HYPERBOLIC

The Next-Generation Decentralized Microservices Platform

Hyperbolic Labs

1 Vision & Mission

The Internet has made the world more connected than ever. With years of efforts, companies have developed software stacks for cloud services that power the user-end products. Ironically, together with the backing hardware resources, they are mostly limited to the corporation boundaries, forming dispersed islands that effectively isolate the Internet. This business structure, while being profitable for decades, gradually shows the signs of problems, with the increasing demand for more data control and privacy by the users, more flexible and reusable services by the developers, and more opportunities of providing hardware resources for computation by the investors.

The so-called "Web 3" [6] initiative, driven by blockchains and exemplified by Ethereum, aims to host the computation of all services on a decentralized platform, backed by a chain (ledger), replicated by all validators. Nevertheless, encoding each single step of the computation as a transaction and interpreting all transactions on the chain is fundamentally impractical to realize existing services from the traditional cloud computing, as they vary in terms of performance demand, trust model and fault tolerance. As a result, various "Layer-2" solutions, bridges, and oracles have been proposed and built to move things off-chain again. However, the absence of a more foundational, expressive, and fault-tolerant computational paradigm has once again turned these solutions into pre-blockchain isolated islands.

We introduce Hyperbolic, a high-performance microservices platform driven by a new chain-less abstraction for decentralized computing, fabricated together with an Internet-scale mesh network. It unifies diverse services and applications, fostering a connected, interoperable, and composable digital computing world.

Hyperbolic's relay network enables permissionless participation and self-healing resilience. Nodes in the network forward data to others in a peer-to-peer manner while receiving rewards as incentives for their work. Utilizing a smart routing protocol, the network autonomously recovers from failures, ensuring uninterrupted end-to-end connectivity during disruptions. This design cultivates a healthy ecosystem that promotes active participation and serves as a resilient and efficient alternative to traditional corporate-wide networks.

On top of the network fabric, Hyperbolic facilitates the development and deployment of a wide spectrum of composable Decentralized Services (D-Services) that follow a new Invoke-Return-Combine (IRC) paradigm. These D-Services are hosted by a dynamic subset of nodes, making trade-offs between performance and fault tolerance according to the nature of the individual service. D-Services follow a classic client-server model with server-side fault tolerance. Each client talks to multiple gateway servers to access the service with redundancy, but also conducts client-side computation to locally work out the output given multiple servers' responses. This unique two-level design of D-Services and relay network reduces unnecessary resource consumption (e.g. bandwidth, CPU, storage), to horizontally scale much better than a purely P2P solution. To help developers build their D-Services, Hyperbolic will offer useful, basic D-Services as node modules, such as Content Delivery Network (D-CDN), video streaming (D-Streaming), Publish-Subscribe patterns (D-PubSub), and blockchain ledger (D-Ledger) out of the box. Developers who are building different products can utilize these microservices, offered by Hyperbolic nodes, to compose larger services for their applications, which again follow the same IRC pattern. As a result, they can contribute/lease their useful D-Services to others, fostering an ecosystem of reusable modules.

Hyperbolic's versatile platform advocates numerous applications across multiple domains, such as social networks, document collaboration apps, AI training/inference infrastructures, AI data collection and cleaning tools, Multi-Party Computation, and even blockchains. Its flexibility makes it an ideal choice for both small startups and well-established organizations, promoting innovation and collaboration in a loosely organized, inclusive manner with customizability. We also develop Hyllo, our first-party messaging app that functions as a noncustodial, open-standard messaging app bound to cryptographic identities. More importantly, Hyllo serves as a portal to Hyperbolic, allowing end users to directly interact with applications based on D-Services seamlessly. It is the browser of our Hyperbolic web services.

2 HYPERBOLIC: THE MICROSERVICES PLATFORM

After years of research and development experience in areas including Cloud-computing, AI, Fintech, and Blockchains, we have come to realize a fact that many may have overlooked: the current philosophy of distributed computing infrastructure has reached its limits in terms of flexibility, versatility, security, and privacy (left of Figure 1). On the other hand, the alleged Cloud-computing ("Web 2") killer, blockchains, while solving the trust issue of decentralization, have taken a more rudimentary infrastructure design than Cloud-computing (middle of Figure 1). Since Ethereum, projects and developers have been painstakingly trying to directly squeeze the computational logic onto the chain as much as possible. Ironically, the chain, being overloaded with application traces as transactions, struggle to even enable traditional applications from the less celebrated Web 2 world with promised additional security guarantees, leaving the main usable applications mostly DeFi/NFT to the general public. Admit it or not, the relatively small market is far from fulfilling the futuristic picture of "Web 3".

Most of us, do acknowledge the deficiency of having a single blockchain and may advocate to move things "off chain" again or resort to Layer-2 solutions, further complicated by bridges and oracles needed to let the chains talk to each other, and to the real world. However, by taking a step back and rethink, the crux of the problem is *never* about debating over on-chain vs. off-chain solutions, but having a well-established, disciplined paradigm that organizes many decentralized computation activities that do not need to frequently consult the ledger. As an analogy, Ethereum contributed a lot to the blockchain world because it has successfully established a paradigm, or a computational model, that led developers to think in the way of "smart contracts", their states, methods and interactions. Currently, the Web 3 world has adopted blockchain as the de-facto model for decentralized computation, but today we would like to question whether that is the only way. We propose a new model, Hyperbolic, that is more fundamental, inclusive, and opens up a lot of new possibilities. In this new view, however, blockchain consensus, like Paxos [10]/Raft [14] as to Web 2, becomes a useful service with strong consistency and ordering guarantee, which could fit straight into our ecosystem. Thus, Hyperbolic is not an L2 or ad-hoc/centralized off-chain trick, but a brand new journey of fulfilling the original dream of Web 3. Blockchains are still handy, but only as participating services of our ecosystem. Unlike those "Layer 0" initiatives, Hyperbolic is not chain-centric, and is not powered by a chain.

2.1 Bolic: Self-Healing Network Fabric

Distributed computing, decentralized or not, always happens by exchanging data in-between the participating entities (nodes). For a decentralized system, we need to have a simple, robust, and efficient infrastructure for network communication. In Hyperbolic, we first build *Bolic*, an Internet-scale subsystem that delivers arbitrary binary data from one crypto entity to another with minimal overhead. Unlike many other network/P2P libraries, Bolic is designed with modularity from the ground up and comes with careful thoughts in abstractions. As experienced distributed systems builders, we apply our best practice to the low-level library to make sure the code is lightweight but future-proof. We also realize that the important abstraction in an easy-to-use, Internet-scale network infrastructure includes the disaggregation of the following three aspects:

- The notion of data transmission from one end to another, it should be as simple as Apple's "Airdrop": you tag (and encrypt) the message with the recipient (crypto-id) and then "drop" it to Bolic, and the rest of the process is hidden from the user.
- The actual transport to transmit data in a logical hop, which could be backed by TCP/UDP/WebSocket/WebRTC/SMS/Emails, etc. Or even by the action of scanning a QR code. The transport does not know the data because it is end-to-end encrypted.
- The routing mechanism to balance the resource cost, reliability, and performance to get the data delivered to the recipient end, offering different tradeoffs and plans.

In short, our Bolic subsystem resembles building another "Internet" (the "Web" of Web 3), on top of the existing, prevailing data transmission methods, such as the current Internet, but there could also be other data carriers (or virtualized carriers such as Tor-style privacy channels). We hope to retain code minimalism to establish a data transmission and routing network with high capacity and low latency to set the ground for our Hyperbolic decentralized services.

