CLIQUESENSUS: Ephemeral Overlays for Committee-based PoS Blockchains

Alexandros Antonov
Department of Informatics
Athens University of Economics and Business
Athens, Greece
aantonov@aueb.gr

Spyros Voulgaris
Department of Informatics
Athens University of Economics and Business
Athens, Greece
voulgaris@aueb.gr

***Abstract*—Recent advancements in Proof-of-Stake (PoS) consensus algorithms have given rise to a family of protocols that demand strong consensus, requiring supermajority agreement among participating nodes. Such protocols need significant network resources due to the concurrent voting of a large number of consensus nodes. As a solution, these nodes are divided into committees, with each committee voting individually at a dedicated time slot. We introduce CLIQUESENSUS, a protocol that, given a distribution of consensus nodes into committees, lets them self-organize into small, ephemeral clusters structured in clique topologies, to accelerate the voting process, while using only a small fraction of the network resources required by widely deployed systems. Our evaluation demonstrates that our protocol exhibits rapid convergence and operates with minimal network overhead. We focus on the PoS concensus algorithm adopted by Ethereum 2.0. In addition to our protocol, we also analyze and simulate the clustering approach that Ethereum has adopted, showcasing that our protocol can reduce message dissemination time by 23% to 70%, while requiring about 190 times fewer message forwards.**

***Index Terms*—Peer-to-Peer, Gossiping, Proof-of-Stake Consensus, Ethereum Blockchain**

I. INTRODUCTION

Distributed voting has found usage in a range of decentralized applications, notably in replicated log management and database replication [1]–[3], cloud computing coordination [4], and Decentralized Autonomous Organization (DAO) governance [5], [6]. This has been emphatically stressed in recent years through the widespread adoption of sophisticated consensus protocols in the domain of blockchains, particularly those behind Proof-of-Stake (PoS) mechanisms.

Blockchains that adopt such protocols typically utilize wealth distribution to engage a moderated number of nodes in a consensus voting process, to establish properties such as liveness and finality [7]–[10], These properties are crucial for the seamless operation and usability of these blockchains.

In pure Peer-to-Peer (P2P) terms, distributed voting entails each consensus participant disseminating a single message (its vote) to all other participants. Unfortunately, this problem is far from trivial, as the number of consensus nodes can grow to very high figures. At the time of writing, 800K consensus nodes are being reported[1] for the Ethereum blockchain. This causes major scalability issues. Involving this number of nodes into concurrent dissemination through a conventional method, such as flooding, would certainly result in the depletion of network resources.

In order to address this, the entire set of consensus nodes is usually divided into committees. Each committee votes at a specific point in time, influencing the determination of the blockchain state as it has progressed until that particular moment. Such committees are formed in an ephemeral fashion. Once the voting procedure concludes, committees have already fulfilled their objectives and are subsequently disbanded.

Our work is centered on creating a substrate that facilitates efficient message dissemination among groups of consensus nodes. We specifically focus on the Ethereum blockchain [11], presenting the problem within its context and adopting its terminology.

The contributions of this paper are twofold:

1) We propose CLIQUESENSUS, a protocol that clusters nodes into small, ephemeral overlays, with each overlay corresponding to a distinct committee. The constructed overlays are structured as cliques, offering desirable properties in terms of dissemination speed and reduced message overhead.
2) We analyze and simulate the existing approach adopted by Ethereum, replicating the message dissemination procedure and its current configuration for network sizes up to 2^{20} nodes. These simulations allow for a comprehensive comparison with our proposed protocol.

The rest of this paper is structured as follows. Section II lays down the necessary background, outlining the Ethereum 2.0 components that are relevant to our research, explaining its dissemination approach, and pointing out the networking concerns stemming from its adopted approach. Section III presents our system model. In Section IV we elaborate on our protocol's design, describing its operations in detail and providing the rationale behind our design decisions. An evaluation of our protocol is provided in Section V. Section VI surveys the related work, and Section VII concludes our work.

[1]History of daily active validators: https://beaconcha.in/charts/validators

II. Background

Our work focuses on the PoS consensus protocol of Ethereum 2.0. Ethereum initiated its transition to PoS with the launch of the Beacon Chain component on December 1st, 2020, running alongside the seminal Proof-of-Work mechanism of Ethereum. The definite switch to the new consensus mechanism took place on September 15th, 2022, on an upgrade event coined as *The Merge*[2]. In the context of this work, when we mention Ethereum, we are specifically referring to the post-merge Ethereum 2.0.

A. Ethereum Consensus

Consensus in Ethereum is collaboratively carried out by a large number of decentralized nodes, known as *validators*. Anyone can start a validator by staking 32 ETH. The initial stake value opts to achieve Sybil resistance, while preserving decentralization, considering consensus speed and message overhead.

In Ethereum, time is divided into epochs, with each epoch further divided into slots. Each slot has a duration of 12 seconds, and each epoch consists of 32 slots (6.4 minutes). All validators contribute to consensus, being assigned specific roles in each epoch. By the end of an epoch, validators are reshuffled and their roles are pseudorandomly assigned anew.

More specifically, the following roles are being assigned. First, precisely one validator is appointed ***block proposer*** per slot, responsible for proposing a block in the respective slot. The validity of this block is subsequently *attested* (i.e., voted for) by other validators. Second, and in order to produce the aforementioned attestations, *all* validators are designated to serve as ***attesters*** exactly once per epoch. In fact they are equally distributed (or ± 1, to accommodate uneven division) across all 32 slots of the epoch. All attesters assigned to a given slot are further split up into as many equal-sized ***committees*** as possible, of no less than 128 members each, with a maximum of 64 committees per slot. Finally, a number of validators are additionally selected as ***aggregators*** for an epoch, which entails aggregating duties alongside the voting process. Aggregator selection follows a probabilistic approach, aiming for an average of 16 aggregators per committee, per slot.

Given that there should be at least one committee per slot, and each committee should have at least 128 validators, a bare minimum of $128 \times 32 = 4096$ validators are required for Ethereum to operate. Currently, the count of active validators stands at 800K, overwhelmingly exceeding that bare minimum. That is, there are 25K active validators associated with each slot, corresponding to 64 committees per slot, with each committee comprising 390 validators. It should be pointed out that the number of committees per slot is already maxed out at 64, which we will consider fixed for the rest of this paper.

B. Shuffling Algorithm

Splitting up validators into equal-sized subsets to be appointed attesters in the respective slots, is achieved through an intricate shuffling algorithm, known as *swap-or-not shuffling* [12]. This algorithm constitutes a crucial element for the operation of Ethereum, as well as for our work.

Given a list of elements in some initial order, this algorithm shuffles the list into a pseudorandom, yet deterministic, permutation. Given that the list of validators is publicly known through the staking process (i.e., their public keys and staked amounts are listed on-chain), every validator executes this algorithm locally to figure out its assigned committee for the current epoch.

