MULTISYNQ

SYNCHRONIZE EVERYTHING

The Missing Protocol of the Internet

Multisynq Litepaper

April 2024

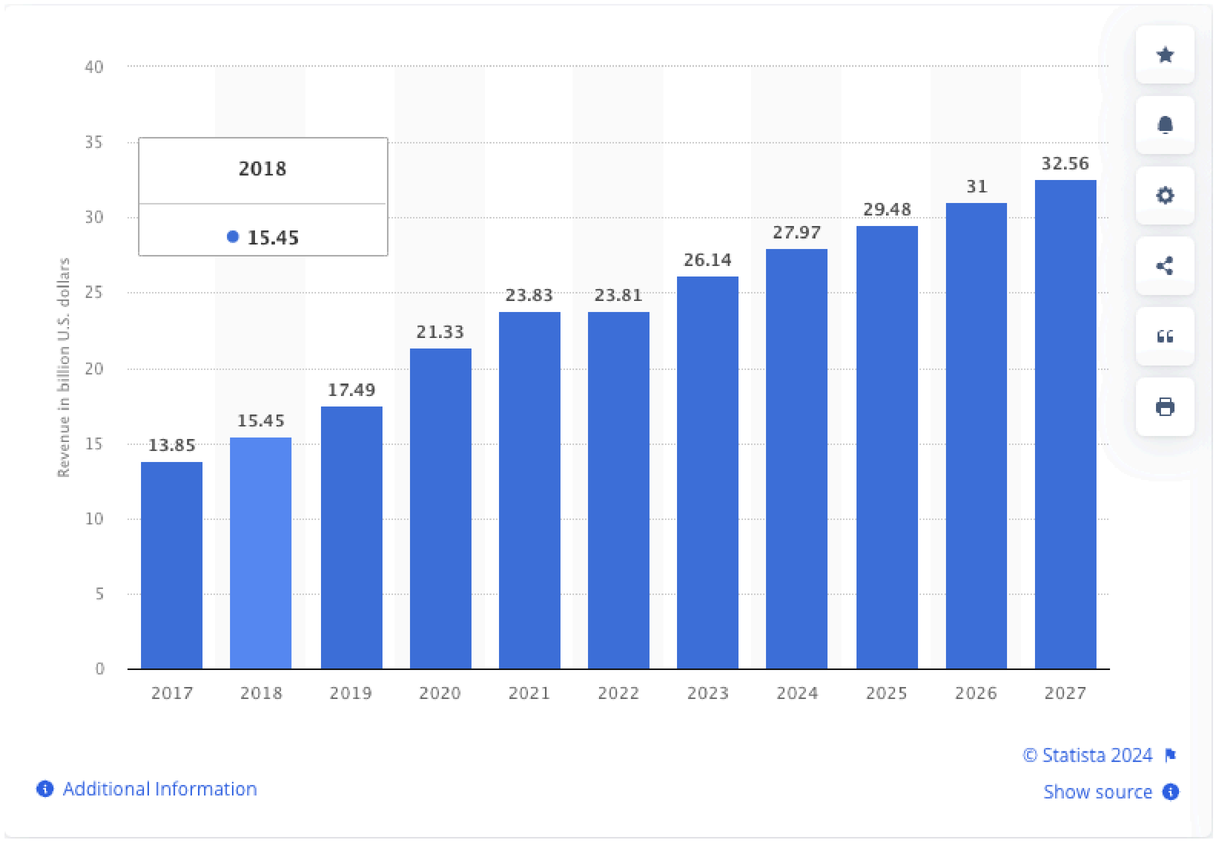
Version 1.3w

MULTISYNQ

Multisynq—It's About Time

The internet happens in real time. Every second of every day, hundreds of millions of people across the globe depend upon instantaneous connectivity to work and play together—from armies of hardcore gamers battling each other in online multiplayer games, to teams of doctors consulting in virtual reality to save lives, and legions of investors tracking and responding to fleeting changes in the market.

In 2023, the global online gaming market generated approximately 26.14 billion U.S. dollars in revenues, which translates to a 9.8 percent growth compared to the previous year. There are an estimated 1.1 billion online gamers worldwide with China, South Korea, and Japan having the biggest online gaming reach among the population.
https://www.statista.com/topics/1551/online-gaming/



https://www.statista.com/statistics/240987/global-online-games-revenue/

Growth in collaborative enterprise applications is also strong. It is expected to grow from USD 17.5 billion in 2021 to USD 40.79 billion in 2028 at a CAGR of 13.2% during the 2021-2028 period.[1]

With so many things on the internet depending on real-time interactions, it's strange that there's no general-purpose solution for real-time synchronization. Large multiplayer games run on bespoke servers. Virtual trading floors and medical collaboration apps use entirely different real-time networks.

All these multi-user applications suffer from lag spikes and glitches caused by lost connectivity. In addition, creating new ways to collaborate online is hamstrung by the need to reinvent the wheel whenever an entrepreneur wants to exploit a new niche.

That's where Multisynq comes in. The Multisynq Protocol is a robust general-purpose synchronization system for the Internet that relies on a decentralized fleet of DePIN Multisynq Synchronizers. The host of a Synchronizer is called a "Synqer",. a person (and device) whose Internet connected digital resources are used for application synchronization. A "Coder" is the developer of a Multisynq application. The Coder is responsible for providing access to their application and for payments enabling users to access and use the Synqer's Synchronizer. Rather than being forced to build a secure solution from scratch or suffer from connectivity issues, The Coder can simply use Multisynq's global infrastructure of Synchronizers to create their multi-user apps and games. That's why Multisynq has been called "the missing Internet protocol". It adds easy multi-user synchronization to the standard suite of frameworks available to developers.

And it does so in a way that is scalable. Built as a DePIN (Decentralized Physical Infrastructure) platform, the Multisynq network automatically adds capacity as more users need synchronization. There's a built-in financial incentive for Synqers, who host Multisynq Synchronizers, to add bandwidth resources to the pool. That means there's no upper bound on how many users can use Multisynq at the same time, no limitations because of bandwidth or latency, and no artificial constraint on the unlimited demand for synchronization.

The Multisynq Foundation is dedicated to fostering the success of the Multisynq Network around the world for developers and Synchronizer hosts and other members of the Multisynq ecosystem.

Croquet Labs is the software engineering and technology provider to the Multisynq Foundation.