2.2 Best of Both Worlds: P2P vs. Client-Server

It is a prevailing thought that a decentralized system should be based on peer-to-peer (P2P) communication for all stages of services, in contrast to the traditional client-server model where clients (e.g., mobile/web apps) send requests and pull data from the hosting servers. No doubt, client-server is more seen in centralized systems where servers have unequally more control and authoritative interpretation of users' data and computation, however, this model still has advantages over a pure P2P network because the server can

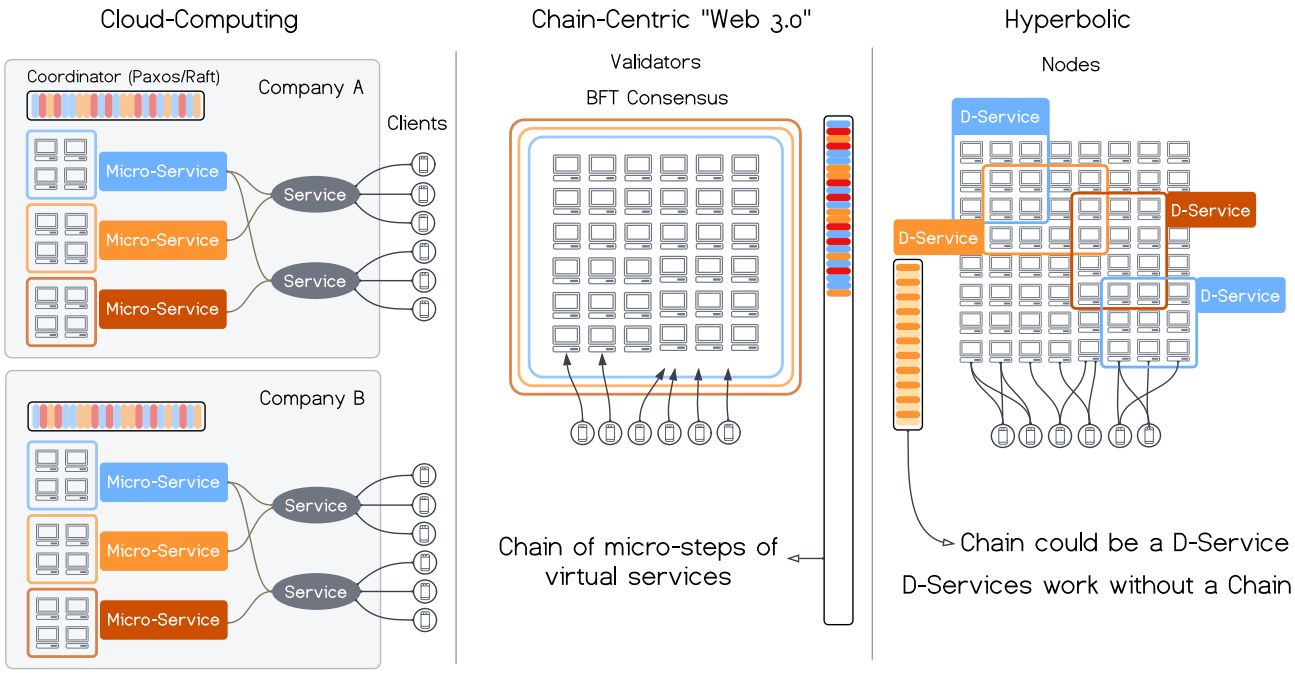

Figure 1: Traditional cloud-computing limits the reusability of both hardware resources and software stack (private services) at the company boundary and has a single point of trust. Existing blockchains alone are not enough to realize the "Web 3" dream because the chain has limited capability of encoding general computation, and lacks a disciplined way for "off-chain" operations with guarantees. Hyperbolic envisions the future of computing could be done by some carefully designed primitive that is chain-less, while incorporating chains as part of the ecosystems. We are shifting from chain-centric to service-centric.

asymmetrically and simultaneously serve multiple (hundred-thousands of) clients at a much lower cost in bandwidth and power.

Imagine the implementation of a "super" group chat, where a chat room accommodates a lot of (say, 100K) users. In this extreme case, with P2P approach, each message will need to be gossiped throughout the network to reach all recipients, resulting in high duplication of load to the system (which is the common problem in the existing design of "decentralized" messaging platforms), whereas if they all talk to the same server, the server only needs to send a message once to every other user, without blindly engaging more nodes in the network. Of course, the latter case has a single point of failure as the server could deliberately omit the message for some or all of the recipients, or it could simply refuse to render the service.

P2P and client-server approaches are like two extremes where one offers more redundancy and open-membership ("decentralization") at the expense of preventing a scalable, complex service, while the other one offers more efficient and accurate resource utilization but susceptible to failures and limits participation. This inspired us towards the Hyperbolic service model that chooses a middle ground to get benefits from both.

Back at the group chat example, in a network of 100K users, if we choose to run 100 chat servers who also join the P2P network, then we let each of 100K users, instead of directly sending to the network or to a single server, talking to 60 of these servers at a time, then we will only have 100 peers in the P2P network relaying messages to each other. In the meanwhile, we still retain very reasonable resilience to failure as each client talks to 60 servers at a time. Moreover, since any two clients will be guaranteed to have $60 \times 2 - 100 = 20$ servers in common so these servers can directly pass on messages between them without even going through the P2P network. This simple idea also resembles the concept of quorum intersections in consensus protocols that power Proof-of-Stake blockchains, but sending end-to-end encrypted messages in a virtual chat group is a much easier problem than consensus. As a comparison, a typical decentralized solution would have blindly gone for an on-chain solution which works but creates unnecessary overhead.

From this example, we demonstrated how to shrink down the cost of gossiping among 100K nodes down by 1000 times to 100 nodes while still maintaining very decent fault tolerance for an open-hosting environment where service providers can be malicious or faulty. On the other hand, this allows the platform to have computation and network capability close to traditional cloud computing, but more loosely organized and open. We envision that a good paradigm (model) should be this simple but generally applicable. Fortunately, this idea of balancing performance and fault tolerance can generalize to other services, like Pub-Sub where multiple clients subscribe to some topics and get notified.

2.3 D-Services

More formally, in our Hyperbolic network, we have all connected participants called *nodes*. Most of the nodes will have their relay ability enabled, running Bolic that forwards encrypted data in the network (Section 4 describes fee model and algorithm for routing to ensure they do the work). In addition to the role of a relay, a node can opt in one or multiple *D-Services* by running the *server part* of the logic for the D-Service, as shown by circles filled with different solid background colors in Figure 2. For a D-Service named "S", let us denote these nodes by N_s Whereas each client i (could also be a node in Hyperbolic, or just some third-party application) connects to a service-prescribed subset of servers $Q_{si} \subseteq N_s$. D-Service paradigm defines a client-server style of microservice with additional fault tolerance customizable by the designer of the service. The service is rendered via RPC-style invocations of *methods* offered by the service, whose responses get aggregated and deduplicated locally at each client. The entire *Invoke-Return-Combine* round trip is as follow:

INVOKE Each client $i \in N_c$, to initiate an RPC call for the service, it signs and directly sends a message $S.\texttt{Invoke}(i, \texttt{method}, \texttt{nonce}, \texttt{input})$ to all nodes in Q_{si}. Where `nonce` is per-invocation identifier to distinguish the calls and gets incremented each time, which helps the server to bookkeep the service state for the client. In the figure, for example, a client with the "blue" D-Service client part (a triangle) will send the same signed invocation to all nodes (represented in hexagons) that have the corresponding server part (small circles), by following the direct connections in blue lines.

RETURN Each server u, upon receiving the RPC call message, it validates the message and returns its response by signing and sending

$$S.\texttt{Return}(i, \texttt{method}, \texttt{nonce}, \texttt{input-hash}, u, \texttt{output}_u)$$

back to i, following the blue lines backwards in the figure.

COMBINE Each client i, upon the receiving at least a threshold (constant $T_{si} \leq |Q_{si}|$ predefined by the service for each client) of responses from servers:

$$R_i = \{\texttt{output}_u : S.\texttt{Return}(i, \texttt{method}, \texttt{nonce}, \texttt{input-hash}, u, \texttt{output}_u), u \in Q_{si}\}$$

, where $|R_i| \geq T_{si}$, it computes the final output locally by a service-prescribed *client part* of D-Service logic: $\text{output}_\text{i} = S.\text{Combine}(R_i)$. The D-Service designer needs to make sure as long as the number of collected respones $|R_i|$ is not less than the threshold T_{si}, the combined output is a deterministic value that is consistent regardless of the number of responses $|R_i|$.

The aforementioned group chat example can use $T_{si} = 51, \forall i \in N_c$ threshold for all clients, and run $|N_s| = 100$ servers. The predefined subset for each client is any set Q_{si} such that $|Q_{si}| = 60$. This guarantees if at least $51 \times 2 - 100 = 2$ out of 20 servers to which any two clients connect render the service, the message could be passed over directly. Even if there is no non-faulty server in common between any two clients, the clients can still rely on each one of the 60 servers to transmit the message through the Bolic P2P network.

Simple yet expressive, D-Service paradigm offers the flexibility to engage "quorum-style" fault tolerance as in Proof-of-Stake consensus protocols, and also naturally fits the idea of "aggregation-style" of cryptography like threshold signatures. But like our group chat example, one does not always have to use additional cryptography to implement such result aggregation. For chat and Pub-Sub, the message itself could be verified to see if that's from the desirable sender. With the same framework, one can describe the run of a quorum-based consensus, or even a centralized service. Here are some examples of D-Service.

