Asynchronous Proof-of-Stake

Jakub Sliwinski(✉) and Roger Wattenhofer

ETH Zurich, Zürich, Switzerland
{jsliwinski,wattenhofer}@ethz.ch

Abstract. We introduce a new permissionless blockchain architecture called Cascade (Consensusless, Asynchronous, Scalable, Deterministic and Efficient). The protocol is completely asynchronous, and does rely on neither randomness nor proof-of-work. Transactions exhibit finality within one round trip of communication.

Cascade is consensusless and only satisfies a relaxed form of consensus by introducing a weaker termination property. Without full consensus, the protocol does not support certain applications, such as general smart contracts. However, many important applications do not require general smart contracts, and Cascade is an advantageous solution for these applications. In particular, the architecture can implement the functionality of a cryptocurrency such as Bitcoin, replacing Bitcoin's energy-hungry proof-of-work with a proof-of-stake validation.

1 Introduction

Nakamoto's Bitcoin protocol [12] has taught the world how to achieve trust without a designated trusted party. The Bitcoin architecture provides an interesting deviation from classic distributed systems approaches, for instance by using proof-of-work to allow anonymous participants to join and leave the system at any point, without permission.

However, Bitcoin's proof-of-work solution comes at serious costs and compromises. The security of the system is directly related to the amount of investments in designated proof-of-work hardware, and to spending energy to run that hardware. Since the system's participants that provide the distributed infrastructure (often called miners) bear significant costs (hardware, energy), the protocol compensates them with Bitcoins. However, adversaries might disrupt this scheme by bribing the miners to behave untruthfully or disrupt the reward payments.

Irrespectively of how costly Bitcoin's proof-of-work gets, this solution can only process a fixed amount of transactions in a given time period, hampering adoption and often making it infeasible to use Bitcoin at all.

To make matters worse, proof-of-work protocols assume critical requirements related to the communication between the participants regarding message loss and timing guarantees. In other words, such protocols are vulnerable to attacks on the underlying network.

C. Johnen et al. (Eds.): SSS 2021, LNCS 13046, pp. 194–208, 2021.
https://doi.org/10.1007/978-3-030-91081-5_13

In the decade since the original Bitcoin publication, researchers have tried to address the wastefulness and ineffectiveness of proof-of-work. One of the most prominent research directions is replacing Bitcoin's proof-of-work with a proof-of-stake approach. In proof-of-stake designs, miners are replaced with participants who contribute to running the system according to the amounts of cryptocurrency they hold. Alas, proof-of-stake protocols require similar communication guarantees as proof-of-work, and thus can also be attacked by disrupting the network. Moreover, proof-of-stake introduces some of its own problems. Prominently, existing proof-of-stake designs critically rely on randomness. To achieve consensus, the participants of such systems repeatedly choose a leader among themselves. Despite being random, this choice needs to be taken collectively and in a verifiable way, which complicates the problem.

Due to the way blockchains typically process transactions, participants have to wait a significant amount of time before they can be confident that their transactions are accepted by the system. For example, it usually takes around an hour for merchants to accept Bitcoin transactions as confirmed, which is unacceptable for time-sensitive applications.

In his seminal paper, Nakamoto made the crucial assumption that his system has to be able to totally order the transactions submitted to the system in order to reject the fraudulent ones. However, meeting this requirement is equivalent to solving the problem known as consensus. Nakamoto's assumption has shaped the design of blockchain systems to this day. Thus, many blockchain systems achieve consensus while not taking advantage of this powerful property, but suffering the associated costs.

Our Contribution. We relax the usual notion of consensus to extract the requirements necessary for an efficient cryptocurrency. Thus we introduce a blockchain design that is Consensusless, Asynchronous, Scalable, Deterministic, and Efficient (Cascade). We claim the protocol to offer a host of exciting properties:

Permissionless: Most importantly, Cascade offers its advantages without relying on permissioned participation. The protocol is permissionless in the same way as other proof-of-stake systems, where participants of the system freely exchange cryptocurrency tokens. Token holders run the system by validating new transactions. Additionally, any token holder can delegate the validation role to other participants, but preserving his ownership of the associated tokens.

Parallelizable: In Cascade, validators running the system can parallelize the processing of transactions. There is no limit to the number of transactions a validator can process by parallelization.