[1] Fortune Business Insights: Team Collaboration Software Market Size, Share & COVID-19 Impact Analysis

How the Multisynq Protocol Works

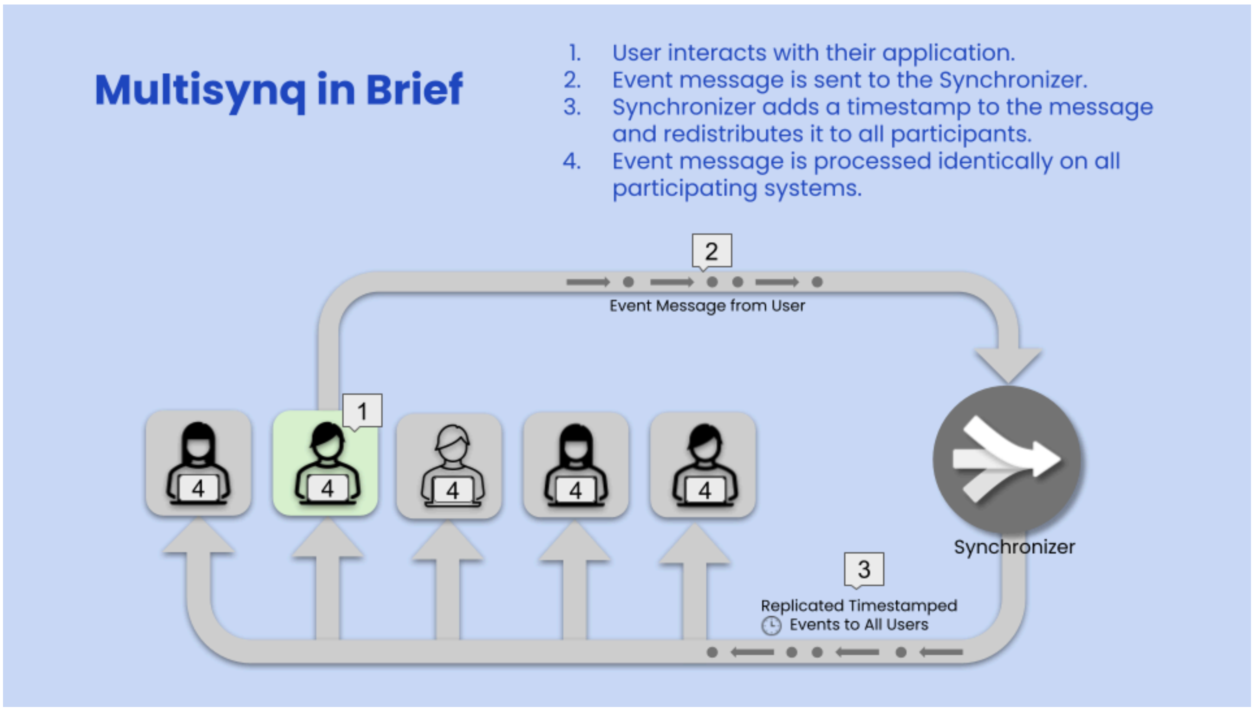
The core idea behind the Multisynq Protocol is decentralized deterministic computation.

A typical multi-user system has a powerful central server that performs all important calculations. It transmits the results of these calculations to its clients so they can stay in sync with each other.

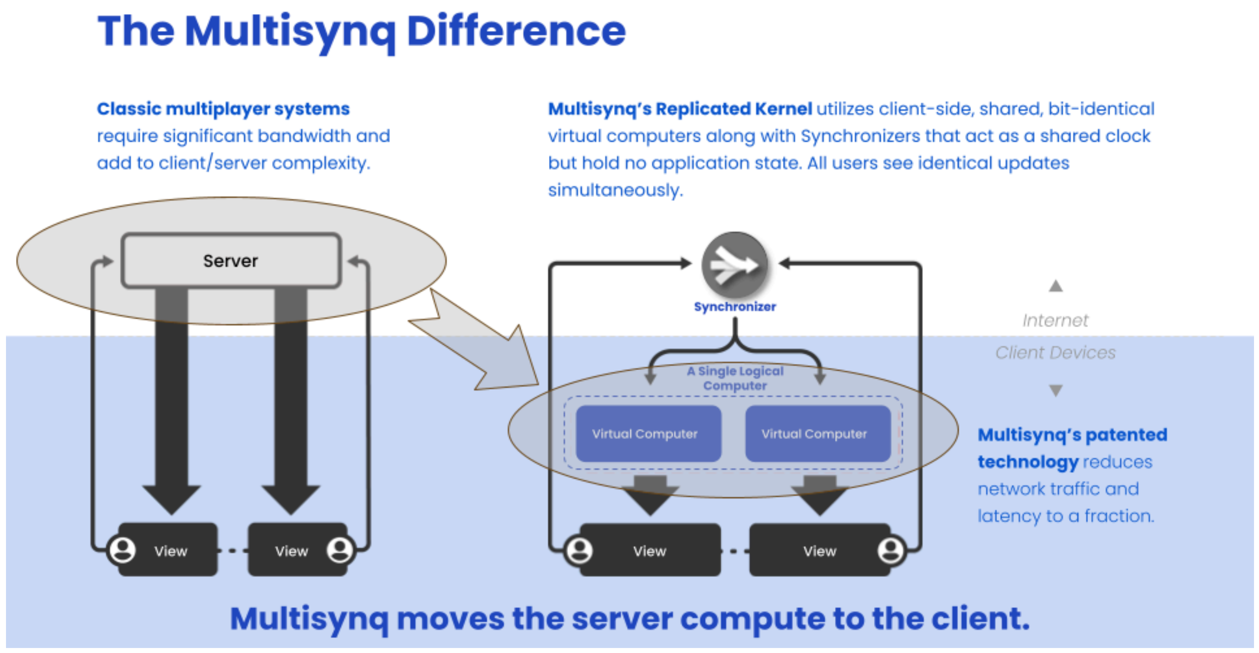
There are two problems with this approach:

- Central servers are complicated to program and expensive to maintain.
- Transmitting state info to the clients takes a lot of bandwidth.

Multisynq replaces the heavy central server with a worldwide fleet of lightweight, stateless Synchronizers.. No application computation actually occurs on the Synchronizer, nor is there a need to communicate with each other. The only role of a Synchronizer is to serve as a shared clock, supplying all the participating clients in a multi-user application with time stamped, replicated user events and synchronization ticks. All computation happens on the clients themselves. They stay in sync because Multisynq's built-in libraries guarantee bit-identical results for all computations on all platforms.


Multisynq in Brief
1. User interacts with their application.
2. Event message is sent to the Synchronizer.
3. Synchronizer adds a timestamp to the message and redistributes it to all participants.
4. Event message is processed identically on all participating systems.
2
Event Message from User
1
4
Synchronizer
3
Replicated Timestamped Events to All Users

The Multisynq Network will have hundreds of thousands (eventually millions) of Synchronizers deployed around the world, with the objective of delivering the same high quality experience in New York City as in a village in a developing country.