BLOCKCHAIN CONSENSUS, VERSION A For blockchains, validators are running the server part of the Consensus D-Service, whereas clients could be user agents (wallet apps or developers). The $3f + 1$ validators talk to each other through the Bolic P2P network whereas the vast majority of clients can use $|Q_{si}| = f + 1$ validators (servers). $|N_s|$ could be at the level of thousands of nodes whereas $|N_c|$ could be millions and hundreds of millions. This is not just to fit the existing blockchain into our D-Service ecosystem, but actually adds more disciplined security guarantees for users: nowadays, users usually connect to some trusted source (chain API providers like Infura or user-oriented block explorers) to submit transtions and retrieve account balance/state. Ironically, although blockchain guarantees the safety for all participating validators as they work out their own local, replicated state based on consensus, this is never the case for end users. Of course, one user can manually check whether a transaction has gone through by browsing multiple explorers and some exchanges do so for deposit, it is not part of the existing blockchain infrastructure, making the users actually trust the centralized service provider, causing frauds or front-running issues. Our D-Service addresses this issue systematically.

BLOCKCHAIN CONSENSUS, VERSION B An alternative way to host a blockchain as a D-Service is to run validators both as severs and clients in our D-Service model. In this case, no traffic is poured into the P2P network and the D-Service implementor manages the

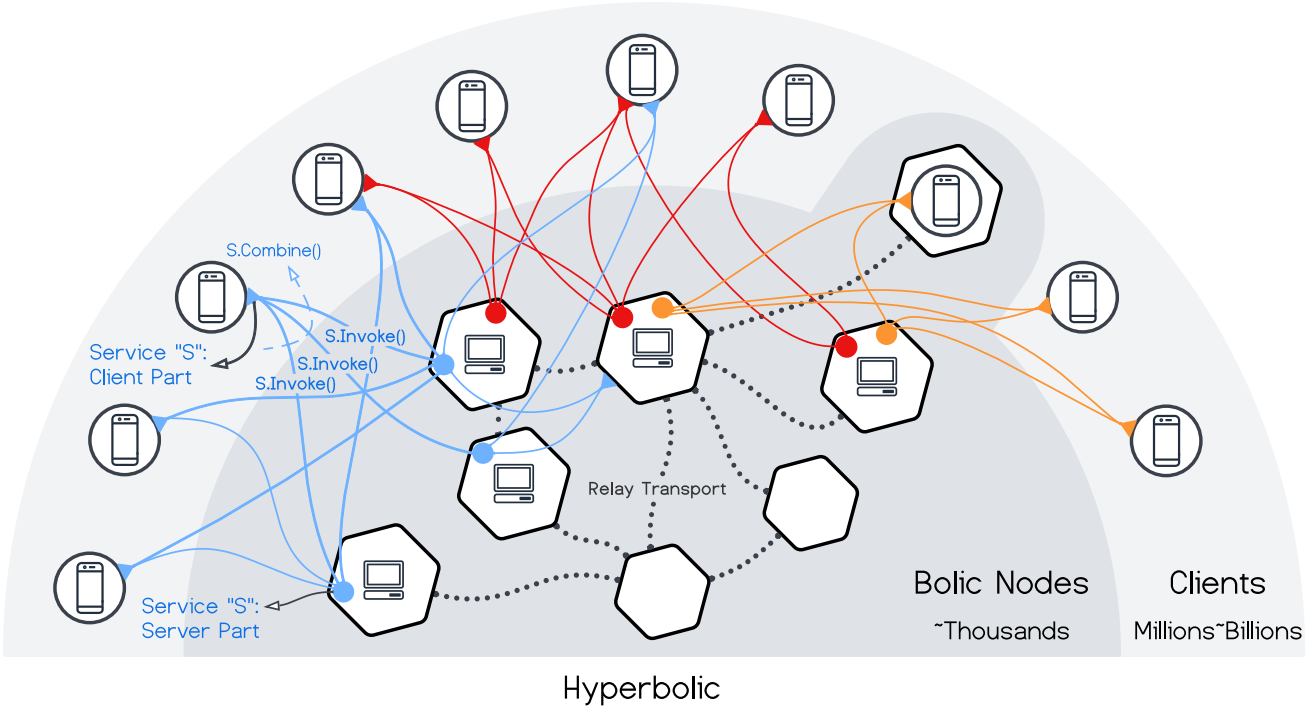

Figure 2: Hyperbolic System Architecture. Different D-Services are in different colors. One node could host the server part (circles with solid background) for multiple D-Services, whereas clients may also implement the client part for multiple D-Services.

votes in the protocol in the form of D-Service messages exchanged among the validators. Then $N_s = N_c$, and $|Q_{si}| = |N_s| = 3f + 1$, whereas $T_{si} = 2f + 1$. The S.Combine() function is defined to verify the votes and aggregate them into a quorum certificate (used in many state-of-the-art blockchain consensus mechanisms [25, 19, 20, 11]). This solution can easily turn existing blockchains into D-Services and join Hyperbolic ecosystem, without having the network traffic of the blockchain heavily load our Bolic network.

VRF BEACON Verifiable-Random-Function beacons can be smoothly described as a D-Service as well, imagine we have a typical setup of $N_s = 3f + 1$ nodes, then threshold T_{si} will just be the threshold value for VRF (say, $2f + 1$), whereas S.Combine() function will be the VRF math combination of the given output shares from beacon nodes.

CONTENT DELIVERY NETWORK (AKA. DECENTRALIZED STORAGE) In Hyperbolic, we need some decentralized storage solution that helps preserve any blobs of data used by other D-Services or users. We name our official implementation of such service *D-CDN*, as it could be viewed as a more generalized and decentralized version of the traditional Content Delivery Network (CDN). Such D-Service will store end-to-end encrypted chunks of user data and offer data availability in presence of D-Service server failure. The combine function for the client end is easy to implement as one just needs to verify the hash of the stored data and take any copy that checks out. A robust, efficient D-CDN not only benefits existing decentralized applications, but also makes it possible to render services like Netflix, Youtube, in a crowd-sourced manner.

LIVE STREAMING AND GAMING Our D-Service design enables the decentralization of those computation tasks requiring more real-time responses, which would have been impossible for a chain-centric platform. For a Twitch-like streaming service, there are usually much more audiences than the hosts, whose performance footprint is exactly captured by our D-Service design: N_c could be much greater than N_s. In this case, streaming packets can have fault tolerance and a client-end application only need one authentic copy of the packet from the video host. Together with our Hyperbolic tokenomics, we believe it not only makes live streaming more secure without sacrificing resources, but also possibly faster, as hosts (or relay nodes) are incentized to help move forward data across Bolic P2P network. In a way, Bolic by itself is like a "VPN accelerator".

AI TRAINING, FINE TUNING, AND INFERENCE With the recent advances in AI, especially Large Language Models, there has been an increasing demand in computational resources, namely GPUs. With expertise in AI/ML Systems, we envision a future of both doing large-scale, powerful computation with beefy hardware, and engaging more commodity-level devices and consumer-grade devices to be part of the game. Traditional cloud computing is limited in its turn-

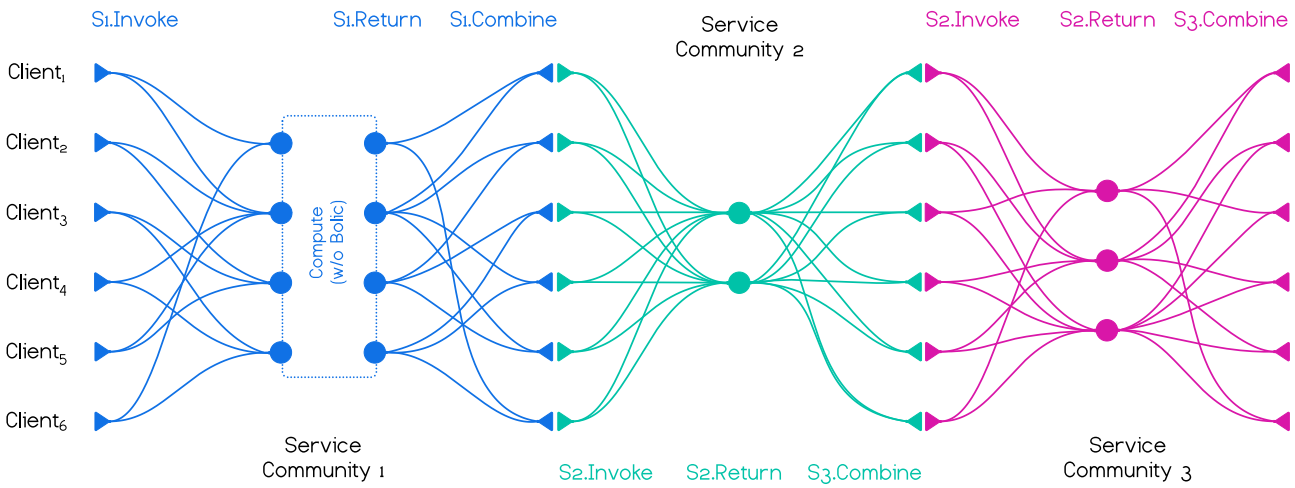

Figure 3: Clients utilize multiple D-Services one after another for the application. The output from one IRC primitive could be used as the input for the next IRC.

