsolutions-state-channels-plasma-and-truebit-22cb40dcc2f4

[10] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. (2014). *Enabling Blockchain Innovations With Pegged Sidechains*. [Online]. Available: http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains

[11] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K.-R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *J. Netw. Comput. Appl.*, vol. 149, Jan. 2020, Art. no. 102471.

[12] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," Ethereum Found., Zug, Switzerland, White Paper, 2017, pp. 1–47.

[13] (2021). *Layer 2 Rollups*. Accessed: May 16, 2021. [Online]. Available: https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/

[14] I. Allison, "Ethereum's Vitalik Buterin explains how state channels address privacy and scalability," Ethereum Found., Zug, Switzerland, White Paper, 2016.

[15] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16440–16455, 2020.

[16] P. R. Rizun. (2016). *The Excessive-Block Gate: How a Bitcoin Unlimited Node Deals With 'Large' Blocks*. Accessed: May 16, 2021. [Online]. Available: https://medium.com/@peter_r/the-excessive-block-gate-how-a-bitcoin-unlimited-node-deals-with-large-blocks-22a4a5c322d4#.riqy7lm36

[17] A. Singh, R. M. Parizi, M. Han, A. Dehghantanha, H. Karimipour, and K.-K. R. Choo, "Public blockchains scalability: An examination of sharding and segregated witness," in *Blockchain Cybersecurity, Trust and Privacy*. Cham, Switzerland: Springer, 2020, pp. 203–232.

[18] S. Bouraga, "A taxonomy of blockchain consensus protocols: A survey and classification framework," *Expert Syst. Appl.*, vol. 168, Apr. 2021, Art. no. 114384.

[19] S. M. H. Bamakan, A. Motavali, and A. B. Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Syst. Appl.*, vol. 154, Sep. 2020, Art. no. 113385.

[20] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proc. Int. Conf. Manage. Data*, Jun. 2019, pp. 123–140.

[21] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "SoK: Sharding on blockchain," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, Oct. 2019, pp. 41–61.

[22] S. S. Chow, Z. Lai, C. Liu, E. Lo, and Y. Zhao, "Sharding blockchain," in *Proc. IEEE Int. Conf. Internet Things (iThings), IEEE Green Comput. Commun. (GreenCom), IEEE Cyber, Phys. Social Comput. (CPSCom), IEEE Smart Data (SmartData)*, Jul./Aug. 2018, p. 1665.

[23] (2021). *Layer 2 Scaling*. Accessed: Apr. 29, 2021. [Online]. Available: https://ethereum.org/en/developers/docs/layer-2-scaling/

[24] J. Coleman. (2015). *State Channels*. Accessed: Mar. 29, 2021. [Online]. Available: https://www.jeffcoleman.ca/state-channels/

[25] P. Grau. (2016). *Lessons Learned From Making a Chess Game for Ethereum*. Accessed: Mar. 29, 2021. [Online]. Available: https://medium.com/@graycoding/lessons-learned-from-making-a-chess-game-for-ethereum-6917c01178b6

[26] *Payment Channels*. Accessed: Mar. 29, 2021. [Online]. Available: https://en.bitcoin.it/wiki/Payment_channels

[27] J. Poon and T. Dryja, "The Bitcoin lightning network: Scalable off-chain instant payments," Lightning Netw., San Francisco, CA, USA, White Paper, 2016.

[28] S. Lee and H. Kim, "On the robustness of lightning network in Bitcoin," *Pervasive Mobile Comput.*, vol. 61, Jan. 2020, Art. no. 101108.

[29] BrainBot. (2021). *Raiden Network*. Accessed: Mar. 29, 2021. [Online]. Available: https://raiden.network/

[30] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Off the chain transactions," IACR Cryptol. ePrint Arch., Tech. Rep. 2019/360, 2019.

[31] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Layer-two blockchain protocols," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2020, pp. 201–226.

[32] M. Jourenko, K. Kurazumi, M. Larangeira, and K. Tanaka, "SoK: A taxonomy for layer-2 scalability related protocols for cryptocurrencies," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 352, Apr. 2019.