The Multisynq Difference
Classic multiplayer systems require significant bandwidth and add to client/server complexity.
Multisynq's Replicated Kernel utilizes client-side, shared, bit-identical virtual computers along with Synchronizers that act as a shared clock but hold no application state. All users see identical updates simultaneously.
Server
View
View
Synchronizer
A Single Logical Computer
Virtual Computer
Virtual Computer
View
View
Internet
Client Devices
Multisynq's patented technology reduces network traffic and latency to a fraction.
Multisynq moves the server compute to the client.

**Compute and state are decentralized to the shared virtual machine running on user devices while time is managed within the Synchronizer, which is selected based on geography of the session.*

Deterministic Computation

Multisynq is designed to ensure bit identical computation occurs within the same session on a variety of devices and operating systems. As an example, different operating systems often return slightly different values for transcendental functions like sin() or cos(). Even a tiny source of divergence like this will—over thousands or millions of calculations—gradually cause two systems to arrive at very different results.

Multisynq avoids this problem by patching all sources of divergence with its own deterministic code. It even has its own random(). Multisynq clients will always generate exactly the same stream of pseudorandom numbers, so even complicated Monte Carlo simulations stay perfectly in sync.

However, one potential source of divergence can't be fixed with deterministic libraries—user input. The whole point of a multi-user application is to let multiple people interact in real-time. Of course, every user will make different choices which affect the state of the shared simulation.

Multisynq deals with user input by passing it through the Synchronizer, interleaved with the synchronization ticks. Every client receives the same user input with the same timestamp so they all stay in sync.

Snapshots

When a new client joins a running Multisynq session, it needs to be brought up to speed with the current state of the system. Once it’s in sync with the other clients, it will stay in sync forever, but how does it get in sync in the first place?

The answer is the snapshot system. Periodically all Multisynq clients take a snapshot of their internal state and send it to the Synchronizer. This period depends on the complexity of the computation, where heavier workloads result in more frequent snapshots . When the snapshot occurs, every participating client performs a quick test to verify the replicated states are identical by computing and sharing the hash of that state, and one of the clients, typically the fastest, constructs and saves a snapshot to the cloud.

When a new client joins, the Synchronizer sends it the last snapshot along with any messages sent to the other clients since the snapshot was taken. The new client initializes itself with the snapshot, then quickly replays the messages it missed, bringing itself in sync.

Clients can join and leave Multisynq sessions freely without disrupting the flow of the simulation. The speed of individual clients is also immaterial. A slow client can be in the same session as a fast client without dragging it down.

If every client drops out, the simulation freezes at the time of the last message before a snapshot. A new client can join weeks or months later and take up the simulation at exactly the same point where it left off.

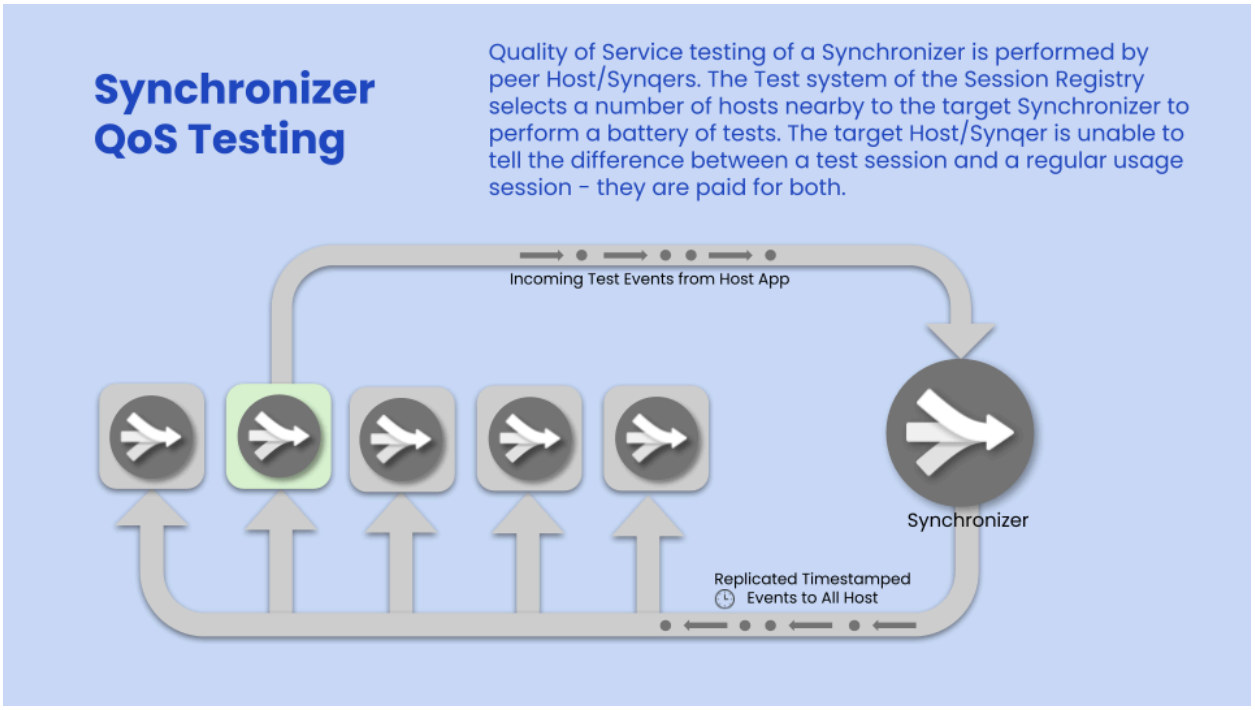
Session Registry

A key part of the Multisynq infrastructure is the Session Registry. The registry routes connecting clients to the Synchronizer handling their desired session. It also serves as a system-wide load-balancer—striving to keep active sessions spread evenly across the live Synchronizers—and a quality-of-service monitor—preferentially assigning clients to low-ping Synchronizers with stable connections.

Synchronizers are completely agnostic to the apps running on them. Multiple different apps can run on the same Synchronizer simultaneously, and an app that originally ran on one Synchronizer can dynamically migrate to another.

Quality of Service

The Multisynq network is designed to be self-testing to determine Synchronizer Quality of Service as a reputation score. Each host application, in addition to running a Synchronizer, will participate as a client in test sessions targeting other Synchronizers. All hosts are constantly participating in testing the network. The Session Registry provides this service, determining which Synchronizers need to be tested and which hosts will participate in that testing.


Synchronizer QoS Testing
Quality of Service testing of a Synchronizer is performed by peer Host/Synqers. The Test system of the Session Registry selects a number of hosts nearby to the target Synchronizer to perform a battery of tests. The target Host/Synqer is unable to tell the difference between a test session and a regular usage session - they are paid for both.
Incoming Test Events from Host App
Synchronizer
Replicated Timestamped Events to All Host

These tests are used to determine Synqer/host Quality of Service – they can also be used to determine the QoS of the tester – as all participants can "see" the messages from the other participants.