Asynchronous: Cascade does not require the messages to be delivered within any known period of time. Thus the protocol is fully resilient to all network-related threats, such as delaying messages, denial-of-service, or network eclipse attacks. An adversary having complete control of the network always

Table 1. Comparison of cascade to selected BFT/blockchain protocols. Permissioned protocols are on the left, permissionless protocols on the right. We mark all protocols providing full consensus as supporting general smart contracts, even though particular implementations might not feature smart contracts.

	PBFT [3]	HoneyBadger BFT [11]	Broadcast-based [7]	Bitcoin and Ethereum [16]	Ouroboros [8]	Algorand [4]	Cascade
Permissionless				✓	✓	✓	✓
Proof-of-work free	✓	✓	✓		✓	✓	✓
Finality	✓	✓	✓			✓	✓
Asynchronous		✓	✓				✓
Deterministic	✓		✓				✓
Parallelizable			✓				✓
General smart contracts	✓	✓		✓	✓	✓	

can delay the progress of the system (by simply disabling communication), but otherwise cannot interfere with the protocol or trick the participants in any way. Previously approved transactions cannot be invalidated and impermissible transactions cannot be approved.

Final: Under normally functioning network communication, transactions in Cascade are instantly confirmed. Confirmation is final and impossible to revert. This is in stark contrast to systems such as Bitcoin, where the confidence in a transaction being confirmed only probabilistically increases over time.

Deterministic: We assume the functionality provided by asymmetric encryption and hashing. Apart from these cryptographic necessities, Cascade is completely deterministic and surprisingly simple.

Efficient: Unlike proof-of-work, the security of the system does not depend on the amount of devoted resources such as energy, computational power, memory, etc. Instead, similarly to proof-of-stake protocols, Cascade requires that more than two-thirds of the system's cryptocurrency is held by honest participants.

Cascade does not support consensus. This prevents the protocol from supporting applications that involve smart contracts open for interaction with anybody. For example, the smart contract functionality of Ethereum cannot be directly implemented with Cascade. Many important applications (e.g., cryptocurrencies or IoT systems), do not require consensus, and Cascade offers an advantageous solution for these applications.

Table 1 compares the properties of Cascade with some of the most relevant existing BFT/blockchain paradigms. Many more protocols exist that improve some aspects, for example, many protocols improve upon PBFT. While many of these protocols are more performant and efficient than the original PBFT, they share the fundamental disadvantages of PBFT: They are not permissionless, they are not parallelizable, and in order to make progress ("liveness"), they need synchronous communication.

Table 1 shows the close relation of Cascade with broadcast-based protocols. One may argue that Cascade brings the simplicity, robustness, and efficiency of broadcast-based protocols to the permissionless domain.

In this paper we focus on the basic correctness properties of the protocol and leave in depth discussion of the scalability aspect to future work.

1.1 Relaxing Consensus

In the context of a cryptocurrency, consensus is used to solve the problem of double-spending. Suppose Alice holds one cryptocurrency coin. Now Alice sets up a transaction that transfers her coin to Bob (in exchange for a good or service). However, Alice wants to cheat, trying to simultaneously spend the same coin in another transaction to Carol. Upon receiving one (or both) of Alice's transactions, honest agents need to agree on what happens to Alice's coin, preventing Alice from doubling her money. In this context, achieving consensus consists of the following requirements:

Definition (Consensus).
Each honest agent observes some transaction from a pairwise conflicting set of transactions $\{t_0, t_1, \dots\}$.

Agreement: *If some honest agent accepts a transaction* t_i*, every honest agent will accept* t_i*. No conflicting transaction can be accepted.*
Validity: *If all honest agents observe only one transaction* t_i*, only* t_i *can be accepted by honest agents.*
Termination: *One of the transactions* t_i *will be accepted by honest agents.*

The insight leading to the relaxation, is that malicious agents do not need to enjoy any guarantees. Alice tried to cheat by issuing two conflicting transactions. Cascade does not guarantee that any of Alice's conflicting transactions will be accepted.

On the other hand, an honest agent will only create one transaction spending her coin. Thus, every honest agent will see the same candidate transaction. Hence we relax consensus to guarantee termination only for honest agents:

Definition (Cascade Consensus).