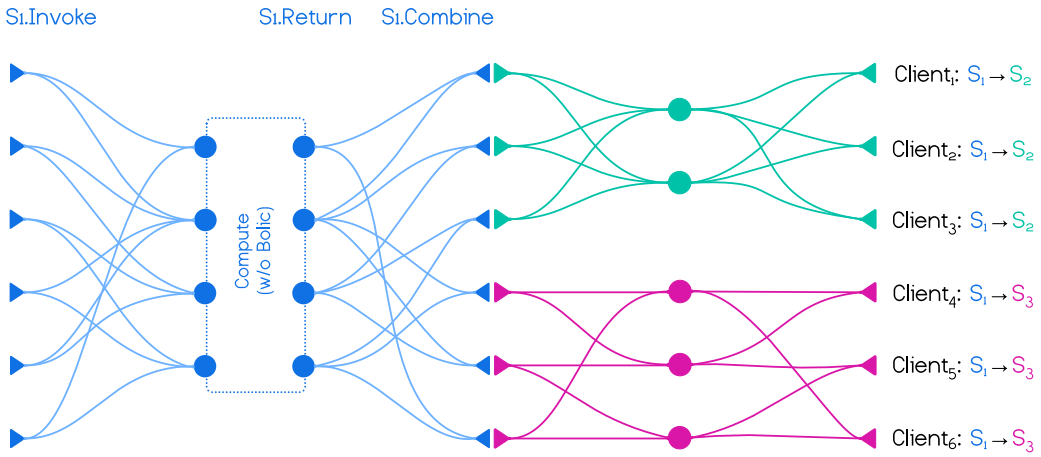

Figure 4: Different clients may choose to use different D-Services as part of their computation flows.

around cycles for new hardware and software as they are controlled by a single entity (e.g. Amazon AWS, Google Cloud, Microsoft Azure, etc.). On the other hand, many players in the market would like to invest in hardware assets and lease the computational capability to companies, developers, and users. There has been abundant research in exploring distributed training of relatively large models. We believe for those latency-insensitive cases, it is feasible and beneficial to train models in a distributed manner, opening up more opportunities for business and research. To tailor the pre-trained model to certain applications, fine-tuning a model on domain-specific data will make the model a specialist in certain tasks (e.g., customized LLM for answering legal questions). Effective fine-tuning depends on high-quality data to yield good results, hence it demands domain expertise, especially for data labeling and cleaning. Once the model is well-trained and fine-tuned, applications with large models like ChatGPT also need inference and service hosting. These tasks (data curation, model training, fine-tuning, and inference) do have different hardware and software requirements, and our vision is to also encapsulate them into D-Services and improve the interoperability of different stages in an AI training-to-product pipeline.

2.4 Invoke-Return-Combine Composability

With D-Service's Invoke-Return-Combine (IRC) primitive, one can build more complex service solutions by repeatedly applying it for each major step required in the pipeline of computation logic. The internal logic of a D-Service's server part may also contain the client part of another D-Service, and thus may invoke one or more times of the other service. As an interesting coincidence, we find this primitive resembling map-reduce primitive [4] in Machine-Learning systems where each component/operator implements the same interface and data can flow through the entire system by making each step. However, although systems like Spark [26] has notions like Resilient-Data-Set (RDD), RDD is in the realm of a beneign and domain-specific setup where the "resilience" is about

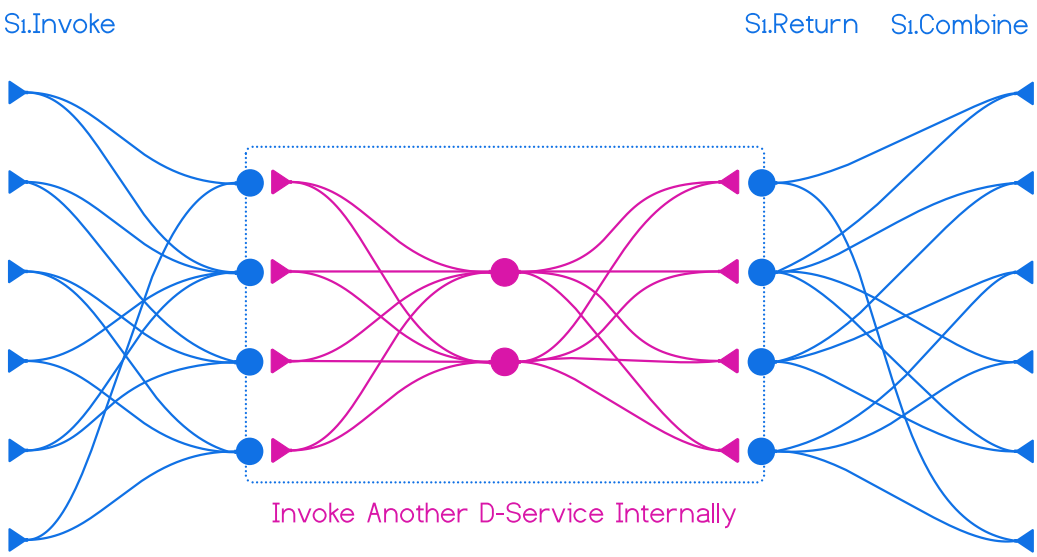

Figure 5: Like Ethereum's smart contracts, D-Services can (recursively) invoke others, thus the developers can try to build up more complex D-Services by "gluing" together basic ones.

recovering data loss, as the intermediate results in ML/AI computation can always be numerically recomputed at the expense of some extra time, whereas the authenticity or correctness is less of a concern. Our resilience, however, is Byzantine fault tolerance, and not ad-hoc to ML tasks. Unlike map-reduce, IRC process uses multiple parallel servers for their indepent outputs of the same task, and then "reduces" the outputs at each client for some customizable resilience to servers' misbehavior. The IRC primitive also resembles Ethereum's smart contracts in that different IRC implementations have interoperability, but it does not need on-chain transactions, degrades into a centralized service easily, giving much more freedom and possibilities to the infrastructure developers. On the contrary, a chain server could be wrapped into a D-Service to participate in this playground.

STAGED COMPOSITION For client app developers, the computation may invole the invocations of multiple different D-Services. Depicted in Figure 3, each client can first participate in S_1, followed by S_2, S_3, forming a staged flow of decentralized computation. The composed flow of computation offers the flexiblity that has never existed before: different stages possibly hosted by different parties, together with the difference in the nature of the task, can choose different levels of redundancy that balances the resilience and performance. Thus, the IRC primitive allows "heterogenous" fault tolerance, which is not provided by chain-centric platforms.

MIXING COMPOSITION On the other hand, clients' staged flows could differ. Like shown in Figure 4, another composability is to let D-Service like S_1 usable in different flows. For clients 1–3, they go through S_1 and S_2, whereas clients 4–6 diverges to S_3. As an example, our aforementioned D-DCN service could be such S_1 that is used in all kinds of other D-Services.

RECURSIVE COMPOSITION So far, only clients can compose one or more D-Services for their application-level purposes. The servers in a D-Service can also serve as the "client" role for some other D-Services, enabling recursive engagement of D-Services. Figure 5 demonstrates such an idea, and it is possible to apply the other two composition strategies along with this one.

2.5 The Renaissance of End-to-End Argument

Our D-Service design retains the scalability of traditional cloud-computing. Unlike blockchain platforms, one can use client-server model for computation, where clients largely outnumber the servers and easily scale horizontally. The composable D-Services make the server side also horizontally scalable. IRC-style is different, however, from a normal centralized setup in that it offloads more computational logic to the clients ("Combine" primitive and how clients interact with multiple different D-Services). Thus, compared to cloud-computing, Hyperbolic advocates a "client-centric" philosophy in that the D-Services only solve some crucial data synchronization and computation without knowing too much about the rest of the client's demand to finish up its own computation. For example, a group chat D-Service server may never know the actual contents of the messages sent to the group, thanks to end-to-end encryption.