Test sessions are designed to reveal latency, geolocation, scale, and other parameters that make up the QoS vector. The Synchronizer being tested can't tell that the session it is hosting is actually a test – in fact, the host of the session is paid in tokens as if it is a normal session. Synchronizer hosts to participate in the session are chosen based upon location and their own quality of service value. Higher QoS means they are trusted more in computing the QoS of the target Synchronizer. Local hosts are chosen so that they can better test the geolocation of the target Synchronizer using a statistical triangulation algorithm based upon latency.

Security & Robustness

With the application code running directly on the clients it might seem at first like Multisynq would be vulnerable to a malicious user hacking the simulation state on his local client. However, the snapshot system protects against this. Any unauthorized change to a client's local state is caught when the Synchronizer will check to make sure all the snapshots match. A diverging client will be dropped from the session, eliminating the hacker from the multi-user simulation.

Another layer of security exists in how the Synchronizer handles messages. Because no computation happens on the Synchronizer, it never needs to read anything in the message body. All it looks at is the metadata in the message header. All internal messages and snapshots in Multisynq are end-to-end encrypted. The system passes them around as raw streams of bytes with no knowledge of what they contain. Even if a malicious hacker were to gain access to a Synchronizer, they couldn't steal or alter the message traffic.

Because any session can run on any Synchronizer, the system can quickly recover from a Synchronizer failure. If a Synchronizer goes down, the Session Manager will immediately direct its clients to another Synchronizer, which, if necessary will be able to rebuild the lost session from the previous snapshot, and the time stamped user event messages posted since that previous snapshot.

The system can also compensate for one client having a poor network connection. Dropped messages won't affect the other clients in the same session. The marginal client will experience lag, but the other clients will continue to run normally.

Why DePIN?

The unique features of Multisynq make it particularly well-suited to implementation on a Decentralized Physical Infrastructure Network. The Multisynq Network only manages the Session Registry, the Coder database, and the snapshot servers in the cloud today. These systems will also be fully decentralized in the future. The Synchronizers are run by "Synqers"—individuals or companies who add bandwidth resources to the network in exchange for $SYNQ which is a cryptocurrency that will be bought and sold in an online marketplace. When there's a high demand for synchronization services, the value of $SYNQ goes up, providing a strong incentive for Synqers to add more Synchronizers to the network.

Multisynq's security features protect users from unethical Synqers. There's no way for a Synqer to spy on the message traffic flowing through their Synchronizers. If a Synqer attaches a slow Synchronizer to the network, or one with an unstable internet connection, the Session Manager will route around it to faster, more stable Synchronizers. If a Synchronizer abruptly fails, little to no data is lost and another can quickly take its place. Furthermore, Synqers are dynamically replaced in a Session on a regular basis so it is difficult for the Synqer to even track traffic patterns in a Session.

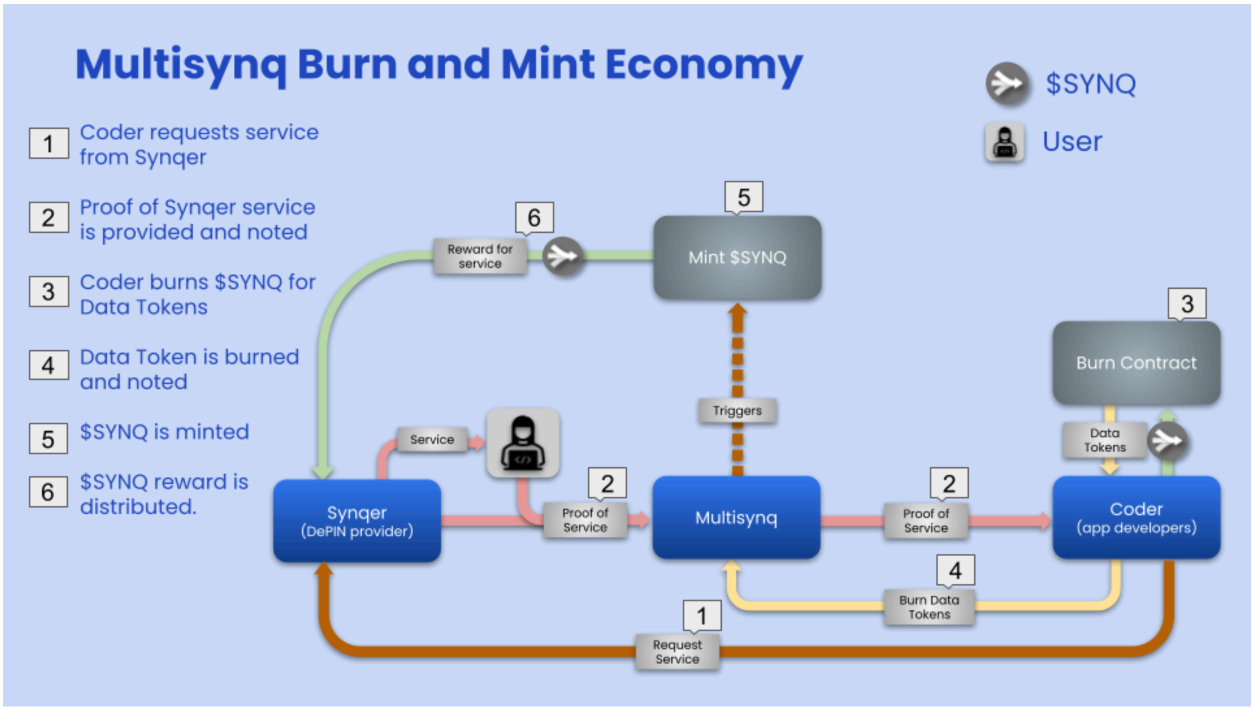
Burn & Mint

The Multisynq tokenomics model consists of a burn and mint mechanism. After burning a $SYNQ Token, the same (or similar) value is minted in the form of a Data Token. The key difference between those tokens is that burned $SYNQ Tokens are subject to price fluctuations, while minted Data Tokens serve as payment within the ecosystem and their ratio is fixed to the amount of services provided. This combination provides the opportunity to set a specific price for the service, which will be fixed once the $SYNQ Token is converted to a Data Token, regardless of the valuation of the $SYNQ Token.

Multisynq is focusing on a model that is commonly used in the market and is based on the rule that there is a constant max supply of Tokens in the system. The system mints the same number of (or possibly fewer) Tokens as are burned. Thus, the following model assumptions apply:

- The number of Minted Tokens equals the number of burned
- Max Total Supply is specified in the system
- The Synqer knows what reward they will receive in Tokens for the service it provides
- Fiat value used to purchase tokens by Coder should match fiat converted from tokens by Synqer (if we were to assume instantaneous transactions)

This model guarantees rewards for the Synqer while not diluting the number of Tokens. From the Coder/company's perspective the ratio of Data Tokens to data used (measured in bytes as well as other units of work that may be performed) is well defined, and from the Synqer's perspective the ratio of bytes (and consequently Data Tokens) to the number of $SYNQ Tokens they receive is also well defined.