Agreement: *As above.*
Validity: *As above.*
Honest-Termination: *If all honest agents observe only one transaction* t_i*,* t_i *will be accepted by honest agents.*

Under this relaxed notion of consensus, if Alice tries to cheat, it is possible that neither Bob nor Carol will accept Alice's transaction. Some honest agents might see one of the transactions first, while others might see the other first. Then the requirement of Honest-Termination does not apply, and the transactions might stay without a resolution forever. This turn of events can be seen as Alice losing her coin due to misbehaviour.

Otherwise, Consensus and Cascade Consensus do not differ. Agreed upon results are final, conflicting results are precluded and honest transactions are accepted. Despite the difference being insignificant with respect to the functioning of a cryptocurrency, this relaxation allows Cascade to combine a large set of advantages.

1.2 Intuition

For simplicity of presentation, we describe Cascade in the terminology of a cryptocurrency and refer to the cryptocurrency managed by the protocol as the money. A more formal description follows in Sect. 3.

Transactions. A transaction transfers money from one or more inputs to one or more outputs. Inputs and outputs are money amounts paired with keys required to spend them.

Validators. In proof-of-stake systems, the agents that own some of the money in the system also run the system. These agents (validators) stay online and participate in validating transactions. In Cascade, we do not require agents to stay online and participate, but allow agents to delegate this responsibility to other agents. Every agent can be a validator. Validators sign correct transactions. The system works correctly as long as agents holding more than two-thirds of the system's money delegate to honest validators.

Confirmations. A transaction t is confirmed by the system if enough validators ack (acknowledge by signing) t. If a transaction receives enough acks, no other transaction conflicting with t can become confirmed. If a cheating Alice attempts to issue two conflicting transactions t and t' at roughly the same time, it is possible that (a) either t or t' gets confirmed (but not both), or (b) neither t nor t' are ever confirmed. Case (b) happens if some validators see and sign t, while others see and sign t'. The system might stay in this state forever with the validators' approval split between t and t'. The result is equivalent to Alice losing the money she attempted to double-spend, and does not constitute any threat to the system.

It is intuitive to verify that such a system does work correctly, if the validating power amounts are statically assigned to the validators, and a set of validators controlling more than two-thirds of the cryptocurrency obeys the protocol. Our system still works correctly when the agents can freely exchange the cryptocurrency and change the appointed validators, even in the harsh conditions of an asynchronous network. Thus, we establish a system with the participation model similar to proof-of-stake protocols, but much simpler than known proof-of-stake protocols.

2 Model

Agents and Adversary. Our blockchain is used and maintained by its participants called agents. Agents who follow the protocol are called honest. The

set of agents who do not follow the protocol is controlled by the adversary. The adversary behaves in an arbitrary (adversarial) way.

We make a standard assumption pertaining to proof-of-stake systems that the adversary always controls less than one-third of the cryptocurrency in the system. The assumption is the equivalent of assuming that the adversary controls less than one-third of the permissions in a BFT protocol, or half of the hashing power in a proof-of-work system such as Bitcoin. The idea behind the assumption is that an agent owning a large stake in a system is heavily invested in the system. While sufficiently deep pockets make it possible to disrupt any system, the proof-of-stake assumption ensures that an attack is costly and self-destructive. We introduce more concepts to state this requirement precisely in Sect. 3.4.

Asynchronous Communication. All agents are connected by a virtual network supporting a message diffusion mechanism (such as Bitcoin's network), where agents can broadcast their messages to all other agents. Like in Bitcoin, new agents can join this network to receive new and prior messages.

The network is asynchronous: The adversary controls the network, dictating when messages are delivered and in what order. Messages are required to arrive *eventually*, without any bound on the time it might take. Under such weak requirements, an adversary delaying the delivery of messages can delay the progress of an agent, but otherwise will not be able to interfere with the protocol.

Cryptographic Primitives. We assume the functionality of asymmetric encryption where a public key allows every agent to verify a signature of the associated secret key. Agents can freely generate public/secret key pairs.

We also assume cryptographic hashing, where for every message a succinct, unique hash can be computed. Whenever we say that a transaction t_2 refers to a transaction t_1, we mean that t_2 includes a hash of t_1, and as such uniquely identifies t_1.

Apart from these two cryptographic primitives, the Cascade protocol is completely deterministic.