This self-serviced, end-to-end pattern was deemed as the major merits of Web 3. Interestingly, in systems research literature, this is not new. An end-to-end argument [18] was made back in the 80s, suggesting something that is still very true for building an in-

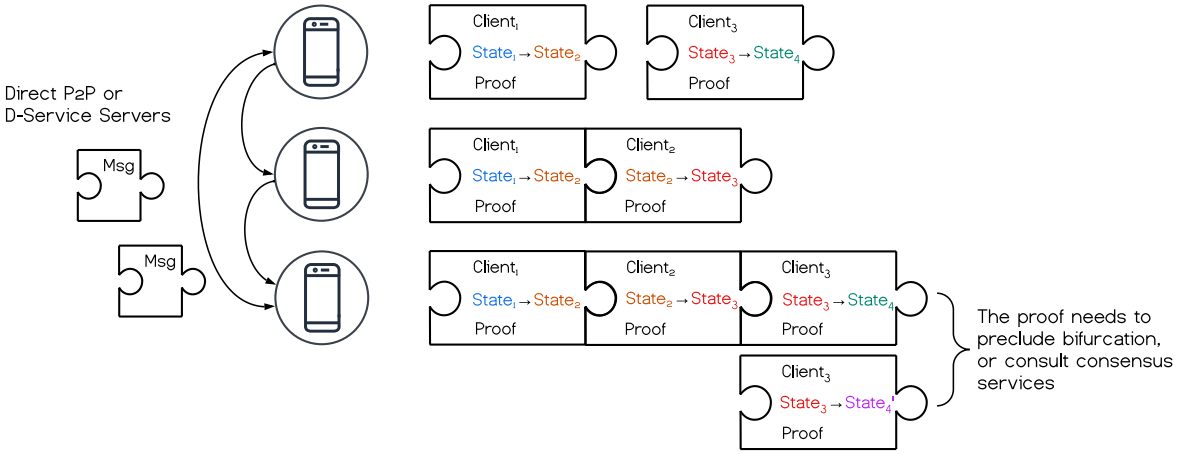

Figure 6: Client-centric, Message-carried State Machines can be locally maintained by putting together the state transition messages.

frastructure: the middleware/servers usually cannot precisely see or predict the application-level needs, but engineers can be tempted to put in more functionalities into the system for the "potential use". As a result, the system becomes bloated with functionalities that are not quite useful by the "ends" (applications). Thus, each layer in the entire software stack has to repeatedly implement something similar, with better knowledge of the application as it gets closer to the end. In contrast, an end-to-end argument says the services should be kept simple and not make too many assumptions for the additional features, leaving the flexibility to the ends.

As a further step in our Hyperbolic initiative, we would like to explore the feasibility of distributed computing by a client-centric paradigm. In the past decades, researchers have come up with a successful State Machine Replication (SMR) model that offers consistent view of computation with fault tolerance, serving as the backbone for both cloud-computing and blockchains. However, this model is platform-centric. While it has the benefits of keeping users' states in one coherent "state machine" that is replicated and interpreted by multiple servers, it creates bottlenecks in performance and scalability. In the meanwhile, we realize that many useful applications may not need server-maintained states. Some off-chain services such as Ethereum state channels, allow users to directly negotiate and/or securely sign a transaction before it is finalized on chain. The recent trend in Ethereum "rollups" is another example.

We envision a *Message-carried* State Machine (MSM), that keeps track of the computation state in a more "stateless" fashion. In this paradigm, clients keep track of their own *local* state, whereas they update their state by exchanging state-transitioning messages. As a basic example, multi-signature signing can be done in this way because there is no need to have a database (e.g., smart contracts) to keep track of the intermediate signing state. One just needs to exchange the signature shares between the signers and aggregate their own local state with the help of cryptography. Moreover, the local state may also be carried in the state transition message so that the recipients work out their local states by "chaining" the verified transitions, as shown in Figure 6. MSM model is more general as a blockchain could be viewed as an MSM where clients are the validators, and the consensus mechanism is used to resolve the possible state bifurcation shown in the figure. However, MSM offers more flexibility as some computation, such as signature aggregation used in VRF, does not have state bifurcation as any $\geq 2f+1$ shares will always generate the same output.

3 HYLLO: THE BROWSER FOR NEXT-GEN INTERNET SERVICES

3.1 More than a Natural Way of Interaction

As the first application, we would like to launch something that could both be friendly to end users and serve as the portal to our Hyperbolic platform, connecting the D-Service/application developers directly to the users via a unified entry point. We believe the best way is via a text-messaging app. But unlike any existing messaging apps in the market, Hyllo is built with a different philosophy. First, it utilizes Bolic P2P network to directly send encrypted messages in a decentralized manner. For group chats, it makes use of the aforementioned Pub-Sub D-Services to enable scalable group size. Live-streaming D-Services also equip Hyllo with additional social capabilities, potentially fulfilling what "Social-Fi" dreamed of but never managed to achieve. Because Hyllo app is also a node in the network, the user can even opt in the Bolic network to help relay traffic, or join as a lightweight D-Service server, to earn a reward for the useful computation it renders. This opens up

endless opportunities for crowd-sourced computation and socialization with security and resilience.

3.2 In the Era of New AI

The recent development in Large Language Model (LLM) such as ChatGPT [15], Llama 2 [23], has started a revolution of how humans interact with computers, i.e., the automation tools. Although there has been a heated debate regarding whether the model could be viewed as Artificial General Intelligence (AGI), we believe at the minimum, the successful new models can serve as a "programming language" that takes the form of natural languages. This is revolutionary because most of the non-programmer crowd have limited knowledge or capability in harnessing the computational power of machines for their own tasks and business. On the other hand, builders and hackers also would like to be liberated from the burden of writing boilerplate code, but instead, spend more time and energy in creating and testing new algorithms and products. Thus, LLMs offer a very promising boost in productivity for all. We will soon see "chat bots" that also serve as our personal assistants in that we "program" the logic for task handling, business activities, etc. to the bots by speaking to them directly, whereas they work on the repeated tasks at an unseen efficiency based on our strategy. In Hyllo, we plan to offer a native, secure way for such Human-AI interaction, and explore the future of AI-AI interaction: imagine both sides of a business negotiation could first use bots to lay out some common ground of understanding to save time. The existing apps such as Telegram and Discord do offer an integration with the new AI, but still through a rudimentary approach. We hope to offer a more native experience in Hyllo for prompt-engineering and the human interaction with the AI-generated content.

4 INCENTIVIZATION ALGORITHMS

In order to encourage active participation of Bolic network and advocate availability of D-Services, we design a *Proof-of-Relay* scheme for Bolic network, and a *Proof-of-Storage* scheme (see Appendix A) for the D-CDN service. As developers can design and implement new D-Services, similar to smart contracts in Ethereum's ecosystem, the proof schemes for different D-Services may vary. We hope our D-CDN proof scheme will serve as an example for other developers to design the scheme suitable for their own D-Services.

In Proof-of-Relay, due to the high volume of messages within the Bolic network, requiring nodes to generate a proof for each message and then redeem for reward is impractical. To minimize the performance overhead, we randomly select nodes as a verifiable starting point to conduct a "verifiable random walk" for connectivity in each epoch. By running the challenge epochs indefinitely, it provides network-wide coverage, thereby incentivizes the network to stay healthy without stuttering the transmission. We describe the algorithm in details in the remainder of this section.

4.1 Setup and Requirements

The Proof-of-Relay scheme is built upon several existing systems and protocols:

- **blockchain** [13, 2, 17]: a customized ledger that tracks epoch-related events and keeps a record of rewards.
- **Verifiable Random Function (VRF)** [12]: a protocol allowing multiple parties to collaboratively generate a verfiable random output, whose value is determined by the majority of the participants.
- **randomness beacon** [1]: a service that regularly produces pseudo-random seeds for the VRF. A simple implementation could be using block hashes.
- **Elliptic-Curve Diffie–Hellman (ECDH)**: a key exchange protocol that allows two parties, each having an elliptic-curve public–private key pair, to establish a shared secret over an insecure channel. It is a variant of the Diffie–Hellman protocol using elliptic-curve cryptography.

We would like a design with the following constraints to make it practical:

NO RESPONSE There should be no response messages to challenge messages. This simplifies our overall design and offers more robustness in an Internet-scale deployment.

NO REAL-WORLD CLOCK We shall not rely on the concept of real-world time or a timestamp system to coordinate steps within each epoch. It helps prevent potential synchronization issues and precludes unnecessary wait time.

AGNOSTIC TO NETWORK TOPOLOGY The design should not assume there is a pre-defined or static network topology. Namely, the underlying relay path should not matter to the proof scheme. This gives flexibility to the routing algorithm employed by Bolic.

4.2 Definitions

EPOCH A predefined time cycle for the random walk of challenge messages and reward distribution. It can be a range of block heights from the blockchain.