The interesting property of this algorithm is that it can also be run selectively for a single element, and in fact in both directions. That is, one may selectively compute the new index of a given validator, as well as selectively identify which validator will acquire a given index after shuffling, without having to compute the complete permutation for all 800K validators.

Effectively, the swap-or-not shuffling algorithm constitutes a method for reshuffling validators into committees of precisely equal size in each new epoch, allowing each validator to figure out its currently assigned role in a very lightweight manner.

C. Block Generation

Block generation is carried out in three phases, all taking place in a single slot:

Proposing: The block proposer generates a new block and proposes it to the entire network by disseminating it to *all* validators.

Attesting: Attesters of this slot convey their attestations to the aggegators of their respective committees. As the identity of aggregators is not publicly disclosed, attestations have to be disseminated to a much broader group of validators, effectively also reaching the target aggregators.

Aggregating: Aggregators of this slot disseminate a message with aggregated attestations to all validators. Again, the goal is to reach the next slot's block proposer, however as its identity is not disclosed, aggregations should reach the entire network of validators.

Of these, the disseminations during the first and last phases cannot be circumvented, as it is a consensus requirement that the entire population of validators be reached.

The disseminations, however, taking place during the second phase, can be substantially improved compared to the currently adopted approach, lowering network overhead by orders of magnitude and speeding up the dissemination of attestations, yet offering high reliability guarantees for message delivery. This is the scope of our work.

D. Dissemination of Attestations: Current Approach

Ethereum employs the GossipSub protocol [13] for disseminating messages. GossipSub is a distributed Publish/Subscribe (pub/sub) protocol facilitating the establishment of interest-specific channels (*topics*), ensuring message exchange gets conducted exclusively amongst nodes interested in those particular topics.

[2]The Merge https://ethereum.org/en/roadmap/merge

The operation of GossipSub can be synopsized in the following three main features:

1) Nodes maintain distinct bidirectional links (approximately eight per node) for each pub/sub topic, forming meshes that are utilized in rapid push-based dissemination.
2) Periodically, each node performs gossip with other participants in its subscribed topics to share missed messages.
3) Each node maintains individual scores for connected peers, evaluating their positive and negative behavior. Mitigation strategies are used to address potential malicious behavior.

Ethereum establishes 64 topics for the dissemination of attestations, forming the communications backbone of the corresponding 64 committees of each epoch. Each validator randomly picks and joins two topics, establishing eight bidirectional links with arbitrary other validators in these topics. In order to keep topic membership fresh and balanced, validators leave their topics and randomly join new ones every 256 epochs.

A validator is informed regarding its upcoming role as an attester—and possibly also aggregator—at least one epoch in advance. This period allows it to prepare for vote dissemination. Attesters assigned to committee $C \in [0, 63]$ of *any slot* in the following epoch, simply need to ensure they have a link to at least one validator of topic C, which they will use as a gateway to publish their attestations to that topic. Aggregators of committee C of any slot of the following epoch need to also *join* that topic (in addition to the two they are randomly subscribed to), in order to receive the attestations that will be published in that topic.

E. Scalability Concerns

The design employed for attestation dissemination raises some important scalability concerns, discussed in this section.

First, the topics employed for vote dissemination lack optimization concerning node participation, and overlay structure. Let the total number of validators be N. There is a fixed number of 64 topics, and each node participates in two of them. Therefore, there are $\frac{2N}{64}$ nodes per topic on average (that is, $25K$ for $N = 800K$ at the time of writing), yet it serves to transmit messages to an average of only 16 aggregators per committee. This leads to unnecessary network overhead as well as increased dissemination time.

Employing a more sophisticated clustering algorithm, building finer-grained dissemination topics comprising only the validators actually participating in a committee, would yield substantial benefits. Indeed, as the total number of distinct committees in an epoch is 2048 (64 in each of the 32 slots), and validators are equally distributed across all of them, each committee comprises precisely $\frac{N}{2048}$ validators. These are just 390 nodes for $N = 800K$. That is, attestations of a committee could be disseminated by exclusively involving the 390 members of that committee, which is orders of magnitude smaller than the current setting of $25K$ nodes in a topic, still without exposing the identities of the committee's aggregators.

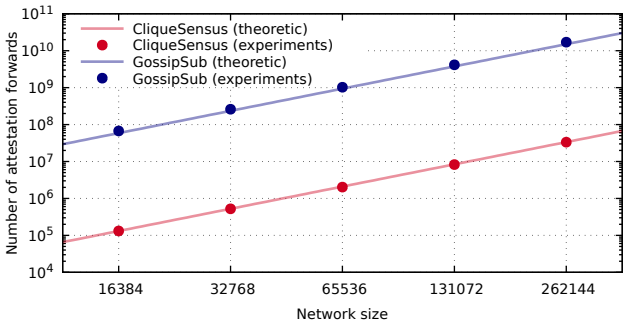


Fig. 1: Number of message forwards to disseminate all attestations in an epoch, as a function of the network size. Theoretical and experimental results for CLIQUESENSUS and GossipSub.

On a back-of-the-envelope calculation, when disseminating a message in a group of M nodes, with the assumptions that (i) each node has eight neighbors on average, (ii) upon receiving a message for the first time a node forwards it to all its neighbors except for the one it received it from, and (iii) eventually all nodes receive that message, then that message must have been forwarded roughly $7 \times M$ times during its dissemination.

In Ethereum's current use of GossipSub, every single attestation has to be disseminated in a topic of $\frac{N}{32}$ nodes, entailing $\frac{7N}{32}$ message forwards. For all N attestations in an epoch, a total of $\frac{7N^2}{32}$ message forwards are expected to take place. Indicatively, for $N = 800K$, this number rises to 140 billion forwards. Using finer-grained clustering of validators into 2048 distinct committees, each dissemination would involve $\frac{7N}{2048}$ forwards, entailing a total of $\frac{7N^2}{2048}$ message forwards per epoch, a 64-fold fewer messages. For $N = 800K$, this translates to 2.1 billion forwards.

In fact our proposed protocol achieves even better results. Given the small size of committees (a few hundred nodes), and the requirement for all-to-all communication among them, we organize them into cliques, therefore requiring only $M - 1$ messages for a single attestation's dissemination in a committee of M nodes. For N attestations in committees of $\frac{N}{2048}$ nodes, this implies a total of $\frac{N^2}{2048}$ attestation forwards per epoch. This is an astonishing order of 448 *times* fewer message forwards compared to Ethereum's current approach. For $N = 800K$ this comes down to 311 million messages.

Figure 1 plots the message estimation formulas derived above for the two protocols, along with measured data from our simulations with 2^i nodes, for $i \in [14, 18]$. The theoretical and experimental results showing a remarkable accordance between theory and practice.

A second concern is that the overlay includes validators that are mostly not part of the committee performing the ongoing dissemination. Consequently, an external validator may lack the incentives to promptly disseminate a vote from a validator belonging to another committee.

The design of our protocol takes the aforementioned design concerns into account, resulting into a protocol better suited for this specific use case.