By controlling the number of tokens in circulation and adjusting them according to actual use of the service, the value of the $SYNQ Token will reflect the actual demand and user engagement. A key aspect of the model is its ability to provide a stable, non-speculative increase in the value of the $SYNQ Token in proportion to the increasing usage of the service. A balanced ecosystem is promoted in which equilibrium is maintained, even when there are fluctuations in demand.

Existing DePIN projects that use similar models are:

- Helium – is a decentralized wireless platform that has a $HNT native token. By burning this token they generate Data Credit tokens, which are later used in payments on the blockchain. https://www.helium.com/
- Render – they use a similar connection in $RNDR tokens and "Render Credits." https://rendernetwork.com/

The Multisynq Tokenomics architecture is currently being designed and simulated.

Proof of Service

Service of the Synchronizer is defined simply by its providing the transfer of data to all participants within a Session. This data transfer is measured in bytes where the measured amount of data transferred results in a token reward to the Synqer hosting the Synchronizer. Multisynq utilizes a consensus mechanism to provide a proof of service by the Synchronizer to the client, and hence to the Coder's application. This is the foundation for how a payment is determined to the Synqer for the use of their Synchronizer.

There are a number of different kinds of messages from the Synchronizer of a Session to the participant. We focus on the three that represent by far the largest amount of data transferred.

- **User event messages** – these are the messages that are generated by user interactions – clicking, dragging, moving one's avatar in a space. These messages are replicated to all other participants via the Synchronizer.
- **Tick messages** – these are messages generated by the Synchronizer to move time forward when there are no user events available. This is used to enable the replicated simulations advance over time – this can include things like physics simulations and AI-based bots.
- **Snapshots** – these are generated at regular intervals and represent the instantaneous state of the shared client. The snapshot is computed by one of the participating clients, usually with the fastest compute, and uploaded directly to the snapshot database manager.

As a key part of that snapshot process, the data size of all the received user event messages is summed by all participants and the result is also added to the system state before the hash is computed. In addition, the Synchronizer itself sums up the size of the messages it sent to the participants during that snapshot period. This is the basis of part of the consensus proof for the service provided by the Synchronizer which in turn is the basis of part of the reward provided to the Synqer.

The tick events provided by the Synchronizer are used to move time forward in smaller increments than might occur with user event messages by themselves. This is how Multisynq is able to perform perfectly synchronized animations on multiple devices. Tick events do not need to be, and sometimes can't be identical in timing or size for each session client. In fact, each client may even have a different tick event rate. Instead, the Synchronizer and each individual client must agree on the tick rate to be provided at the start of the snapshot period as well as agree on the actual resulting tick events. This total amount of data from all clients is summed and is also the basis for part of the reward provided to the Synqer.

Finally, the snapshot is generated by one of the Session participants and uploaded to the Snapshot database manager for long term storage. The data size of the snapshot is also added to the total data transferred by the Synchronizer. In addition, an extra fee may be added to cover the cost of storage of the snapshot.

Roadmap

The core technology for Multisynq is already operational under the name Croquet and has been used for multi-user apps since 2020. The same tech team that built Croquet is migrating it to Web 3.0 right now—adding hooks to connect the Synchronizers to the Solana blockchain, and increasing security to allow third-party hosting Synqers to run Synchronizers.

Next steps are to build better integration frameworks for using Multisynq with existing technologies—the Unity and Unreal game engines for example, and the React UI library. The goal is to make adopting Multisynq as easy as possible for existing developers.

As Multisynq grows as a platform there will be a secondary market for new tools that are "Multisynq compliant." An example of this is the Rapier physics library which was written with Croquet in mind to be fully deterministic and provide a snapshot of its internal state. Croquet already supports multi-user physics simulations using Rapier, and there are many other opportunities to create similar packages for Multisynq—real-time fluid dynamics, finite element analysis, behavior trees for virtual agents and so on. Initially the Multisynq Foundation will "prime the pump" with these libraries, but as the platform becomes more ubiquitous, the opportunities for additional tools will explode.

Multisynq Project Phases

Phase 0: Initial Designs, Proof of Concept

This phase of the project began in September 2023 and included the preliminary designs of the decentralized Synchronization architecture, initial model of the Tokenomics, and the initial capital raise. Key assumptions were tested, particularly performance capabilities on various platforms and networks.

Phase 1: Kernel Development, Framework Development, Tokenomics Design

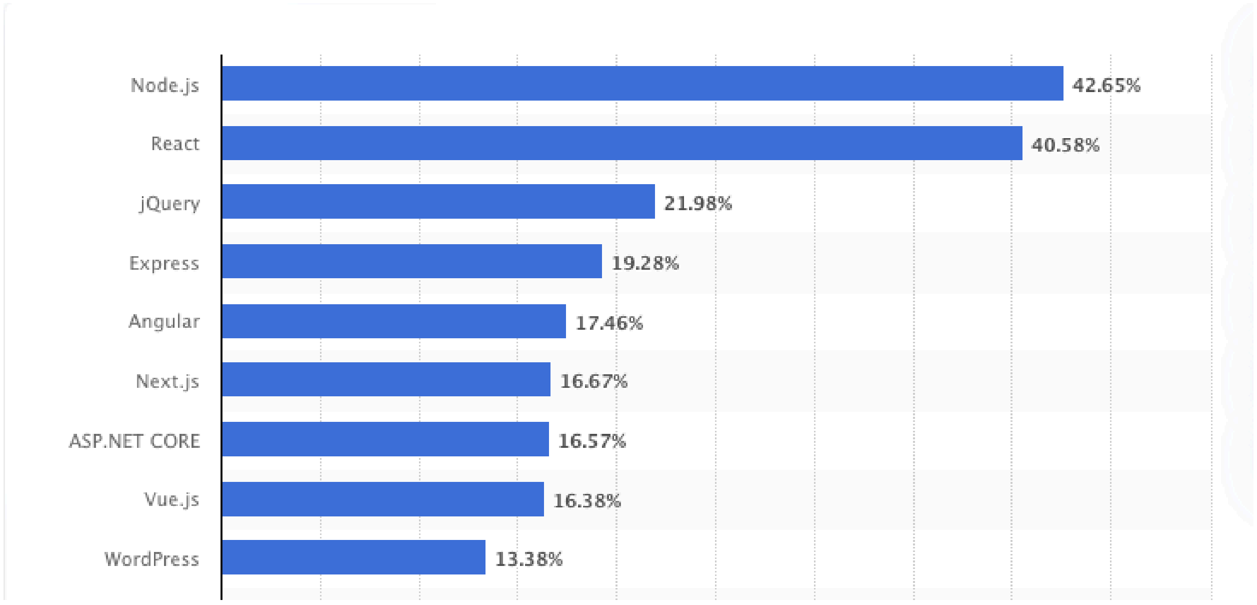
The initial kernel development phase began in January, 2024. This phase will be completed by early May, 2024.