BERNOULLI DISTRIBUTION The discrete probability distribution of a random variable which takes the value 1 with probability p and the value 0 with probability $q = 1 - p$. p is predetermined in the Proof-of-Relay scheme.

REPUTATION SCORE A numerical score that represents the reliability and performance of a node within the network. Nodes are initialized with some default score, and then their scores are adjusted over time based on their behaivor during the epoches.

4.3 Proof-of-Relay Scheme

As shown in Figure 7, the procedure for our Proof-of-Relay scheme contains the following steps:

1. Prior to each epoch, the randomness beacon emits a random value x. Consider an arbitrary Node C_0. This node then calculates the Verifiable Random Function (VRF) output y_A and its corresponding proof π_A. These values are derived from input x and C_0's private key sk_A, that is, $y_A, \pi_A \leftarrow VRF(x, sk_A)$. Node C_0 then seeds a sampler with its own y_A (optionally salted with some public information such as the crypto address of the node, as it adds more unpredictability). This sampler, with a Bernoulli distribution, generates a pseudo-random value to determine if C_0 should act as a challenger in this epoch. If selected, a challenge message m_A of random size and the node C_1 to be challenged are chosen by the same sampler with uniform distribution, in the range of size limit of the message and the range of all participanting nodes other than C_0.

2. Node C_0 encrypts m_A using an ECDH scheme involving its own private key and node C_1's public key. The encrypted message is then transmitted to C_1 through the relay network. This may involve zero to many intermediate relay nodes in the network who do not know m_A is a challenge message.

3. Upon receiving the encrypted challenge message, node C_1 decrypts it. Recognizing it as a challenge, C_1 commits the message along with a proof to the blockchain.

4. Node C_1 then computes its own VRF output y_B and associated proof π_B which is derived from the challenge message and its private key sk_B. Using y_B, it generates a new challenge message m_B and identifies the next node C_2 to challenge. Node C_1 repeats the process in step 2, sending the new challenge message m_B to C_2.

5. Node C_2, upon receipt of the encrypted challenge message from C_1, repeats steps 3 and 4. The random walk continues until either a node realizes the epoch has ended (no reward to continue) or a Time-To-Live (TTL) hard limit is reached.

6. After the epoch ends, a verifiable path of challenges $C_0, C_1, C_2, \ldots$ is established. The nodes on this path and their peers are awarded points for their reputation score as described in Section 4.4. Additionally, after each epoch, each node's reputation score is decreased by a constant. Thus, if a node fails to participate in the random walk, its reputation will decay over time. Only nodes with the reputation above a certain threshold are eligible for receiving relay rewards.

4.4 Reputation Scoring

After the epoch ends, there are paths of challenges which originate from nodes who are selected as starting points based on the Bernoulli distribution. Consider one path of challenges $\{C_0, C_1, \ldots, C_k\}$, and we denote its length as k (some nodes may appear more than once). The i-th node C_i in this path, along with its peers, will collectively receive a total of $R(i,k) = \sum_{j=0}^{k-i} 1/2^j$ reputation points, i.e., C_i, will receive reputation points calculated based on the remaining length of the path *after* that node. Nodes at the start of long paths are rewarded more than later nodes, but the reward for any node is still bounded because of the exponentially diminishing tail rewards.

Let's assume C is a node in the path of challenges and C has m peers. To incentivize nodes to maintain a well-connected network, we incorporate a node's *degree* (number of directly connected peers) into the reward distribution. Let N be the desired minimum degree for which nodes should aim. The reputation reward R is divided between node C and its peers using the following formulas:

$$\begin{aligned}\text{Reward}(C) &= R \times \frac{\min(N,m)}{\min(N,m)+m} \\ &= \begin{cases} \frac{R}{2}, & m \le N \\ R \times \frac{N}{N+m}, & m > N \end{cases}\end{aligned}$$

$$\begin{aligned}\text{Reward}(\text{Peer}) &= \frac{R}{\min(N,m)+m} \\ &= \begin{cases} \frac{R}{2m}, & m \le N \\ \frac{R}{N+m}, & m > N \end{cases}\end{aligned}$$

In order to preserve fairness and integrity during challenges, security measures are established to deter any node from prematurely broadcasting the challenge message to its peers or other colluding nodes. If a node makes an early claim regarding the identity of a challenge message before the epoch ends, both the node and its colluder will have their rewards revoked. As an incentive for honesty in the challenge, the node that reports such a violation will be awarded some

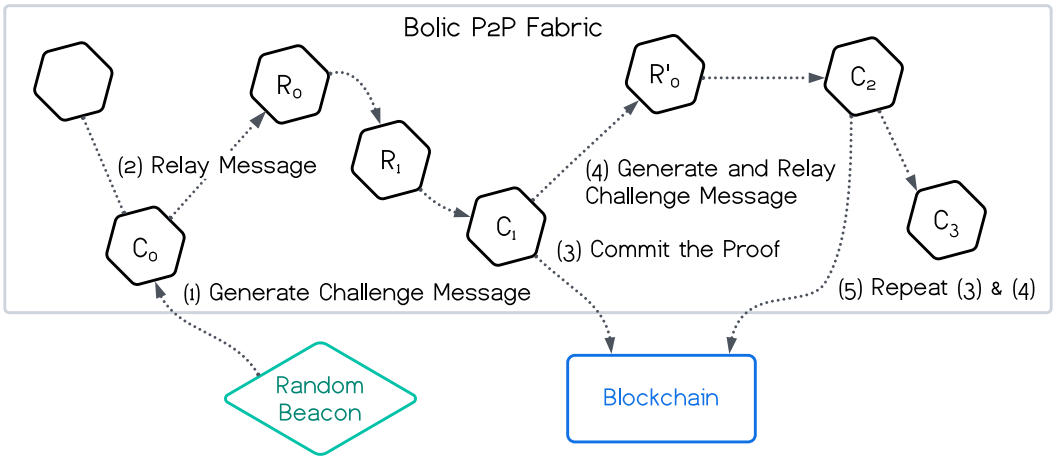

Figure 7: Schematic Diagram Illustrating the Proof-of-Relay Protocol.

rewards. To avoid an overload of such claims, each node is permitted to submit only a limited number of claims per epoch.

Overall, our scheme has the following desirable properties:

INDISTINGUISHABLE PROBING To prevent the intermediate nodes from only relaying the challenge messages, challenge messages deliberately vary in their size, and are encrypted to be indistinguishable from normal messages. This forces every node to treat and relay every messages seriously for their best interest.

UNIFORM DISTRIBUTION OF CHALLENGE MESSAGES Due to the unpredictable delivery time and continuing steps of the random walk, challenge messages are heard in the network throughout the entire epoch, so nodes cannot guess a specific time period when they could strategically omit the messages and only care for the period when there are more challenge messages.

MINIMIZING BEACON ASYNCHRONY The time span of each epoch is significantly larger than the delay of the randomness beacon. This minimizes the discrepancy between the times at which different nodes receive the random beacon, ensuring a fair start for all.

FAVOR LOW LATENCY The length of the path of challenges provides an indirect measure of network connection speeds. Within a fixed epoch, a longer challenge path indicates a shorter average time per challenge hop. Our reputation scoring design takes that into account, which incentivizes nodes to upload their proof of relay as swiftly as possible, thereby fostering network efficiency.

FAVOR STABLE CONNECTIONS Setting up connections with new nodes takes time and can potentially affect a node's ability to quickly relay challenge messages. Thus, nodes are discouraged from frequently changing their connections.

FAVOR HIGH-QUALITY PEERS The reputation scoring system motivates nodes to maintain a balanced number of high-quality peer connections. If a node C has significantly more peers than the desired degree N, its share of the reputation reward R decreases. This drives nodes to prune low-quality peers and retain only those reliable with good network traffic.

ADVOCATE RELAY DILIGENCE The confidentiality of the message source prevents nodes from knowing if they are directly connected with the challenge sender. As a result, all nodes are encouraged to diligently *relay* challenge messages, and unknowingly assist a peer, as oppose to keeping all messeages from the peer to itself.

The properties above provide a robust and fair framework for nodes to participate in and benefit from the Proof-of-Relay system.

5 REWARDS AND PAYMENTS

In Hyperbolic, we use a dual reward structure consists of both *Relay Rewards* and *D-Service Rewards*. Namely, a node first earns Relay Rewards by forwarding data as a Bolic node. Additionally, it can also earn D-Service Rewards by rendering D-Services to clients. Rewards could be in the form of Hyperbolic's native token. Also, D-Service designers can choose to use their service-related tokens or fiat money for billing, with a customized fee model tailored to the specific service.