III. System Model

We consider a set of N nodes connected over a routed infrastructure, enabling the communication between any pair of them based solely on the receiver's network address (i.e., its IP address and port).

Each node has a public/private key pair. A node's public key also serves as its unique ID. Public keys are registered in a public trusted repository, such as a public blockchain. Thus, the identities of the entire set of nodes are globally known. Yet their IP addresses are not known, let alone they may change at any time.

The network address of a node is accessible through *link records* generated by the node itself. A link record, or simply *link*, is a data structure including the node's public key, its network address, and a sequence number to distinguish fresh links from obsolete ones. To prevent masquerade attacks, node links are signed by their owner at creation time.

We assume K equal-sized committees, with each node being associated with exactly one of them. We assume a global order on nodes, determining the committee a node is associated with and its rank within that committee. Given a node's public key, anyone may locally infer the node's committee and its rank therein, at a negligible computational cost.

The global order is determined by a swap-or-not shuffling algorithm (see Section II-B) and a global seed. This seed is updated by some external oracle periodically, with a period corresponding to a consensus protocol's epoch, effectuating a permutation of nodes into a new order. Consequently, nodes are reshuffled across all K committees in each new epoch.

We assume the nodes and the network infrastructure to possess a degree of stability. Although sporadic system staggering or network congestion may lead to minor message delays, we presume that registered nodes are highly incentivized to promptly resolve these transient issues, ensuring seamless participation in the system.

The goal is for nodes to discover *all* other members of the same committee. In other words, the goal is to let nodes self-organize into a topology comprising one clique per committee. At epoch change, nodes should automatically drop their previous committees, and cluster together anew to reflect their new committee associations.

It is crucial for this process to be completed during a *preparation period*, which coincides with an epoch's duration. The shuffling algorithm produces a new permutation at the beginning of each epoch to be used in the subsequent epoch. This gives nodes one epoch's time to build the target topology, forming the corresponding links among themselves.

IV. The CliqueSensus Protocol

The ultimate goal of CliqueSensus is to let nodes belonging to the same committee self-organize into cliques. This should be achieved fast enough for committee cliques to be in place before the next epoch starts. And it should restart with every new epoch, to get the overlay ready for the subsequent epoch.

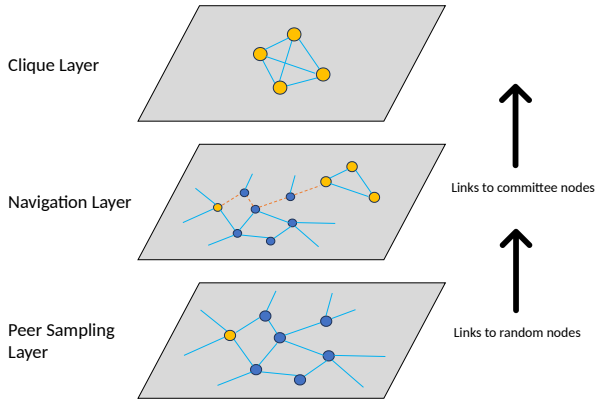


Fig. 2: CliqueSensus architecture as a three-layer protocol. Each layer constitutes a distinct gossiping protocol, maintaining its own list of neighbors and gossiping with them. Lower layers help higher layers by equipping them with links from their own neighbors.

This is a three-faceted endeavor. First, nodes need to form and maintain a single connected overlay, preventing them from getting seggregated into disjoint components, and constituting the basis on which to perform peer discovery. Second, there should be a mechanism to facilitate nodes to "navigate" towards other nodes of the same committee. Third, nodes within a committee should help each other get to know *all* members of their committee.

CliqueSensus addresses the aforementioned goals by deploying three distinct gossiping protocols, depicted as three layers in Figure 2. Each node maintains, separately per layer, a list of links to other nodes, its *neighbors*, collectively known as the node's *view* of the network. On behalf of each layer, each node periodically performs *gossip*, that is, it contacts one of its neighbors in that layer and they exchange a few links in an effort to improve their views. In CliqueSensus, the three layers are closely intertwined and operate in a cooperative manner, assisting each other in improving their views and reaching their goals faster.

Intuitively, CliqueSensus works as follows. At the beginning of each new epoch, a node has no other neighbors than a set of links to random peers provided by the peer-sampling layer. This layer is used for peer discovery. With each gossip exchange, the node progressively gets closer to contacting nodes within its own committee (navigation layer). Once nodes of the same committee meet, they drastically help each other so that they promptly become aware of all nodes in their committee (clique layer).

The following sections detail the gossiping layers comprising CliqueSensus, discussing their rationales and reasoning on their design choices. We present them in a bottom-up order.

A. Peer Sampling Layer

Peer Sampling protocols [14] form a well-studied family of gossiping protocols known for maintaining robust P2P overlays in a self-organizing manner and with strong self-healing properties.

The Peer Sampling layer serves two primary functions. First, it guarantees that the entire set of nodes remains connected in a single connected component. Second, it constitutes a continuous source of random peer samples, that is, fresh links to randomly chosen peers among all alive nodes in the network. The continuous link provision ensures the systematic replacement of old—and possibly obsolete—links by newly acquired, fresh links. This is key to peer discovery.

We have opted to use the SecureCyclon [15] version of the Cyclon protocol [16], a popular representative of the peer-sampling protocol family. Cyclon is highly scalable and lightweight in network resources, as each node is only required to retain a very small, fixed-sized view and to engage in small gossip exchanges once per cycle, irrespectively of the network size. The SecureCyclon flavor of the protocol enhances our design with resilience against malicious actions, which is particularly important in the inherently hostile environment of blockchain systems.

As shown in [16], the links in Cyclon have an average lifespan equal to the configured view length. To achieve a high link refresh rate, we opted for a low view length. As shown in our experiments, a Cyclon view length of 8 is sufficient for achieving rapid convergence in our protocol.

It should be noted that the Peer Sampling layer is the only one of the three layers that retains its view when a new epoch starts. The other two layers discard their views and start rebuilding them from scratch. This is thanks to the self-healing proerties of the Peer Sampling layer, removing stale links of departed nodes, fully incorporating newly joined nodes into the overlay, replacing old and invalid links of nodes (e.g., if their IP address changed) with fresh ones, and keeping an equal representation of all nodes in the overlay with uniformly random connectivity.

B. Navigation Layer

The Navigation layer helps nodes navigate through the overlay to reach nodes of the same committee in a timely manner.

Rather than letting nodes come across same-committee peers relying purely on random encounters, which could take unpredictably long given the high number of 2048 committees, the Navigation layer defines a proximity metric by which nodes progressively get *closer* to their same-committee peers, and eventually connect to them.

This layer employs Vicinity [17], a gossiping protocol that organizes an initially arbitrarily connected overlay into a target structure. The target structure is defined through a proximity metric, which quantifies some notion of proximity between any pair of nodes.