[33] T. Close, "Nitro protocol," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 219, Feb. 2019.

[34] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 949–966.
[35] A. Miller, I. Bentov, R. Kumaresan, C. Cordi, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," 2017, *arXiv:1702.05812*.
[36] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "PERUN: Virtual payment channels over cryptographic currencies," IACR Cryptol. ePrint Arch., Tech. Rep. 2017/635, 2017.
[37] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 106–123.
[38] T. Close and A. Stewart, "ForceMove: An n-party state channel protocol," Magmo, White Paper, 2018.
[39] J. Coleman, L. Horne, and L. Xuanji, "Counterfactual: Generalized state channels," L4 Ventures, Toronto, ON, Canada, White Paper, 2018. Accessed: Nov. 4, 2019.
[40] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, Oct. 2019, pp. 16–30.
[41] M. Dong, Q. Liang, X. Li, and J. Liu, "Celer network: Bring internet scale to every blockchain," 2018, *arXiv:1810.00037*.
[42] C. Buckland and P. McCorry, "Two-party state channels with assertions," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2019, pp. 3–11.
[43] P. McCorry, C. Buckland, S. Bakshi, K. Wüst, and A. Miller, "You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2019, pp. 35–49.
[44] M. M. Chakravarty, S. Coretti, M. Fitzi, P. Gazi, P. Kant, A. Kiayias, and A. Russell, "Hydra: Fast isomorphic state channels," IACR Cryptol. ePrint Arch., Tech. Rep. 2020/299, 2020.
[45] M. M. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, M. P. Jones, and P. Wadler, "The extended UTXO model," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2020, pp. 525–539.
[46] T. Dryja and S. B. Milano, "Unlinkable outsourced channel monitoring," Lightning Netw., San Francisco, CA, USA, White Paper, 2016.
[47] G. Avarikioti, E. K. Kogias, R. Wattenhofer, and D. Zindros, "Brick: Asynchronous payment channels," 2019, *arXiv:1905.11360*.
[48] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2019, pp. 625–656.
[49] Z. Hess, Y. Malahov, and J. Pettersson. (2017). *Æternity Blockchain*. [Online]. Available: https://aeternity.com/aeternity-blockchainwhitepaper.pdf
[50] J. Longley and O. Hopton, "Funfair technology roadmap and discussion," Funfair Technol., New York, NY, USA, White Paper, 2017, p. 6.
[51] J. Longley. (2019). *A Reference Implementation of State Channel Contracts*. Accessed: Mar. 29, 2021. [Online]. Available: https://funfair.io/a-reference-implementation-of-state-channel-contracts/
[52] A. Chernyaeva, I. Shirobokov, and A. Davydov, "Game channels: State channels for the gambling industry with built-in PRNG," IACR Cryptol. ePrint Arch., Tech. Rep. 2019/362, 2019.
[53] M. Xu, "Cryptopoly: Using Ethereum state channels for decentralized game applications," Barrett, Honors College, Arizona State Univ., Phoenix, AZ, USA, White Paper, 2020.
[54] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 136–145.

LYDIA D. NEGKA is currently pursuing the bachelor's degree with the Department of Computer Science and Biomedical Informatics, University of Thessaly. She is also engaging in research focused on distributed ledger technology, blockchain scalability, and layer 2 solutions.

GEORGIOS P. SPATHOULAS received the Diploma degree in electrical and computer engineering from the Aristotle University of Thessaloniki, in 2002, the M.Sc. degree in computer science from The University of Edinburgh, in 2005, and the Ph.D. degree from the Department of Digital Systems, University of Piraeus, in 2013. He is currently a Laboratory Teaching Staff Member of the Department of Computer Science and Biomedical Informatics, University of Thessaly, in 2014, and he teaches in both undergraduate and postgraduate study programs of the Department. He is also a Postdoctoral Researcher with CCIS, Critical Infrastructures Security and Resilience Group, NTNU. He has coauthored more than 30 publications in peer reviewed journals and conference proceedings. His research interests include network security, privacy preserving techniques, and blockchain technology. He has also served as a program committee member for international conferences and has taken part in both national and international research programs.