Relay rewards are designed to encourage early adoption and participation in the network. They will initially be significant to offset the operational costs of running a node, thus ensuring node operators exceed their break-even point during the network's inception phase. Over time, as the network matures, we anticipate these rewards may gradually decrease. Relay rewards will serve as the primary source of rewards in the initial years, with D-Services income taking over

as they have more demand from the applications, and also need Bolic network hosted by the same nodes.

D-Service rewards serve as an incentive for nodes to provide D-Services. The mechanism is simple: users compensate nodes for their services, thereby incentivizing nodes to perform more work. To ensure transparency and fairness, D-Servce rewards can be disbursed to the node operators only after the successful completion of tasks or prorated for segmented periods. This approach not only safeguards the interests of the users, but also ensures that nodes are adequately rewarded for their contributions by some controllable loss from users' dishonest behavior.

From a payment perspective, users are free to utilize the Bolic relay network without incurring any charges. However, the payment mechanisms for each D-Service will differ and be independently established by their respective developers. This ensures that each D-Service can adopt a payment model that best aligns with its unique functionality, making a competitive and healthy marketplace for all.

6 OTHER RELATED SYSTEMS

In this section, we discuss related systems and projects, their features and inherent limitations, and as a comparison, why Hyperbolic is fundamentally different.

6.1 Historical P2P File-Sharing and IPFS

Gnutella was a pioneering P2P network protocol established in 2000. As the first decentralized peer-to-peer network of its kind, it set the stage for subsequent networks to adopt a similar model. Gnutella enables users to share files through a decentralized search mechanism.

BitTorrent is a popular P2P file-sharing platform that enables users to distribute large files efficiently. Many users use it to share multimedia files (movies), often with legal issues. eMule, started in 2002, offers a very similar service to BitTorrent. It uses its own algorithm and zlib for compression.

InterPlanetary File System (IPFS) is also for P2P storage and sharing. It is written in more modern, Go language and unlike the above file-sharing clients, focuses on the functionality as a "middle-ware" to other Internet-scale projects. It is plug-and-play for storage hosting machines and offers a directory-like hierarchy for file storage. It was later on incentivized by tokens like Filecoin to encourage the availability of the stored data.

However, all of the above P2P storage platforms emphasize on the possibility of storing data in a decentralized manner, so the architecture is ad-hoc to the only task. Usually the latency is less predictable and the implementation is either dated or just enough to support file-sharing. While storing data is important, there is no actual computation or interaction with the stored data on these platforms, limiting their evolution.

In contrast, the backbone of Hyperbolic is a modern P2P network, Bolic, designed to be a messaging platform and generalized for different purposes. Using this backbone, we build our D-Service framework which is not necessary in a P2P model at all stages. This allows a wide spectrum of services, including (e.g., D-CDN) but not limited to file-sharing.

6.2 Tor

Tor, also known as The Onion Router, is an anonymous communication network that enables users to browse the Internet and access services without revealing their identity or location. Tor achieves this by obfuscating traffic through a series of relays, encrypting at each step to maintain user privacy. While Tor provides a valuable tool for preserving anonymity, its focus on privacy comes at the expense of performance. It also uses the traditional web model. Hyperbolic, on the other hand, offers a new D-Service platform that can potentially accommodate Tor-style routing as an optional feature.

6.3 Content Delivery Networks

Content Delivery networks (CDNs) are large-scale distributed systems designed to serve static contents to end users with different geographical locations. Traditional CDNs are typically centralized and operated by companies. Thus, the implementations of CDNs vary and are usually private to the operators. They are also specialized in use so that streaming services like Netflix have their own video CDN servers, whereas many websites use Cloudflare [3] to cache web pages. In Hyperbolic, we build a general-purpose D-CDN that resembles a distributed key-value store with hashes as keys. Any machines in the network can opt in as a D-CDN server to store and replicate the encrypted data.

6.4 Blockchains

Blockchains, such as Bitcoin [13], Ethereum [2], and Avalanche [17], emerged as prominent P2P networked systems providing decentralized finance and/or applications in the form of smart contracts. While they demonstrate the possibilities of decentralized computing, they are built around a chain (or a collection of chains) that directly captures the computational steps of the hosted (virtual) services, which is less suitable for more general computational demands that traditional cloud-computing can satisfy. By finding the new abstraction that is lower-level than a chain-based consensus, Hyperbolic encapsulates the

existing chains as ordering services, while exposing a chain-less framework that allows more services with composability and fault tolerance.

6.5 Waku and XMTP

Waku [22] is a decentralized messaging protocol designed for privacy, scalability, and friendliness to IoT devices. Waku's messaging adopts a gossip-based routing algorithm, leading to excessive redundant messages flooding the network in both one-to-one and one-to-many communication scenarios. In contrast, the smart routing protocol of the Bolic network ensures the messages are routed to the recipient(s) with low cost whereas group messages could be handled by D-Services. XMTP is building a network to enable secure messaging between blockchain accounts, but it uses Waku under the hood to relay messages. While Waku and XMTP focus only on messaging, Hyperbolic expands the scope of decentralized systems to cater to a more comprehensive set of services and applications.

6.6 Nym

Nym [5], a Tor-like network, prioritizes user privacy with its Proof-of-Mixing design. All participants in Nym generate measurement messages in a pseudo-random manner and a full path to relay the message through the network using a VRF and a sampler. They then employ the Sphinx packet format to encrypt data and relay it along a pre-determined path through the routing network. However, this approach relies on deterministic relay paths. In contrast, our Proof-of-Relay does not make such an assumption and thus allows the Bolic network to have more flexibility in terms of the routing algorithm.

6.7 Helium

Helium [7] aims to establish a decentralized wireless network of hotspots for users. They proposed a Proof-of-Coverage scheme to verify that miners are providing wireless network coverage. Additionally, there is a Proof-of-Serialization scheme, achieving cryptographic time consensus among decentralized clients. Yet, similar to Nym, Helium uses deterministic paths for transmitting challenges. Moreover, due to the complexity of achieving synchronized timestamps, they transitioned from using Proof-of-Serialization to simply employing centralized oracles as the reference point for timestamps. Bolic's Proof-of-Relay scheme does not have these issues and provides a feasible solution that also considers relay speed.

REFERENCES

[1] Joseph Bonneau and Valeria Nikolaenko. Public randomness and randomness beacons. https://a16zcrypto.com/posts/article/public-randomness-and-randomness-beacons/, 2022.

[2] Vitalik Buterin. Ethereum: a next-generation smart contract and decentralized application platform. https://ethereum.org/en/whitepaper/, 2014.

[3] Cloudflare, Inc. Cloudflare - the web performance & security company. https://www.cloudflare.com/.

[4] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In Eric A. Brewer and Peter Chen, editors, *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150. USENIX Association, 2004.

[5] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym network: the next generation of privacy infrastructure (2021). https://nymtech.net/nym-whitepaper.pdf, 2021.

[6] Mark Fenwick and Paulius Jurcys. The contested meaning of Web3 and why it matters for (IP) lawyers. *Available at SSRN 4017790*, 2022.

[7] Amir Haleem, Andrew Allen, Andrew Thompson, Marc Nijdam, and Rahul Garg. Helium: A decentralized wireless network. *Helium Systems Inc., Tech. Rep.[Online]. Available: http://whitepaper.helium. com*, 2018.

[8] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[9] Protocol Labs. Filecoin: a decentralized storage network. https://filecoin.io/filecoin.pdf, 2017.

[10] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[11] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. Build it super simple: Introducing single broadcast consensus on a DAG. https://blog.chain.link/bbca-chain-single-broadcast-consensus-on-a-dag/, 2023.

[12] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[13] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[14] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In Garth Gibson and Nickolai Zeldovich, editors, *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, pages 305–319. USENIX Association, 2014.

[15] OpenAI. ChatGPT. https://openai.com/blog/chatgpt, 2023.

[16] Henning Pagnia, Felix C Gärtner, et al. On the impossibility of fair exchange without a trusted third party. Technical report, Citeseer, 1999.

[17] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless BFT consensus through metastability. *CoRR*, abs/1906.08936, 2019.

[18] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[19] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: DAG BFT protocols made practical. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2705–2718. ACM, 2022.

[20] Chrysoula Stathakopoulou, Michael Wei, Maofan Yin, Hongbo Zhang, and Dahlia Malkhi. BBCA-LEDGER: high throughput consensus meets low latency. *CoRR*, abs/2306.14757, 2023.

[21] Storj Labs, Inc. Storj: a decentralized cloud storage network framework. https://www.storj.io/storjv3.pdf, 2018.

[22] Oskar Thorén, Sanaz Taheri-Boshrooyeh, and Hanno Cornelius. Waku: a family of modular P2P protocols for secure & censorship-resistant communication. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 86–87. IEEE, 2022.