Following the classic gossiping paradigm, each node maintains a small number of neighbors and periodically gossips with one of them to exchange a few links. Both nodes aim to discover neighbors of as close proximity as possible. Assuming a transitive property in the specified proximity metric, the closer a node gets to its neighborhood of proximal peers, the higher its chances to discover even more of them, as its current neighbors must have also discovered other nodes in that same vicinity.

In CLIQUESENSUS, the Navigation layer models committees as vertices in a ring structure, enumerated from 0 to 2047, and defines the one-dimensional Euclidean distance, modulo 2048, as the Vicinity proximity metric. A node's goal is to pick neighbors belonging to committees close to its own. With each node sharing the same objective, the probability of a node acquiring neighbors from its committee steadily rises with the progression of cycles, as it gradually gets connected with neighbors closer to its committee.

More specifically, when two nodes are gossiping, the sender first incorporates all its current Peer Sampling neighbors in its Navigation view, and subsequently picks to send the peers that are closest to the receiver's committee. This way, every time a node gossips, either on its own initiative or in response to the gossip request of another node, it is exposed both to the other node's accummulated proximal link concentration, as well as to its own and the other node's random entropy stemming from the links in their Peer Sampling layers. The former accelerates clustering once the topology has started forming, while the latter plays a crucial jumpstart role in the early stages of clustering, by offering potential shortcuts leading nodes very close to their committees.

We deviate from the standard Vicinity protocol in two ways. First, given the ephemeral nature of clustering in our usecase, nodes do not need to discard any neighbors at the Navigation layer while clustering is underway. They are altogether discarded, however, when a new epoch starts and the clustering procedure commences all over again.

The second deviation is that, although nodes get to acquire neighbors of the same committee, they do not store them in the Navigation layer view. Instead, they hand them directly over to the Clique layer, discussed in the following section. The rationale behind this decision is that the Navigation layer's goal is to build and strengthen inter-committee paths, to point nodes of other committees in the right direction, rather than getting isolated within one's committee.

Finally, a node decides whom to contact for gossiping by picking its closest proximity neighbor. However, to maximize mingling diversity, a node does not contact the same neighbor twice before all other neighbors have also been contacted.

As far as network resources are concerned, the Navigation layer is extremely lightweight. As we will see in our evaluation (Section V), a surprisingly small value of $g_N = 3$ is sufficient for CLIQUESENSUS to converge fast to the target clustering. That is, in each gossip exchange, each node sends as few as three links to the other party, consequently exchanging negligibly small messages.

C. Clique Layer

The Clique layer enables nodes of the same committee to share their knowledge on committee membership, effectively letting nodes of the same committee self-organize into a clique structure.

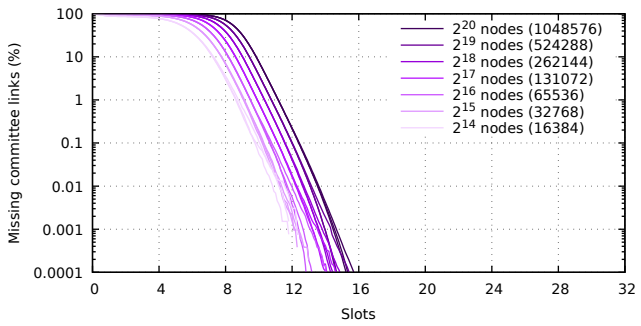


Fig. 3: Percentage of committee links still missing from the network. Seven network sizes are considered (2^i nodes for $i \in [14, 20]$) , with three individual runs for each. Different runs of the same configuration exhibit hardly any deviation, demonstrating the protocol's highly predictable behavior.

The operation of the Clique layer can be outlined as follows. Each node maintains in its Clique layer's view *all* peers of the same committee it has discovered so far. This discovery is initially triggered by the Navigation layer, which, as explained in the previous section, feeds all same-committee peers it gets to see into the Clique layer's view. Given that the Navigation layer incorporates all links acquired from the Peer sampling layer, the Clique layer also peeks into the acquired random links for potential matches with its committee. Gossiping is triggered periodically, by the Clique layer picking one of its neighbors at random.

Clique view synchronization leverages the global ordering provided by the shuffling algorithm (Section II-B), and more specifically the fact that for any given node, one can infer: (i) which committee it currently belongs to, and (ii) what its rank within that committee is. When gossiping in this layer, nodes also exchange bitmaps reporting which committee members they already know and which ones they are still missing. This also entails that a gossip exchange involves three messages. A first one from the initiator carrying its bitmap, a second one returning the receiver's bitmap and any links the initiator is missing, and finally a third message from the initiator providing links the receiver is missing.

The selection of which neighbor to gossip with is made in a round-robin fashion, in a random initial order. This maximizes the diversity of contacted nodes, thereby optimizing information exchange. Moreover, it accelerates the incorporation of nodes that were slow in discovering committee members.

Finally, the Clique layer discards its entire view by the start of a new epoch and starts building it all over again. This is an obvious decision, as the links of a past and disbanded committee are of no use in a new epoch.

V. Evaluation

We implemented both CliqueSensus and GossipSub on the *PeerNet Simulator* [18], a fork of the popular event-driven *PeerSim Simulator* [19], and we conducted an extensive set of experiments to assess their individual and comparative performance.

A. Experimental Setup

For all communication between nodes we simulated network latencies based on the WonderNetwork traces [20]. These traces report latencies from a set of 250 servers distributed globally, which recorded the round-trip times among all pairs on an hourly basis for an entire day in 2020.

We implemented and simulated GossipSub based on the specifications detailed in [21]. We parameterized GossipSub by adopting the values provided by the Ethereum specifications [22], as described in Section IV-A. We simulated the core GossipSub version 1.0. While we did not incorporate any of the security specifications from version 1.1, we did implement flood publishing, which has a positive impact on the message dissemination speed of GossipSub.

The current peer discovery mechanism in Ethereum is `Discv5` [23], which utilizes a Kademlia-based recursive lookup operation for peer sampling. Instead of implementing and deploying Kademlia, we initialized each GossipSub node with 10 links to other nodes in its committee. Given that links are bidirectional, nodes ended up having on average 20 random neighbors each within their respective committees, which is sufficient to fulfill the requirements of the GossipSub protocol.

We establish a vanilla configuration of our protocol as a point of reference, which aims for message efficiency by achieving rapid convergence with lightweight messages. In our vanilla configuration, every layer operates at the same frequency, gossiping once per cycle. The cycle is defined to coincide with an Ethereum slot (i.e., 12 seconds). The Peer Sampling layer is configured with a small view length of 8 neighbors. Finally, when gossiping, nodes in the Peer Sampling and Navigation layers send as few as $g_{PS} = 2$ and $g_N = 3$ links, parameters known as the *gossip length*.

B. Convergence Evaluation

Timely convergence is the most important requirement in our usecase. We show that, even at its most lightweight configuration, CliqueSensus manages to organize committees into cliques in just half of the required time, that is, within half an epoch.