Key elements of this system were designed and are being constructed including:

- **Synchronizer/Host application.** A deployable Multisynq Synchronizer kernel allowing the system to be deployed on traditional servers and as an Electron application for deployment on personal computer/network infrastructure. It can also be directly integrated into other DePIN infrastructure. Since the system is deployed as an Electron application, it will be easily upgraded by the hosts.
- **WebRTC communication.** WebRTC is designed to overcome the challenges of direct communication in the diverse and complex environment of the Internet. While the idea of a peer communicating with another peer is simple, things like NATs, the lack of IPv4 addresses, limited IPv6 support worldwide, different firewall and generic networking configurations, and lack of standardized firewall APIs tend to overcomplicate the process. WebRTC provides a solution to these issues.
- **Session and Synchronizer Registries.** Cloudflare-based registry databases of available Synchronizers and associated Sessions.
- **Session joining infrastructure.** Services for connecting users to Synchronizers hosting Multisynq Sessions.
- **STUN/TURN servers.** Aiding WebRTC connectivity to Synchronizers
- **Snapshot Storage mechanism.** Databases for storage and very fast access to Session Snapshot files.

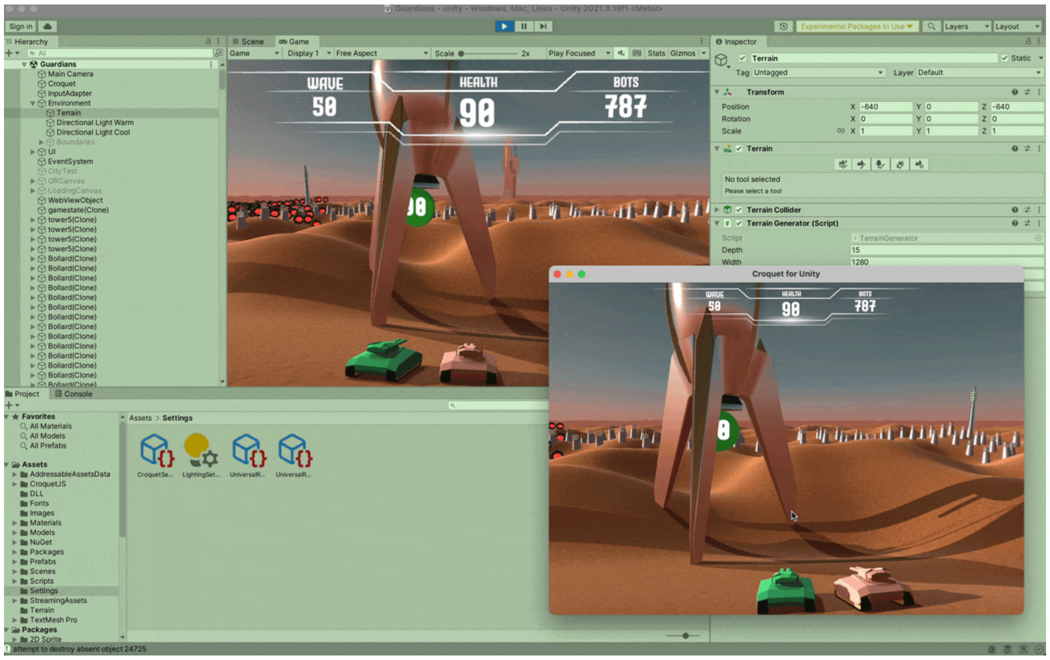
Along with the Kernel, a number of system frameworks to aid application development are being developed. These include:

- **The core JavaScript system.** The foundational framework provided as a library. All frameworks and Multisynq applications are built with this.
- **The Worldcore 3D engine framework.** This acts as a bridge between the JavaScript framework and various 3D engines like Three.js and Unity.
- **A React framework.** React is used by over 40% of all web developers. The new framework is designed around the workflow of React and the React developer.



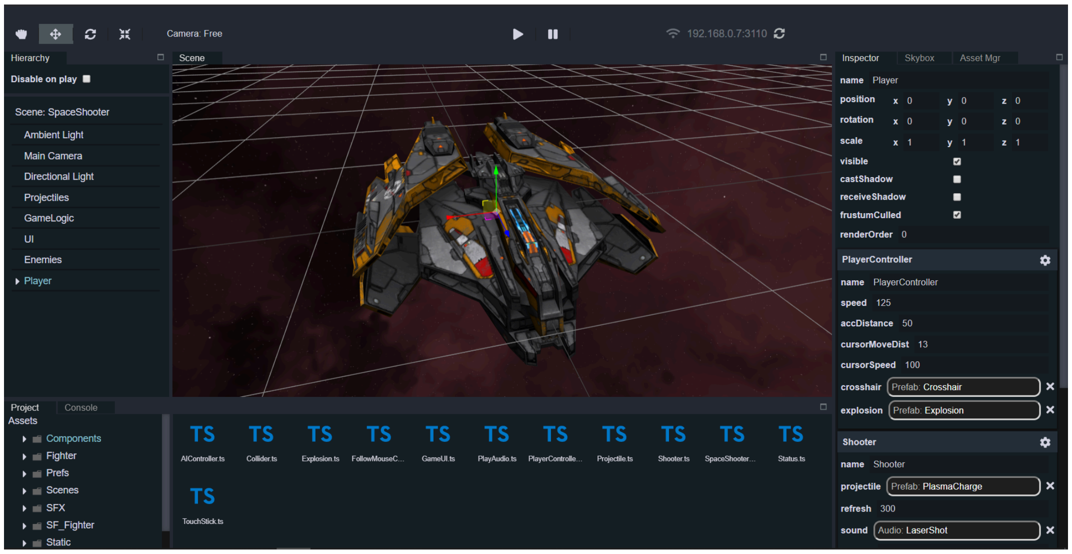
Source: Statista: Most used web frameworks among developers worldwide, as of 2023

- **A Unity game engine framework.** This includes a JavaScript/C# framework as well as a full C# component framework built on top.



The Multisynq Unity framework enables developers to create and test multi-user systems dynamically within the system while editing.

- **Other game engines**. Multisynq is being integrated into a number of additional 3D game engines – starting with the web-based Rogue Engine.



The Rogue Engine interface directly mimics the Unity platform, providing a similar development flow including management of components and simulations.

Multisynq Glossary

Introduction

This glossary is made up of terms used by the Multisynq Core System and the Multisynq DePIN Tokenomics. These are in separate sections for clarity. These systems define the Multisynq Network, a low-latency multi-user synchronization network that allows individuals to monetize their internet connections by selling surplus bandwidth to developers creating synchronized multiplayer apps, games, and virtual worlds. The network utilizes a solution based on a decentralized network of devices providing bandwidth and minimal computational tasks to users. The network consists of five main agents: Multisynq, Synqers/hosts, Coders/companies, end users and speculators.