[23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[24] Sam Williams, Viktor Diordiiev, Lev Berman, and Ivan Uemlianin. Arweave: a protocol for economically sustainable information permanence. https://yellow-paper.arweave.dev/, 2019.

[25] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.

[26] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Steven D. Gribble and Dina Katabi, editors, *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 15–28. USENIX Association, 2012.

A APPENDIX: PROOF-OF-STORAGE

In our Decentralized Content Delivery Network (D-CDN) service, Proof of Storage plays a pivotal role in establishing trust and ensuring data availability. The crux of this process is to allow any network participant, who may not have prior knowledge or a copy of the data, to verify whether a certain storage node is faithfully storing a specific piece of data. This process is done in such a way that the verifying party does not need to download the entire data to confirm its integrity. Instead, the node storing the data provides a cryptographic proof that can be verified efficiently. By integrating Proof of Storage, our D-CDN service ensures the reliability and availability of data while minimizing network and computational overheads. This section provides a detailed explanation of our Proof of Storage protocol and its importance to our D-CDN service.

A.1 Motivations

Decentralized storage networks incorporate a variety of proof systems to ensure data integrity and availability.

Arweave [24], for instance, relies on a Proof-of-Work algorithm where nodes utilize computational resources to solve complex hash problems tied to files in storage, utilizing the full data content of a randomly

chosen prior block to validate transactions and create new blocks in the blockchain. The requirement to generate a hash below a certain difficulty level using a nonce can, however, be computationally intensive and result in unnecessary resource usage.

Filecoin [9], on the other hand, employs computationally demanding procedures, such as sealing and encoding, to generate proofs of storage, which guarantees that a node has securely created and stored a unique copy of some piece of data. Nevertheless, this technique necessitates high-end hardware due to its dependence on computationally intensive operations to produce sealed sectors. Consequently, nodes with constrained computational resources might be excluded from participating in the network.

Storj [21], meanwhile, adopts Reed-Solomon encodingto generate extra data segments, enabling the reconstruction of lost or corrupted data. Alongside, the Berlekamp-Welch error correction algorithm is used to identify and correct erroneous responses. However, Storj's verification process requires multiple data shares from different storage providers. This condition could lead to inefficiencies and dependency on the responsiveness of other nodes for data integrity confirmation.

Therefore, the need for a more efficient and accessible system for Proof-of-Storage is evident. In response, we propose a novel design for our D-CDN service, aiming to overcome these limitations. Our design only requires a proof from only the storage provider to verify the data integrity, thus eliminating the need to depend on other nodes' responses. Furthermore, by using KZG polynomial commitment scheme [8] instead of Merkle trees, the proof size is reduced to a constant and verifying the proof only require a constant number of operations. As a result, our approach is less resource-intensive, opening participation to a wider range of nodes.

A.2 Defintions

- **Reed-Solomon codes:** Error-correction codes that process data blocks as finite-field symbols and are able to detect and correct combinations of errors and erasures. Each code is defined by three parameters: the alphabet size q, block length n, and message length k, with $k < n \leq q$. Every codeword in the set corresponds to function values from a polynomial of degree less than k.

- **Elliptic curves** These curves, defined over a field K, represent points in K^2. They are specifically categorized by the plane algebraic curve equation $y^2 = x^3 + ax + b$, where a and b are coefficients in K and the field's characteristic differs from 2 and 3.

- **Bilinear Pairing** A bilinear pairing is a mapping defined between two additive groups G_1, G_2 and a multiplicative group G_T, all of prime order p. Here, $g_1 \in G_1$ and $g_2 \in G_2$ are generators of G_1 and G_2 respectively. The pairing is represented as $e : G_1 \times G_2 \to G_T$ and has the following properties:
  - Bilinearity: For all $a, b \in \mathbb{Z}$, $e(ag_1, bg_2) = e(g_1, g_2)^{ab}$
  - Non-degeneracy: $e(g_1, g_2) \neq 1$

A.3 Proof-of-Storage Scheme

Let G_1 and G_2 be two elliptic curves with a bilinear pairing $e : G_1 \times G_2 \to G_T$. Let g_1 and g_2 be generators of G_1 and G_2. We will use a very useful shorthand notation

$$[x]_1 = xg_1 \in G_1 \text{ and } [x]_2 = xg_2 \in G_2$$

for any $x \in \mathbb{Z}$. Here we let G_1 be the elliptic curve BLS12-381 which is a pairing-friendly elliptic curve.

1. **Tursted Setup**

   The initialization of this protocol requires a one-time trusted setup. Once this is established, the protocol can repeatedly engage in committing to and revealing different polynomials.

   Assume we have a trusted setup, so that for a secret s, the elements $[s^i]_1$ and $[s^i]_2$ are publicly available for all $i = 0, \ldots, n-1$.

   Now, it's a fundamental aspect of elliptic curve cryptography that from the provided group elements in the trusted setup, it's computationally infeasible to extract the actual value of s. Despite s being a member of F_p, the prover is unable to determine its specific value. Instead, the prover is limited to performing certain operations with the given elements.

   For a given polynomial $p(X) = \sum_{i=0}^{n} p_i X^i$, the prover is able to compute

   $$[p(s)]_1 = [\sum_{i=0}^{n} p_i s^i]_1 = \sum_{i=0}^{n} p_i [s^i]_1$$

   This process enables the prover to compute the evaluation of the polynomial at the undisclosed point s within the group G_1, while maintaining the obscurity of s's exact value.

2. **Encoding** A user sends a file to a storage provider for storage. A file is represented by an ordered collection of one or more segments. Segments have a fixed maximum size for erasure encoding (in our case, it's Reed-Solomon Encoding). Each segment is treated as a vector of k elements in the finite field F, i.e. $x = (x_1, \cdots, x_k) \in F^k$. By using Polynomial Interpolation, the storage provider

calculates the unique polynomial p_x of degree less than k such that

$$p_x(i) = x_i \quad \text{for all } i \in \{1, \ldots, k\}$$

Once it has been generated, it is evaluated at the other points $k+1, \ldots, n$.

3. **Commitment** The storage provider commits to Reed-Solomon encoding polynomial p_x by calculating $C = [p_x(s)]_1$. This is called KZG commitment scheme or just Kate polynomial commitment scheme. It is called a commitment, because having sent the commitment value (an elliptic curve point) to the public, the prover cannot change the polynomial they are working with. They will only be able to provide valid proofs for one polynomial, and if they are trying to cheat, they will either fail to produce a proof or the proof will be rejected by the verifier.

4. **Verification** Every time when a user or other entities asks for verification at a random point z, the storage providers give the evaluation of $p_x(z) = y$. Besides, he calculates $q(X) = \frac{p_x(X)-y}{X-z}$ and the proof $\pi = [q(s)]_1$. He then send y and π to the verifier. The user can easily verify it by checking the following equation

$$e(\pi, [s-z]_2) = e(C - [y]_1, H).$$

If it holds, then the verifier accepts the proof.

A.4 File Retrieval

Retrieving a file from the Distributed Content Delivery Network (D-CDN) is a process that necessitates a well-coordinated interaction between storage providers and the user. The concept of 'fair exchange' is vital in this scenario. The impossibility results on fair exchange [16] establish that it is unfeasible for two parties to perform an exchange without involving trusted entities. The retrieval process consists of these significant steps:

1. **Bandwidth Calculation and Contract Signing:** Before initiating the retrieval process, storage providers estimate and communicate the total bandwidth and payment required for the file retrieval. Subsequently, the user selects a storage provider and signs a smart contract with them. The required amount is then escrowed by the smart contract.

2. **File Segmentation:** Due to the impossibility results on fair exchange without trusted parties, extensive files are fragmented into smaller, manageable segments. This step ensures fair transactions during the retrieval process.

3. **Segments Retrieval:** Segments are retrieved sequentially by the user. Every retrieved segment is verified by the user to ensure its integrity and authenticity. After successful retrieval and verification of a segment, the user signs a message confirming the receipt of the segment and sends it to the storage provider. Upon receiving the confirmation message, the storage provider transmits the next segment to the user.

4. **Retrieval Interruption:** In case the client stop paying, or the storage provider discontinues data transmission, either party has the right to halt the exchange. Subsequently, the storage provider uploads the confirmation messages of the transmitted segment to the smart contract. The smart contract then allocates the payment for the data confirmed by the user to the storage provider, and the remaining amount is refunded to the user. If needed, the user can sign another contract with a different storage provider to retrieve the remaining data.

Through these steps, the file retrieval process provides a balance of fairness and efficiency, offering users a reliable method to retrieve their files from the D-CDN.