Figure 3 shows the number of committee links nodes are still missing on average in the course of time, for network sizes of up to one million nodes (specifically, for 2^i nodes with $i \in [14, 20]$). A node should have 127 links to all other members in its committee. Initially all nodes have no committee links (missing all 127 of them), but they gradually discover committee neighbors and they eventually get to know them all. The evolution of all experiments is very fast.

Two important observations can be made in these experiments. First, different runs of the same configurations exhibit almost identical convergence, which demonstrates a highly predictable behavior for our protocol. Second, convergence time increases linearly at an exponential increase of the network size, demonstrating a strong scalability.

Figure 4 presents the time required for a fully converged overlay to emerge, as a function of the network size (in logarithmic scale). The time required for full convergence

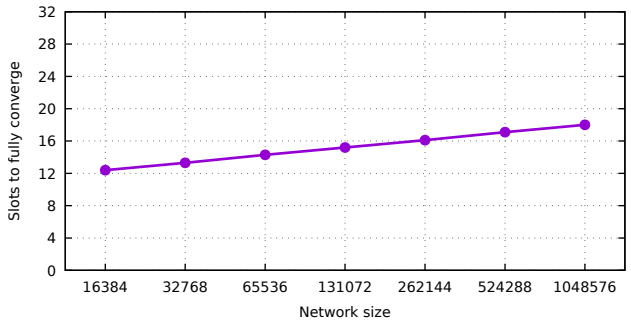


Fig. 4: Number of slots required for all cliques to be formed, for network sizes from 16K to 1M nodes (2^i with $i \in [14, 20]$).

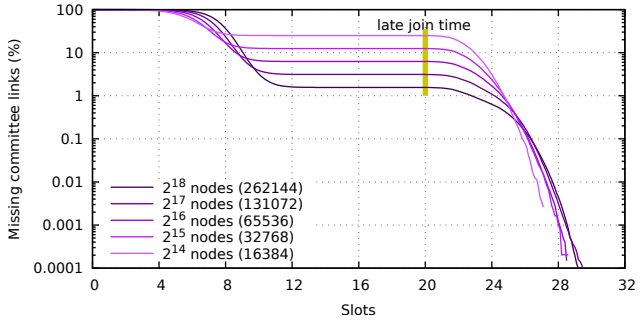


Fig. 5: Convergence for nodes joining late. In these experiments 2048 nodes join at slot 20.

is shown to closely follow the function $f(x) = log_2 x - 2$, confirming CLIQUESENSUS' logarithmic scalability. This indicates successful utilization in extremely large networks, achieving full convergence for several million nodes long before the end of an epoch.

Finally, it is also important to assess the convergence speed of CLIQUESENSUS when individual nodes join late. Figure 5 presents experiments in which 2048 nodes have been held back, joining at slot 20. As expected their convergence is faster than that of a full network's cold start, as new nodes join already formed committee overlays. This demonstrates the efficiency of the Navigation layer, which leads these nodes via educated paths to their respective committees.

C. Gossip Frequencies

In this set of experiments we explore the effect of each individual layer's gossiping frequency on the overall convergence speed. We do so by modifying the gossiping frequency of one layer at a time, while keeping the other layers' frequencies constant and equal to their default value in the vanilla version, that is, one cycle per slot.

Figure 6 presents these experiments, for $N = 262144$. It is clear that the Navigation layer exhibits the most significant improvement when gossiping at a higher frequency. Conversely, the Peer Sampling layer's gossiping frequency, does not appear to affect convergence speed. This observation suggests that the random links provided by the Peer Sampling layer at the start of a new epoch constitutes a sufficient basis to build the

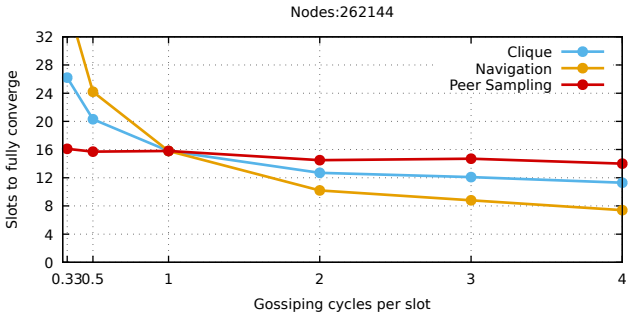


Fig. 6: Number of slots required for all cliques to be formed, for various execution frequencies of the three layers. The x-axis indicates the number of gossiping cycles a given layer executes during a slot. In our vanilla configuration, each layer executes one cycle per slot.

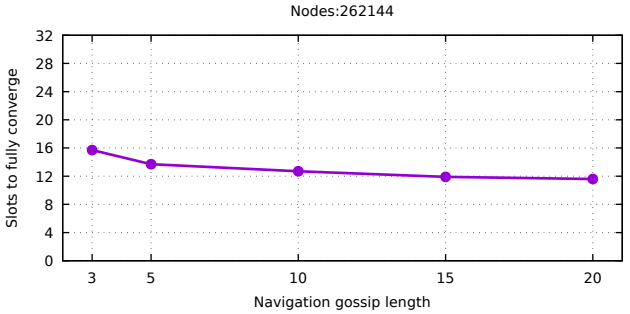


Fig. 7: Number of slots required for all cliques to be formed, for various Navigation layer gossip lengths.

target overlay, however, in later stages exchanging strategically chosen links plays a more important role than refreshing the random ones.

When increasing the frequencies of the Navigation and Clique layers, we initially observe a sharp improvement that smooths out for even higher frequencies. The flattening of the slopes underlines the importance of cooperation between both the Navigation and Clique layers in achieving swift convergence. Having a high Clique layer frequency, but low Navigation layer frequency will lead to a slow discovery of some initial committee members, with a direct negative impact on the convergence speed. The opposite will lead to a fast initial discovery, but a slow completion of each individual clique afterwards. As these two layers are closely coupled, their gossiping frequencies should be set to nearby values.

D. Gossip Lengths

Figure 7 presents the convergence time for various values of the Navigation layer's gossip length, g_N, for the network size $N = 262144$. As expected, when nodes exchange more neighbors per gossip at the Navigation layer, the target overlays emerge faster.

However, when comparing this against Figure 6, we observe that there is a higher improvement to gain by doubling the gossip frequency of the Navigation layer than by increasing its gossip length from 3 to 20, while the former introduces less

network overhead. Specifically, a gossip length of 20 leads to the transmission of 40 links for a node per cycle, on average, considering that each node gossips twice in that period, once as the initiator and once as recipient. Similarly, by doubling the gossip frequency with a gossip length of 3, the node will transmit 12 links per cycle, on average, as it will gossip four times, resulting in an average of 28 fewer links transmitted.

This observation leads us to the conclusion that, taking many, small, iterative steps towards a node's committee is more effective than making fewer, but larger iterations.

E. Overlay Construction

In this set of experiments we assess the number of messages sent for overlay construction in CLIQUESENSUS and GossipSub, considering the network size of $N = 262144$.