3 Protocol

3.1 Transactions

Outputs. Outputs are the basic unit of information. Outputs are included in transactions and identify who owns how much money after the transaction was confirmed by the system.

Definition 1 (Output). *An output contains:*

- *Value: A number representing the amount of money.*
- *Owner key: A public key. The agent holding the associated secret key is the owner of the money.*

Agents can reuse their keys for multiple outputs, but for simplicity of presentation we assume that the owner key uniquely identifies a single output.

Transactions. A transaction is a request issued by an agent (or a set of agents) to transfer money to other agent(s). Outputs of a transaction identify recipients of the transaction. The transaction also indicates a validator – some agent devoted to maintaining the system.

Outputs can be associated with some identifying number, but for simplicity of presentation we assume that outputs uniquely identify the originating transaction.

Definition 2 (Transaction). *A transaction t contains:*

- *A set of inputs, where each input is an output of some previous transaction. Transaction t is said to spend these inputs.*
- *A set of outputs. The sum of values of the outputs equals the sum of values of the inputs. This sum is called the value of the transaction.*
- *Validator key: A public key. The value of the transaction is delegated to the agent holding the associated secret key (validator).*

The transaction is signed by all secret keys associated with the inputs.

The validator cannot spend the transaction outputs. After t is confirmed, the validator's signing stake increases until t's outputs are spent.

Genesis. The *genesis* is a special transaction without inputs. The genesis is hard-coded in the protocol and known upfront to every agent. The genesis describes the initial distribution of money among the original agents and the initial validators (which could or could not be the same as the original agents).

The values of all genesis outputs sum up to M, so M is the total money in the system. In this paper, we assume that M never changes.

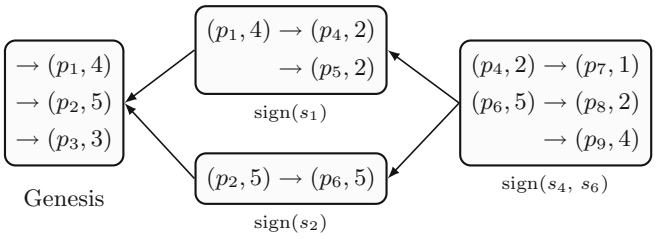


Fig. 1. Example DAG of transactions, validator keys are omitted. The p_i's are owner keys, and s_i's are the corresponding secret keys.

3.2 Validators

Validators are agents processing transactions in the system. Validators listen for transactions being broadcast, and sign them if they have not observed a conflict. An honest validator signs all non-conflicting transactions.

After a transaction t with a value of m is confirmed by the system (explained below), the "signing power" of the validator v indicated in transaction t increases by m (at the cost of the validators indicated in transactions that have output the inputs of t). To spend an output of t, the owner of an output must later broadcast a new transaction, as v cannot spend the outputs of t. An owner of an output of t can change the appointed validator v to any other validator by spending t's output (for instance by self-sending the money), when including a different validator key. Any agent can also indicate themselves as the validator.

The validator v signs transactions in the system to contribute to their confirmation, and the contribution is proportional to the amount delegated to v.

Number of Validators. Similarly to Bitcoin mining pools, the number of validators in Cascade might naturally be relatively small, such that a small number of validator's signatures is needed to confirm a transaction. The protocol can also enforce or encourage the number of validators to form groups, for example by an appropriate fee structure. In contrast to Bitcoin mining pools, the validators forming a group can maintain trustlessness with respect to each other by using an aggregatable signature scheme such as BLS [1]. In this way, a few validator pools would preserve the agency of individual validators. We believe that this is more decentralized than for example Bitcoin. Due to the page limit, we leave these aspects of the protocol to future work.

3.3 Confirmations

A validator broadcasts an *ack* message to communicate a new set of transactions the validator signed.