Multisynq Core System

Bit-identical. The virtual computers synchronized by Multisynq are identical down to the last bit.

Coder. The developer of a Multisynq application. The Coder is responsible for providing access to their application and for payments enabling users to access and use the Synqer's Synchronizer.

Croquet. The original multi-user protocol that the Multisynq Protocol is based on. Originally developed by a team that included Alan Kay, a Turing Award recipient, David P Reed, who designed the UDP protocol, co-designed TCP/IP and formulated "Reed's Law", David A Smith, the creator of the first person adventure/shooter, the first real time 3D design tool, and was co-founder of Red Storm Entertainment with Tom Clancy, and Andreas Raab, one of the core developers of the Squeak Smalltalk system.

Croquet Labs. The software engineering and technology provider to the Multisynq Foundation.

DePIN (Decentralized Physical Infrastructure). Utilizes tokenized economic incentives to develop and maintain real-world infrastructure. Multisynq Synchronizers are decentralized – running on any machine with access to sufficient resources including bandwidth and minimal compute.

Determinism. A deterministic computation will always produce the same result from a given state. Multisynq relies on determinism to keep all participants in a Session bit-identical over time.

End-to-End Encryption. All event messages sent to the Synchronizer from the client are encrypted. The Synchronizer cannot read these messages, all it can do is add a timestamp and forward the message to all participating clients in a Session including the originator of the event message. Only these clients can decrypt the message.

Event Message. A user input event that is forwarded to the Synchronizer which adds a timestamp and broadcasts the event message to all the participants in a Session.

Event Message Queue. A queue of event messages sorted by timestamp kept by the Synchronizer. When a Snapshot is made, the queue is emptied and all subsequent messages are added. When a new user joins a Session, a Virtual Computer is initialized with a Snapshot, and the messages in the Event Message Queue are sent to the new Virtual Computer in order and executed. The new user is now perfectly synchronized.

Future Message Queue. Messages generated within a Virtual Computer as part of the synchronized simulation. These messages are sorted by their timestamp and keep the computation going purely based on the time as received by tick messages from the Synchronizer, even in the absence of event messages.

Hash. A fixed size byte code that is computed from the instantaneous state of a session running on a Multisynq Virtual Computer/client. All participating Virtual Computers compute this hash at the same virtual time which means that it is computed based upon the identical state of the Virtual Computer. This means that the Hash must also be identical on all systems. This is used to prove that the Virtual Computers have not diverged.

Latency. The time an event message takes for a round trip from the client participant in a session, to the Synchronizer and back to the client. Multisynq has the goal of providing between 15 and 30 millisecond latency for accessing local Synchronizers and the best latency physically possible everywhere else on the globe.

Model/View. Often referred to as Model/View/Controller. The basis of the Multisynq client architecture. The model refers to the replicated bit identical Virtual Computer. The view refers to the user interface which displays the state of the shared simulation, and gathers input from the user. The controller routes input events from the view as Event Messages to the hosting Synchronizer.

Multisynq. A multi-user-by-default protocol/platform for low-latency, realtime, interactive collaboration and shared simulation. It guarantees that all users of an application are observing bit-identical dynamic states at all times. This guarantee frees a developer from the bulk of the complexity involved in traditional multiplayer networking, while still allowing users to run on arbitrarily different devices.

Multisynq Foundation. Dedicated to fostering the success of the Multisynq Network around the world for developers and Synchronizer hosts and other members of the Multisynq ecosystem.

Multisynq Library. Client library used by Multisynq applications to provide the necessary interfaces and execution environment for the shared Virtual Computers.

Quality of Service. Or QoS – a metric determined by Synqer/Host peers testing the service provided by a Synchronizer. This will include metrics such as latency, bandwidth supported, number of simultaneous users, etc. Developers can set a minimum level of QoS to be provided by the Synchronizer. Requested higher levels of quality are more expensive. QoS is evaluated both through actual usage and dedicated testing sessions. During regular hosted sessions, metrics such as latency and message reliability are monitored, triggering further peer testing if disparities arise. Testing sessions are regular Multisynq Sessions where the participants are other Synqers/Hosts that perform live consensus tests including scale of users, scale of bandwidth, latency, and estimated geolocation. Synchronizers being tested cannot determine if they are running a test session or not, because payment is in $SYNQ Tokens and all messages are encrypted in the same way any Multisynq Session is. This testing is managed by the Session Registry.

Session. A shared world running bit-identically on a group of virtual computers kept in sync by a Synchronizer. End-to-end encryption protects its privacy. Sessions are regularly saved in the form of a Snapshot. A new user can join a Session by loading a Session Snapshot and playing back all messages received since that Snapshot was made.

Session Registry. A global registry of all Synchronizers and all Sessions. The Session Registry connects users who wish to join a Session to the hosting Synchronizer, and stores session metadata for later session resumption. If a Session requested by a user isn't currently hosted, the Session Registry chooses an available Synchronizer to be the host. All participants must pay for this service. Plans exist to fully decentralize the Session Registry, but this will also require payments for use.

Snapshot. The state of a replicated virtual machine is captured and saved as a file on a regular basis. In addition, a hash of this state is generated by all participants in a Session, the total amount of bandwidth delivered by a Synchronizer is computed, and subsequent event messages are also saved until the next Snapshot is generated. This Snapshot state enables new users to join a session in progress and also allows a session to be later reloaded and joined exactly in the state the last user left it in.

Synqer/Host. The host of a Multisynq Synchronizer. A person (and device) whose digital resources are used for application synchronization. Hosts typically run a single Synchronizer per machine that can in turn run multiple Sessions. The number of Sessions and number of users per session is determined by user settings and Quality of Service (QoS). If QoS decreases, the maximum number of supported sessions automatically decreases as well. Anyone can become a host, and for providing their digital resources, they receive compensation in the form of $SYNQ Tokens.

Synchronizer. Multisynq replaces the heavy central server with a worldwide fleet of lightweight, stateless Synchronizers running on a computer connected to the Internet. The Synchronizer receives event messages from a user, adds a timestamp to that message and redistributes it to all the participants in the session including the originator of the event message. It cannot read that message. Anyone can host a Synchronizer, and for providing their digital resources, they receive compensation in the form of $SYNQ Tokens.

Tick Message. Message automatically generated by the Synchronizer to move time forward in the Session to enable simulations to run continuously without user interactions.

Timestamp. The synchronizer adds a unique timestamp to each event message that it processes and forwards to participants in a session. This Timestamp acts as an external clock to the shared simulation of all session participants.