With respect to the CLIQUESENSUS experiments, we initiate the preparation phase as soon as each node has established links with 8 random other nodes through the Peer Sampling layer.

In GossipSub, validators learn about each other by exchanging *pub/sub* messages that contain the topics in which each validator participates. Following the pub/sub message exchange, each node attempts to establish approximately 8 mesh neighbors for its topic by exchanging *graft*/*prune* messages. We allow each topic to achieve a stable state, and when the dissemination of pub/sub, and graft/prune messages ceases, we initiate the preparation phase.

Figure 8a presents the number of messages transmitted by CLIQUESENSUS, sampled ten times per slot. The Navigation and Clique layers are observed to be generating a stable flow of messages, which peaks at approximately $520K$ messages per slot. The Clique layer starts without transmitting any messages but catches up by cycle 8, when each validator has acquired some of its committee members.

Likewise, Figure 8b presents the corresponding messages for GossipSub. These are the messages triggered by the 16 aggregator nodes per committee, which join the topics of their committees (appearing as a steep rise at the beginning of slot zero). Additionally, they include messages that correspond to nodes which leave their topics and join two new, random ones once every 256 epochs, according to the Ethereum specifications.

The cumulative message count of both protocols is presented in Figure 9. It is evident that our protocol is far more expensive, with respect to overlay construction, sending about 5.3×10^7 more messages than GossipSub in a single epoch.

This result was expected, as CLIQUESENSUS strives to create fine-grained overlays on the fly, while GossipSub relies on mostly static, large overlays. As we will see in the following section, the benefits derived from our protocol during attestation dissemination exceeds by far the cost of overlay construction.

F. Attestation Dissemination

Figure 10 presents the cumulative disseminated attestation count for all 2048 committees of an epoch, for both CLIQUESENSUS and GossipSub, for a network of 262144 nodes. In these experiments, both protocols were left to form their fully converged topologies. In CLIQUESENSUS, attestation dissemination occurred through the corresponding committee cliques, while in GossipSub, each validator was equipped with 10 gateway-links to random nodes from their committee topic. Upon the completion of topology convergence for both protocols, all validators concurrently initiated the dissemination of their attestations.

CLIQUESENSUS outperforms GossipSub by a significant difference. More precisely, CLIQUESENSUS and GossibSub concluded the dissemination with a total of $16,256$ and 8.29×10^6 messages, respectively, per committee. This accounts to a total of 3.3×10^7 and 1.7×10^{10} messages, respectively, for the whole network for the entire epoch (Figure 10). That is, CLIQUESENSUS performs attestation dissemination by sending 515 *times* fewer messages than GossipSub.

Therefore, our protocol's higher network overhead in building a fine-grained overlay pays off in terms of attestation disseminations, with network use that is orders of magnitude lighter. The excessive messages transmitted by CLIQUESENSUS during overlay construction constitute only $1/321$ of the excessive messages transmitted by GossipSub during attestation dissemination.

Concluding, factoring in both the messages for overlay construction and attestation dissemination, our protocol achieves attestation dissemination for each epoch by utilizing 191.5 times fewer messages than GossipSub.

G. Message Overhead Estimation

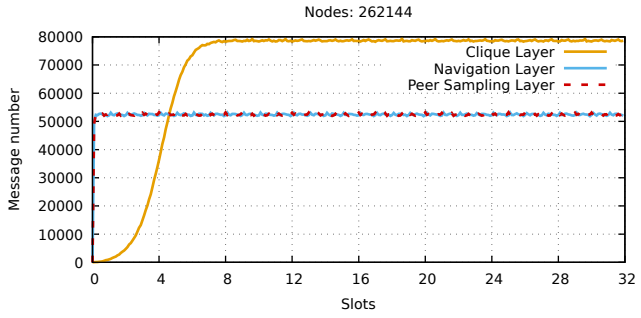
To estimate CLIQUESENSUS message sizes, we perform an informal analysis of the vanilla setting, based on the custom types defined in the Ethereum Node Record's (ENR) specification [24], for the network size $N = 262144$.

A link record in CLIQUESENSUS is of fixed size, containing the following data items: the public key of the node (264 bits), its IP address (32 bits), its port (16 bits), a sequence number that distinguishes link versions (64 bits), and a signature (512 bits). This results into a total of 888 bits per link.

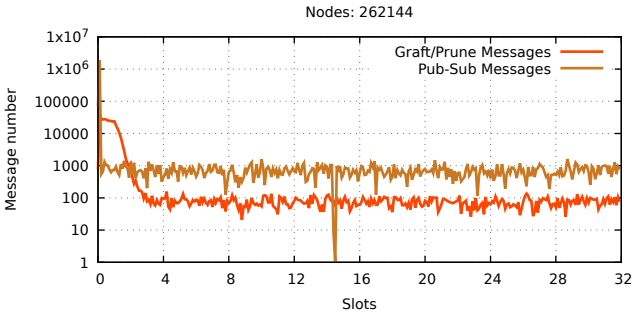
On average, a node engages in two gossip exchanges for each layer per slot, one being the initiator and one the recipient.

Combining the Peer Sampling and Navigation layers, a node consistently exchanges 10 link records per slot, on average. An upper bound for the average number of links exchanged in the Clique layer is 24 per node, per slot, as observed in Figure 11, combining for a total of 34 links. Factoring message signatures (7×512 bits), public keys (7×264 bits), and Clique layer bitmaps (2×128 bits), we conclude that an average of 4485 bytes are transmitted by a node during each slot.

Notably, as illustrated in Figure 11, outside the peak period, which only lasts for five slots (between slots 5 and 10), the Clique layer transmits only one link per gossip, on average, which leads to a total transmission overhead of 1932 bytes. This implies negligible network overhead, especially given the long, 12 sec, duration of each slot.



(a) CliqueSensus



(b) GossipSub

Fig. 8: Messages exchanged during each slot of the preparation phase, for both GossipSub and our protocol, sampled ten times per slot. The overlay of GossipSub and the Peer Sampling layer were let to converge before initiating the experiments.

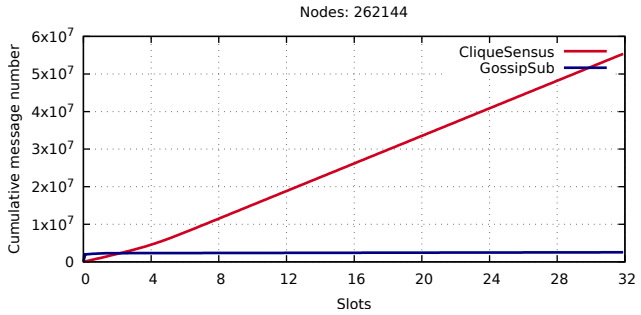


Fig. 9: Overlay construction messages (cumulative) for CLIQUESENSUS and GossipSub, for the entire network.