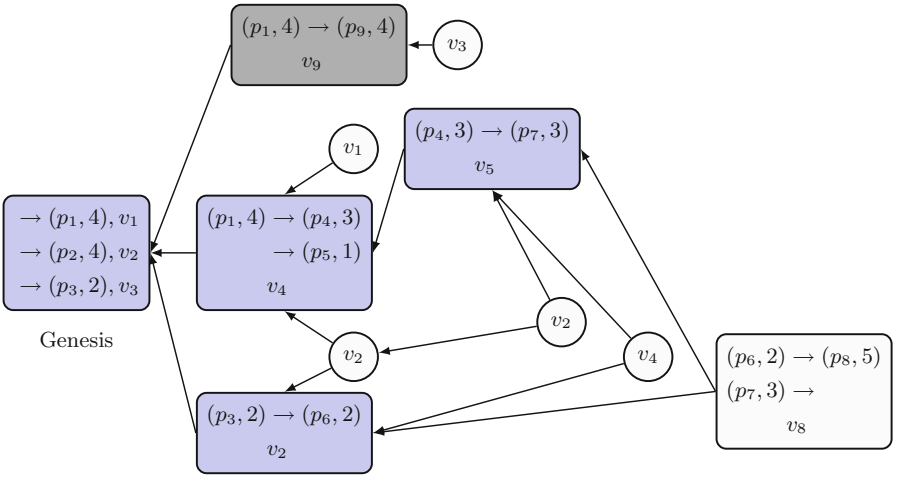
Definition 3 (ack). *An ack contains:*

- *A reference to the previous ack issued by the same validator.*
- *A set of references to transactions the validator signs.*

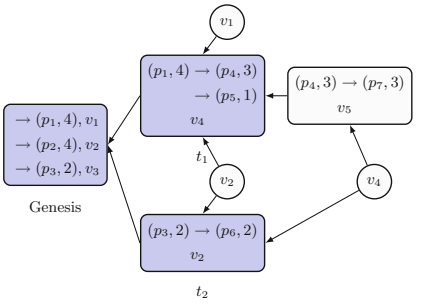
The ack is signed by the validator's secret key.

All messages can only reference previously created messages with hashes. Cyclic hash references are impossible and hence all messages form a directed acyclic graph (DAG), with the genesis being the only root. Messages are processed in any order respecting references. Agents do not process a transaction t until they have fully received $past(t)$.

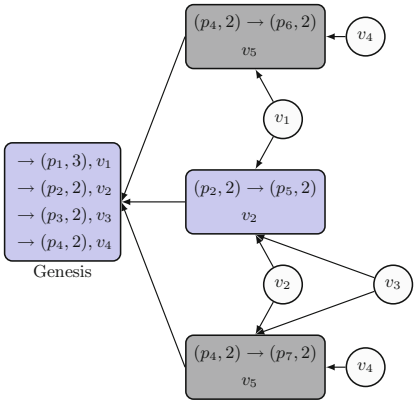
Definition 4 (past). *The set of messages reachable by following references from t is called $past(t)$. For a set of messages T, $past(T) = \bigcup_{t \in T} past(t)$.*



(a) Example transaction DAG, p_i's represent the owners and v_i's the validators. Circle nodes are acks labelled by the issuing validators. Acks point to the transactions being signed and (if available) the previous acks of the same validator. Light blue transactions are confirmed based on the acks. When issuing an ack, validators have to point to the previously issued ack, as exhibited by v_2. The dark grey transaction is an attempt at double-spending; it conflicts with a confirmed transaction and will never be confirmed. The white transaction is not yet confirmed.



(b) A subview of the transaction DAG from Figure 2a. The set A_{t_1} consisting of the acks of validators v_1 and v_2 is proof that t_1 is confirmed. The set A_{t_2} consisting of the acks of validators v_1, v_2 and v_4 is proof that t_2 is confirmed.



(c) Example attempt at double-spending. The validator v_4 is adversarial, does not reference previous acks in new acks and attempts to confirm conflicting transactions. Honest validators are split between conflicting transactions such that neither will ever be confirmed.

Fig. 2. Example DAGs.

Transactions can be confirmed by the system, and confirmation is permanent. A transaction t becomes confirmed when enough validators broadcast an ack signing it. After a transaction is confirmed, the *stake delegated* to the validator indicated in t increases by the value of t (and appropriately decreases for the validators to whom the inputs were delegated). Thus we define transaction confirmation and the stake delegated to a validator inductively (from genesis) with respect to each other. Genesis is confirmed from the start.