Virtual Computer. An isolated computation with strict rules of interaction. Any computation inside the Virtual Computer happens solely in response to the stream of ticks and event messages from the synchronizer. Since all computers in a session receive the exact same sequence of events, and the computation is deterministic, they are identical to the bit.

Multisynq DePIN Tokenomics

Bonus: Geological. A multiplier for the Quality of Service bonus. Its additional sign-up incentives are available for deploying Synchronizers in underserved regions. A global heat map indicates areas where Synchronizers are needed, typically areas with high demand or insufficient local supply. This map might be based on average latency to geolocated clients.

Burn and Mint. The Multisynq network utilizes a burn-and-mint equilibrium (BME) model. This model utilizes a two token system: proprietary data tokens and tradable $SYNQ tokens that attempt to accrue value. Coders, the providers of the Multisynq application, burn the "value-seeking" tokens in order to receive the "Data" tokens. There are two important caveats to the data tokens:
(1) data tokens can only be used to acquire services within the specific crypto network (similar to airline miles)
(2) the exchange rate of value-seeking tokens and Data tokens should be fixed and denominated in USD or some other external currency.

After burning value-seeking tokens to receive Data tokens, users then spend the Data tokens in order to use the network's services. This is a way for the users to display to the network that the service provider completed the work for the value-seeking tokens that were burned. These Data tokens are not transferable. The user isn't sending them to anyone, they're merely using them to acquire a service (again, similar to how a user spends airline miles to get on a flight).

Service providers then earn rewards by minting $SYNQ tokens which is independent of the token burning process. As the network grows in usage, the value-seeking tokens are burned and the tokens accrue more value. As they grow in value less will need to be burned to receive the same amount of USD denominated Data tokens. The reverse occurs if usage decreases, thus creating a dynamic to bring the network to an equilibrium.
https://messari.io/report/burn-and-mint-equilibrium

Payment Token. A utility token participating in the Burn and Mint mechanism. Based on the number of burned $SYNQ Tokens, a corresponding number of Payment Tokens are minted. These Payment Tokens are then transferred to companies. Companies burn Payment Tokens to purchase hosts services, and based on this, new $SYNQ Tokens are minted in the wallets of hosts.

Proofs. A number of proofs are required to verify the authenticity and capabilities of the Synqer/Host and the usage of the system to determine appropriate transfer of tokens.

Proof of Location and Latency. One of the key capabilities of the Multisynq platform is that there will always be geographically nearby Synqers to provide low-latency service to the user. It is insufficient for the Synqer to claim a geolocation, this will be verified as part of the QoS testing service and reported on with regular usage.

Proof of Service. Verification that the computation had been bit-identical across all synchronisers. A Multisynq session has a regular checkpoint called a Snapshot, where a consensus of bit identical state, and the actual amount of data provided by the Synqer to the participants in the Session is verified.

Proof of Synchronization. A cryptographic proof verifying the reliability, availability, scalability, geolocation and performance of the service

- reliability (of the service) — Synchronisers can guarantee a specific uptime with all requests processed atomically and with few failures.
- availability (of the service) — Synchronisers can always be reached, and always responsive. Not the same as reliability in that this can also include fail-fast processing.
- scalability (of the service) — Ability to cope with both a higher and lower demand for specific synchronisers by either improving the performance/bandwidth of individual synchronisers (vertical scalability) or offloading work to more synchronisers (horizontal scalability).
- performance (of the service) — An aggregate metric that encapsulates both improving reliability, latency of request processing, availability, etc.
- location of the Synchronizer — nearby Synchronizers with known locations triangulate on this Synchronizer to ensure the published geolocation reflects actual.

Proof of Quality of Service. A cryptographic proof of performance, availability and reliability of a particular synchroniser. Wraps the Proof of Synchronisation.

$SYNQ Token. A digital reward within Multisynq. It incentivizes hosts for making synchronizers available globally, developers for network contributions, and players for app adoption. Rising demand boosts token value. This token serves as a reward and motivation for hosts for their work, providing computational resource-sharing services. It is burned in the Burn and Mint mechanism. These tokens are purchased for dollars by the company from Multisynq or CEX. In the system, fees are paid in $SYNQ Tokens.

Testnet. A nearly identical replica of a cryptocurrency's full blockchain, designed specifically for testing purposes. It serves two primary objectives: to safely experiment with protocol changes and to allow external developers to integrate the cryptocurrency or protocol into their applications at no risk. Commercially, it functions as a crucial tool for ensuring the proper functionality of updates or new features before deployment on the main network. Through Testnets, developers can conduct thorough testing and validation to ensure the stability and reliability of the cryptocurrency's core functionalities. In the Testnet phase hosts for their engagement are rewarded with System Points.

LEGAL DISCLAIMER

Not An Offer. This Multisynq Foundation Litepaper is designed for general information purposes only and does not constitute a prospectus or financial service offering document and is not an offer to sell or solicitation of an offer to buy any security, investment products, regulated products or financial instruments in any jurisdiction.

Not A Contract. The information shared in this Litepaper is not all-encompassing or comprehensive and does not in any way intend to create or put into implicit effect any elements of a contractual relationship. The primary purpose of this Litepaper is to provide potential Multisynq enablers and token holders with pertinent information in order for them to thoroughly analyze the project and make an informed decision.

Utility Tokens. $SYNQ (Multisynq tokens) are pure utility tokens, meant to be exchanged for credits for usage of the fleet of Multisynq Synchronizers via the Synqer/Hosts of the Multisynq application. $SYNQ is not a corporate security and is not structured as such. Owners of $SYNQ tokens are not entitled to any rights in Multisynq or any of its affiliates, including any equity, shares, units, royalties to capital, profit, returns or income in the Multisynq Foundation or any other company or intellectual property associated with Multisynq.

No Representations Or Warranties. No representations or warranties have been made to the recipients of this Litepaper or its advisers as to the accuracy or completeness of the information, statements, opinions or matters (express or implied) arising out of, contained in or derived from this Litepaper or any omission from this document or of any other written or oral information or opinions provided now or in the future to any interested party or their advisers.

Note On Forward-looking Statements. This Litepaper contains certain forward-looking statements regarding the business we operate that are based on the belief of the Multisynq Foundation as well as certain assumptions made by and information available to Multisynq. Forward-looking statements, by their nature, are subject to significant risks and uncertainties. Forward-looking statements may involve estimates and assumptions and are subject to risks, uncertainties and other factors beyond our control and prediction. Accordingly, these factors could cause actual results or outcomes that differ materially from those expressed in the forward-looking statements. Any forward-looking statement speaks only as of the date of which such statement is made, we undertake no obligation to update any forward-looking statements to reflect events or circumstances after the date on which such statement is made or to reflect the occurrence of unanticipated events.

Trademarks. All companies and trademarks mentioned in this Litepaper are the property of their respective owners.

MULTISYNQ

SYNCHRONIZE EVERYTHING

The Missing Protocol of the Internet