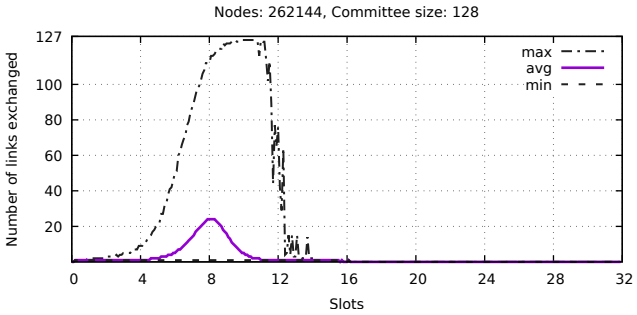


Fig. 11: Number of links exchanged in the Clique layer per node, per bidirectional gossip exchange.

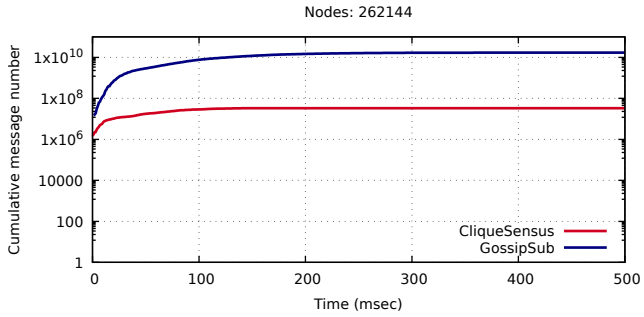


Fig. 10: Attestation dissemination messages (cumulative) for CLIQUESENSUS and GossipSub, for the entire network. Both protocols are let to converge, and in GossipSub, validators are equipped with 10 links to random members of their committee topic.

H. Dissemination Speed

Finally, we compare CLIQUESENSUS and GossipSub with respect to their performance in attestation dissemination speed.

Figure 12a presents the percentage of not-yet-delivered attestations ("missing messages"), as well as the percentage of nodes that have not received all attestations yet ("incomplete nodes"), in the course of time, for networks of 262144 nodes. Protocol initialization was conducted in the same way as in Section V-F. It is evident that the single-hop dissemination through our clique overlay accelerates attestation dissemination, resulting in the faster acquisition of votes by all validators.

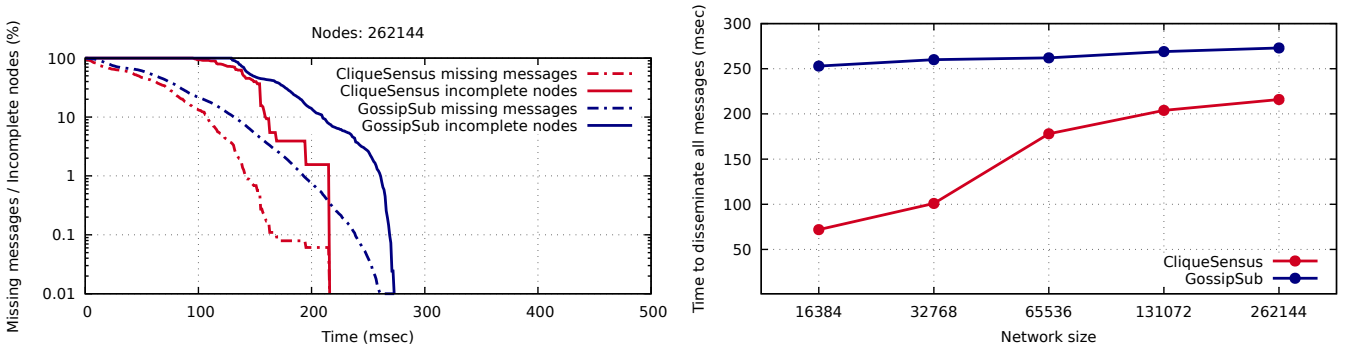
With our approach, all messages have been disseminated within 216 msec, in contrast to GossipSub's 273 msec. This represents a 21% reduction in dissemination time.

Figure 12b presents aggregate results of this type of experiment for various network sizes. It is evident that CLIQUESENSUS consistently outperforms GossipSub across all network sizes, with more substantial advantages at lower sizes, reaching up to approximately a 70% drop in dissemination time for networks of 16384 nodes.

These findings imply that CLIQUESENSUS excels in performance, especially when clique sizes become smaller. Consequently, a consensus mechanism utilizing our protocol could flexibly adapt to smaller committee sizes, particularly when rapid dissemination is imperative for achieving consensus.

VI. RELATED WORK

The closest literature to our work can be found in the domain of topic-based distributed pub/sub protocols (e.g.Poldercast [25], Spidercast [26], Tera [27], and VCube [28]). These works apply the pub/sub paradigm [29] in distributed settings, establishing topics/channels among nodes without relying on a centralized orchestration entity. Typically, such solutions exhibit remarkable scalability, combining the scalability inherent in P2P approaches, where peers have the



(a) The course of the dissemination through the voting period.

(b) Dissemination times for different network sizes,

Fig. 12: Comparison between CLIQUESENSUS and GossipSub on the speed of attestation dissemination, for a single committee. Both protocols are left to converge, and in GossipSub, validators are equipped with 10 links to random members of their committee topic.

role of both the client and the server, and the sender-receiver decoupling of the pub/sub paradigm [30]. Application-level multicast approaches, such as Scribe [31] and Bayeux [32], share similar semantics to distributed topic-based pub/sub, and often fall under the same category.

The diversity of existing approaches shows that there is no one-fits-all solution. Instead, a group-based messaging solution should be tailor-made, or carefully adapted to meet the needs of each individual system.

The work that is most closely aligned with ours is [25], which introduces a scalable distributed pub/sub approach aiming to establish deterministic topic connectivity, considering environments with high churn. Similarly to our design, Poldercast is structured as a three-layer protocol. Utilizing Cyclon as a peer-sampling mechanism, it incorporates a Vicinity-based middle layer that employs a proximity function aiming to establish connections in such a way that a node has a sufficient number of links for each topic, while keeping the sum of all links it maintains at a moderated level. The top layer constructs a ring topology for each topic, ensuring topic connectivity.

Another study which focuses on establishing topic connectivity, with scalable average node degree and topic diameter, for environments with churn is Spidercast [26]. In this approach, each node employs two heuristics to establish neighbors: first, selecting peers with the most topics in common with the node, and second, choosing random peers with at least one topic in common. To maintain an adapted view length, nodes adjust their neighbor count by adding or removing links, while considering the view length of their neighbors. This approach requires each node to retain descriptors for 5% of the total number of nodes, a sharp contrast to our approach, which accumulates descriptors for up to approximately 0.05% of all nodes. Moreover, the experiments in Spidercast were conducted with a limited node count of up to 8K nodes, which is notably low given our use case.

Scribe [31] and Bayeux [32] are two approaches which facilitate multiple individual multicast groups on top of the Pastry and Tapestry DHTs, respectively. These approaches utilize efficient multicast trees that get formed by merging DHT routes, enabling rapid, non-redundant message dissemination within each group. Nonetheless, such approaches suffer from a single point of failure, in the form of multicast roots, and the message dissemination of a group may be facilitated by nodes that are not part of the specific group.