Definition 5 (delegated stake). *Given a set of acks A, let T_A be the set of transactions confirmed in past(A) that indicate v as the validator. The stake delegated to v in past(A) is equal to the sum of values of outputs of transactions in T_A that are delegated to v and that are unspent in past(A).*

Definition 6 (confirmed). *A transaction t is confirmed if the transactions that output the inputs of t are confirmed, and there exists a set of acks A_t such that:*

- *some validators $v_1, \ldots, v_k$ with respective delegated stake $m_1, \ldots, m_k$ in past(A_t) sign t, and $\sum_{i=1}^{k} m_i > \frac{2}{3}M$;*
- *no transaction $t' \in$ past(A_t) shares any input with t.*

Honest agents do not spend their outputs more than once, i.e. every output becomes an input at most once. Assume that t is a transaction by an honest agent. Then we will never see a transaction t' which tries to spend the same outputs as inputs of t. In this case, it is straightforward to collect validator acks for t, and eventually t will have enough acks to be confirmed.

On the other hand, if some t' is sharing inputs with t is also present in the transaction DAG, it is unclear if there can be a set A_t such that t is confirmed. It is only the misbehaving agent's concern to find an appropriate A_t and prove to the recipient of t that t is confirmed.

3.4 Adversary

The adversary behaves in an arbitrary way, and thus might create conflicting transactions, transmit acks that do not reference previously issued acks, send different messages to different recipients, etc.

Any message sent by an honest agent is immediately seen by the adversary. The delivery of each message from an honest agent to an honest agent can be delayed by the adversary for an arbitrary amount of time.

Stake. As explained in Sect. 2, we assume that the value of genesis outputs delegated to the adversary sums up to less than $M/3$. In every transaction, a new validator is indicated. Hence the stake delegated to the adversary shifts over time.

Definition 7 (adversary stake). *Let m_t^h and m_t^a be the sums of values of inputs of t that are outputs of transactions delegated to honest agents and the adversary respectively.*

When transaction t delegated to an honest agent is **confirmed** *(i.e. any A_t exists), then we subtract m_t^a from the amount we count as delegated to the adversary. When transaction t delegated to the adversary is* **issued**, *then we add m_t^h to the amount we count as delegated to the adversary.*

4 Correctness

In this section we outline the proof that the Cascade protocol upholds Cascade Consensus as defined in Sect. 1.1. The proof is available in the online version of the paper [14].

The difficulty lies in the complete asynchrony of the system. In an orthodox blockchain, all confirmed transactions are totally ordered. Such a total order does not exist in Cascade. Moreover, the stake distribution among validators is constantly shifting. The protocol prevents problems by requiring honest validators to reference previous acks. Moreover, when some transaction t shifts the stake from a validator v_1 to a validator v_2, the stake is retracted from v_1 as soon as t is observed, but only credited to v_2 when t is referenced by many other validators and confirmed.

Theorem 8 is the main result we want to prove.

Theorem 8. *The Cascade protocol satisfies Cascade Consensus.*

Under our assumption from Sect. 3.4, more than two-thirds of the money is always delegated to honest validators. Hence, if there is no double-spend alternative to a transaction t, honest validators will sign t and t will be confirmed by the system. Thus Validity and Honest-Termination of Definition 1.1 hold. Whenever any agent observes a transaction t as confirmed, the acks A_t serve as the proof that t is confirmed to any other agent. Therefore, to show that Agreement holds, it suffices to show that no pair of conflicting transactions is ever confirmed. Then the Cascade protocol satisfies Cascade consensus.

For contradiction, assume that some transaction DAG can be produced by the protocol where two conflicting transactions t_x and t_y are confirmed. Consider the instance of such a DAG G that is minimal in terms of the number of transactions.

Consider some transaction t_0 confirmed in G during the protocol's execution based solely on the stake distribution specified in genesis. We show that for any other confirmed transaction t, either $t_0 \in \mathit{past}(A_t)$ or $t \in \mathit{past}(A_{t_0})$ holds in DAG G. We conclude that t_0 cannot conflict with any transaction. Then t_0 does not serve a purpose for the construction of DAG G, as t_0's inputs could be replaced in the genesis with t_0's outputs for a smaller DAG. This contradicts with the choice of G, and Theorem 8 summarizes that under our assumptions, conflicting transactions cannot be confirmed in a single DAG.

As we mention in Sect. 5, in practice agents running Cascade would not need to precisely compute $\mathit{past}(A_t)$ for normal workloads. Every agent would confirm almost all transactions based on a lower-bound of the stake delegated to other agents computed from the observed confirmed and yet-to-be-confirmed transactions. Precise $\mathit{past}(A_t)$ might need to be computed only for some contentious transactions when there is a conflict.

5 Future Work

Due to space constraints we focussed on the basic properties of Cascade in this paper. In this section we briefly outline the aspects of Cascade we plan to discuss and expand on in the future to exhibit the advantages of the protocol.

Parallelization. Provided the topology of the workload is not inherently impossible to parallelize (such as all transactions passing the same token in a chain of transactions), validators can parallelize the signing and processing of transactions. Thus, if we increase the number of machines (with constant bandwidth each) at the validator's disposal, the throughput of Cascade increases without limit. To exclude the corner cases inherently resistant to parallelization, we state Assumption 1.

Assumption 1. *If xM is the value of honest transactions not determined to be confirmed by some honest validators yet, honest validators control more than $(\frac{2}{3} + x)M$ of the stake.*

For example, if some 5% of the system's money is being moved and unconfirmed at some instant, about 71.7% of validators need to be active to process transactions in parallel efficiently.

Signing in Parallel. Each validator v can split the space of keys between multiple servers, for example based on the first few characters of the key. The servers can independently store the spent inputs corresponding to the assigned key space.

To issue an ack signing a lot of transactions in parallel, the implementation might support splitting an ack into multiple parallel messages marked with a message count number and the same ack sequence number.

Determining Confirmation in Parallel. To determine transaction confirmation in parallel, the key space is similarly split between machines that listen to messages being broadcast in the network.

Since bandwidth is limited for individual machines, the network might simply be split into a number of subnetworks corresponding to the key space splitting.

The validators maintain the sets of confirmed and yet-to-be-confirmed transactions in their view. The set of outputs of confirmed transactions that are unspent in the set of confirmed and unprocessed transactions gives a lower bound of the stake delegated to each validator in the view. By Assumption 1, these lower bounds are enough to determine transactions as confirmed without identifying the exact sets of confirming acks A_t or the exact corresponding stake amounts.

Smart Contracts. To support smart contracts callable by arbitrary parties Cascade needs to be augmented with a consensus mechanism ordering inputs. However, such consensus mechanism would be invoked only for the inputs requiring it, where traditional BFT/blockchain protocols totally order *all* transactions, and hence introduce an inherent bottleneck in the design of a system.

The consensus overhead is only necessary for some smart contracts, only when conflicting inputs are issued at the same time, and only with respect to such

relevant inputs. Thus, a system processing mostly parallelizable content could enjoy the properties of Cascade for the most part, while resorting to consensus for the contents that require it.

Pruning the DAG. In contrast to standard blockchain systems, Cascade naturally supports checkpoints and pruning old, redundant data from the blockchain, which we discuss in the full version of the paper [14].

6 Related Work

Permissioned Systems. Traditionally, distributed ledgers [3,9] operate with a carefully selected committee of trusted machines. Such systems are called permissioned. The committee repeatedly decides which transactions to accept, using some form of consensus: The committee agrees on a transaction, votes on and commits that transaction, and only then moves forward to agree on the next transaction.

Gupta [7] proposes a permissioned transaction system that does not rely on consensus. In this design, a static set of validators is designated to confirm transactions. Our concepts (such as the use of parallelization) do work in the permissioned setting as well, and could be applied to this work.

The authors of [6] show that the consensus number of a Bitcoin-like cryptocurrency is 1, or in other words, that consensus is not needed. The paper provides an analysis and discussion of which applications rely on consensus and to what extent, all of which is directly relevant to Cascade. The authors draw parallels between permissioned consensusless transaction systems and Byzantine consistent broadcast [2,10].

HoneyBadger BFT [11] provides an asynchronous permissioned system by relying on advanced cryptographic techniques with full consensus. Again, the main differences from Cascade are that the system is permissioned, much more involved, and reliant on randomization.

The authors of [5] introduce a protocol based on reliable broadcast that allows participants to join and leave the system. However, the adversary is required to control a limited number of participants (as opposed to hashing power or stake), so the protocol cannot be applied in permissionless contexts where unknown participants can join freely. The protocol consists of a few rounds of communication to agree on nodes joining or leaving the system.