Two centralized pub/sub approaches addressing the frequent changes in node subscriptions are Dynatops [33] and Dynamoth [34]. In Dynatops [33], a group of brokers cooperates on top of a DHT to facilitate scalable message transmission that focuses on reducing the required number of hops until a message reaches its destination. Dynamicity is confronted by a central entity, which possesses a complete view of the network, and which continuously runs a function that distributes subscribers to brokers based on two criteria: interest similarity and minimizing subscriptions per broker.

Dynamoth [34] utilizes a dynamic number of servers and load-balancing strategies to confront scenarios where topics are subject to big changes in their subscriber and publisher groups. This work establishes load-balancing in two levels: channel-level rebalancing, which establishes different communication models to confront excessive imbalances among publisher and subscribers, and system-level rebalancing, responsible for distributing the load among servers.

An approach that utilizes clique-graphs in the domain of P2P networks is eQuus [35]. This work introduces a DHT storage system designed for environments with high churn. In eQuus, nodes that are close in terms of network communication speed form cliques, sharing the same ID, and facilitating the storage of identical items. The utilization of cliques in this approach yields several desirable properties: a good degree of redundancy, as node failures are offset by other clique members possessing identical data; swift consistency, facilitated by the close proximity among clique nodes, which leads to fast item synchronization; and resilience to churn, accomplished by restricting communications to the clique-level during node entry and exit from the network overlay.

VII. CONCLUSION

Considering the problem of distributed voting in PoS-based blockchains, with a focus on Ethereum 2.0, we have introduced CLIQUESENSUS, a protocol that organizes consensus nodes

into clique-structured, ephemeral overlay clusters. Through CLIQUESENSUS, each node begins with an initial set of randomly selected, up-to-date overlay links, and progressively navigates through the network to eventually obtain links to the members of its committee. This behavior arises through self-organisation, rendering our solution inherently decentralized and highly scalable.

We evaluate our solution through extensive simulation, using Ethereum's current approach, based on GossipSub, as our point of reference. Our results demonstrate a remarkably fast convergence speed for our protocol. Moreover, CLIQUESENSUS proves to outperform Ethereum's current approach both in the number of messages transmitted and in the overall message dissemination time.

ACKNOWLEDGEMENTS

This work has been funded and supported by the Ethereum Foundation, in the context of the *Eclipse and DoS-Resilient Overlays for High-Performance Block Dissemination* project.

REFERENCES

[1] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX annual technical conference (USENIX ATC 14)*, 2014, pp. 305–319.

[2] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss *et al.*, "Cockroachdb: The resilient geo-distributed sql database," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM New York, NY, USA, 2020, pp. 1493–1509.

[3] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, pp. 1–22, 2013.

[4] A. Ailijiang, A. Charapko, and M. Demirbas, "Consensus in the cloud: Paxos systems demystified," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–10.

[5] C. Jentzsch, "Decentralized autonomous organization to automate governance," *White paper, November*, 2016.

[6] N. Diallo, W. Shi, L. Xu, Z. Gao, L. Chen, Y. Lu, N. Shah, L. Carranco, T.-C. Le, A. B. Surez *et al.*, "egov-dao: A better government using blockchain based decentralized autonomous organization," in *2018 International Conference on eDemocracy & eGovernment (ICEDEG)*. IEEE, 2018, pp. 166–171.

[7] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," *arXiv preprint arXiv:2003.03052*, 2020.

[8] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual international cryptology conference*. Springer, 2017, pp. 357–388.

[9] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[10] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. dissertation, University of Guelph, 2016.

[11] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[12] V. T. Hoang, B. Morris, and P. Rogaway, "An enciphering scheme based on a card shuffle," in *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, 2012, pp. 1–13.

[13] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, "GossipSub: Attack-resilient message propagation in the Filecoin and ETH2.0 networks," *arXiv preprint arXiv:2007.02754*, 2020.

[14] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, p. 8–es, 2007.

[15] A. Antonov and S. Voulgaris, "Securecyclon: Dependable peer sampling," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2023, pp. 1–12.

[16] S. Voulgaris, D. Gavidia, and M. van Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, June 2005.

[17] S. Voulgaris and M. V. Steen, "Vicinity: A pinch of randomness brings out the structure," in *Middleware 2013: ACM/IFIP/USENIX 14th International Middleware Conference, Beijing, China, December 9-13, 2013, Proceedings 14*. Springer, 2013, pp. 21–40.

[18] (2011) Peernet simulator. [Online]. Available: https://github.com/PeerNet

[19] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE, 2009, pp. 99–100.

[20] (2020) Wonder network traces. [Online]. Available: https://wonderproxy.com/blog/a-day-in-the-life-of-the-internet/

[21] (2024) Gossipsub specifications. [Online]. Available: https://github.com/libp2p/specs/tree/master/pubsub/gossipsub

[22] (2024) Ethereum 2.0 specifications p2p. [Online]. Available: https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md

[23] (2024) Ethereum peer discovery service. [Online]. Available: https://github.com/ethereum/devp2p/blob/master/discv5/discv5.md

[24] (2017) Eip-778, ethereum node records (enr) specification. [Online]. Available: https://eips.ethereum.org/EIPS/eip-778

[25] V. Setty, M. Van Steen, R. Vitenberg, and S. Voulgaris, "Poldercast: Fast, robust, and scalable architecture for p2p topic-based pub/sub," in *Middleware 2012: ACM/IFIP/USENIX 13th International Middleware Conference, Montreal, QC, Canada, December 3-7, 2012. Proceedings 13*. Springer, 2012, pp. 271–291.

[26] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, 2007, pp. 14–25.

[27] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "Tera: topic-based event routing for peer-to-peer architectures," in *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, 2007, pp. 2–13.

[28] J. P. de Araujo, L. Arantes, E. P. Duarte Jr, L. A. Rodrigues, and P. Sens, "Vcube-ps: A causal broadcast topic-based publish/subscribe system," *Journal of Parallel and Distributed Computing*, vol. 125, pp. 18–30, 2019.

[29] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[30] A.-M. Kermarrec and P. Triantafillou, "Xl peer-to-peer pub/sub systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 2, pp. 1–45, 2013.

[31] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications*, vol. 20, no. 8, pp. 1489–1499, 2002.

[32] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, 2001, pp. 11–20.

[33] Y. Zhao, K. Kim, and N. Venkatasubramanian, "Dynatops: A dynamic topic-based publish/subscribe architecture," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013, pp. 75–86.

[34] T. Durieux and M. Monperrus, "Dynamoth: dynamic code synthesis for automatic program repair," in *Proceedings of the 11th International Workshop on Automation of Software Test*, 2016, pp. 85–91.

[35] T. Locher, S. Schmid, and R. Wattenhofer, "equus: A provably robust and locality-aware peer-to-peer system," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*. IEEE, 2006, pp. 3–11.