Permissionless Systems. Bitcoin [12] radically departed from the established model and became the first permissionless blockchain. In the Bitcoin system, there is no fixed committee; instead, everybody can participate. Bitcoin achieves this by using proof-of-work. Proof-of-work is a randomized process tying computational power and spent energy to the system's security, while also requiring synchronous communication. However, Bitcoin's form of consensus hardly satisfies the traditional consensus definition. Instead of terminating at any point, the extent to which the consensus is ensured raises over time, approaching but never

reaching certainty. More precisely, in Bitcoin transactions are never finalized, and can be reverted with ever decreasing probability.

Similar to Bitcoin, Cascade allows permissionless participation. In contrast to Bitcoin, Cascade does not rely on proof-of-work or randomization, features parallelizability and finality, and works under full asynchrony.

To address the problems associated with proof-of-work, proof-of-stake has been suggested, first in a discussion on an online forum [13]. Proof-of-stake blockchains are managed by participants holding a divisible and transferable digital resource, as opposed to holding hardware and spending energy. Academic works proposing proof-of-stake systems include designs such as Ouroboros [8] or Algorand [4]. Proof-of-stake blockchains solve consensus and thus do not parallelize without compromises. The reliance on synchronous communication and randomization in proof-of-stake are potential security risks. Despite avoiding these pitfalls, Cascade is also simpler.

DAG Blockchains. To increase the relatively modest throughput of Bitcoin, some proof-of-work protocols employ directed acyclic graphs in the place of Bitcoin's single chain. SPECTRE [15] is likely the closest relative of Cascade among such protocols, as it relaxes consensus similarly to Cascade. However, the similarities are largely superficial, as SPECTRE remains a proof-of-work protocol, employs different techniques, and does not share the other of Cascade's advantages. SPECTRE improves many aspects of Bitcoin, but with respect to the harsh criteria of Table 1, SPECTRE can only earn a tick at permissionless.

ABC. We have been working on the idea of building a consensusfree permissionless DAG blockchain for a few years already. A predecessor of this work [14] discusses related topics not developed here due to space constraints, such as pruning the transaction DAG, fees and money creation.

7 Conclusions

In this paper we presented Cascade, a permissionless and parallelizable blockchain protocol. Cascade provides the functionality of a cryptocurrency without consensus, without proof-of-work, without requiring synchronous communication, without relying on randomness. The protocol is scalable and exhibits finality. The design of Cascade is arguably the simplest possible design for a variety of blockchain applications.

Cascade provides an advantageous solution for applications like cryptocurrencies, where honest participants do not generate conflicting status updates. Supporting general smart contracts would require performing consensus some of the time. Adding this functionality would check the last box in Table 1.

References

1. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) Advances in Cryptology—ASIACRYPT 2001, ASIACRYPT 2001.

LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
2. Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to Reliable and Secure Distributed Programming. Springer Science & Business Media (2011)
3. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. OSDI **99**, 173–186 (1999)
4. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51–68. ACM (2017)
5. Guerraoui, R., Komatovic, J., Seredinschi, D.A.: Dynamic byzantine reliable broadcast [technical report]. arXiv preprint arXiv:2001.06271 (2020)
6. Guerraoui, R., Kuznetsov, P., Monti, M., Pavlovič, M., Seredinschi, D.A.: The consensus number of a cryptocurrency. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pp. 307–316. ACM (2019)
7. Gupta, S.: A Non-Consensus Based Decentralized Financial Transaction Processing Model with Support for Efficient Auditing. Master's Thesis (2016)
8. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
9. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. (TOCS) **16**(2), 133–169 (1998)
10. Malkhi, D., Merritt, M., Rodeh, O.: Secure reliable multicast protocols in a wan. In: Proceedings of 17th International Conference on Distributed Computing Systems, pp. 87–94. IEEE (1997)
11. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 31–42. ACM (2016)
12. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
13. QuantumMechanic (2011). https://bitcointalk.org/index.php?topic=27787.0
14. Sliwinski, J., Wattenhofer, R.: ABC: proof-of-stake without consensus (2019). http://arxiv.org/abs/1909.10926
15. Sompolinsky, Y., Lewenberg, Y., Zohar, A.: Spectre: a fast and scalable cryptocurrency protocol. IACR Cryptology ePrint Arch. **2016**, 1159 (2016)
16. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. **151**(2014), 1–32 (2014)