dRTC: A decentralized Real Time Communication Network

Graphene 01, Inc

October 21, 2024

Abstract

The advent of open data and open source has proliferated technological innovations throughout earlier part of the century. While software has been able to rapidly decentralize, the physical systems that power it i.e *cyber infrastructure* - bandwidth, data, compute - are still majorly operated by a handful of proprietary centralised entities. Rapidly becoming a necessity of life, cyberinfrastructure is in acute need of an economic overhaul to suit its inelastic demand. Real-time data has a substantial array of applications, including powering real-time audio/video meetings and audio spaces, content delivery networks, hosting and connectivity in the metaverse, and large-scale and high throughput gaming applications. The decentralized real-time communication (dRTC) network democratises synchronous connectivity over cyberspace through a decentralized marketplace of real-time data. The role of the owner-supplier and consumer-user are interwoven, making the dRTC network a prosumer marketplace with decentralized governance. By democratising suppliership, rewarding suppliers commensurate with their quality of service, and decentralising governance, we build a strong incentive alignment between all interacting agents while cryptoeconomically ensuring all agents ultimately have a vested interest in the health of the network.

Note: The dRTC network is a work in progress. Active research is underway to improve and evolve the dRTC network. Updates will be posted to huddle01.com. For comments, recommendations, and collaborations, reach us at our dRTC Discourse Forum.

Contents

List of Figures

1 Introduction

The Huddle01 decentralized real-time communication (dRTC) network democratises synchronous connectivity over cyberspace.

Most consumer grade WebRTC[13] products in the market today are pressed with challenges around user privacy, scalability, and reliability. Cyber infrastructure and its products are owned by centralised propreitary entities whose incentives are at odds with users and the public. Such economic protocols make the technology prone to single points of failure and honeypot attacks, incentivize intermediaries to monetise user data and exercise a unilateral control price control over infrastructure resources. The dRTC network resolves this incentive misalignment through a decentralized and algorithmic prosumer marketplace of real-time data.

The dRTC network facilitates democratic node ownership, enabling anyone to operate and supply real-time data, participate in governance, earn rewards, and most crucially, own the means of production in the dRTC network. To improve the accessibility of node ownership, we have enabled delegation to third-party node operators, so that operators without sufficient resources independently may pool in order to participate in suppliership.

The decentralized architecture of the dRTC network distributes data streams across multiple nodes, enhancing performance by eliminating bottlenecks, while making the network secure and robust against single points of failure. Network cascading, intelligent resource allocation, and optimal data stream routing enable the dRTC network to offer ultra-low latency connections.

The dRTC network with decentralized ownership must also be designed and built in a decentralized fashion. Heeding to this, the dRTC network will be open-sourced such that eventually it is entirely democratic both as a software and as an infrastructure.

1.1 Elementary Concepts

These are the novel key components that have been conceptualized in building the dRTC network.

1. **decentralized Real-Time Communication (dRTC) Protocol** is an economic coordination protocol that defines policies, permissions, and interactions between agents on the dRTC network. This makes the network a marketplace for real-time data comprised of prosumers who can assume roles either on the supply side, as a Media Node to provision data and bandwidth, or on the demand side, as a Client to consume data through value-added applications powered by the dRTC network. Real-time data is the unit commodity in this market that allows for democratic ownership, suppliership, and governance.

2. **hyperorganic[12] systems design** The dRTC protocol is built with the *hyperorganic* systems design methodology which enables it to adapt its policies and parameters to changing internal and external market conditions. In hyperoganic[12] systems, policies and parameters are state-dependent, in contrast to static systems where parameters are set during initialization and remain fixed through the system's lifetime. State-dependency lends the system adaptability, and significantly reduces the surface of assumptions made about the system during its conception. *Dynamic mint and burn mechanisms* ensure that any growth in network value is captured and reflected in the token value. The protocol's hyperorganicity[12] enables it to simultaneously adapt whilst preserving the system's core functions.

3. **Proof of Resource** ensures fair compensation for Media Nodes and enforces performance standards. By tracking and rewarding nodes based on on-chain *Quality of Service* data, the system incentivises high up-time and optimal service quality, fostering a competitive environment where node operators are motivated to maintain consistent performance.

4. **Dynamic Allocation of Resources** the dRTC network enables intelligent allocation of resources by considering client's geo-location, operational history of the node and its quality of service, and a variety of network cascading strategies.

1.2 Protocol Overview

The dRTC protocol is a decentralized Real Time Communication Network operating an algorithmic prosumer marketplace for real-time data, powered by a blockchain and a native protocol token (called "HUDL").

Clients - representing the demand side - spend tokens to acquire dRTC data to power meetings, audio spaces, and other applications requiring real-time data. Media Nodes - representing the supply side - earn tokens by enabling low-latency real-time data to provision the Clients.

The dRTC Network utilises Proof of Resource to ensure fair compensation for Media Nodes and enforce performance standards by tracking and rewarding nodes based on on-chain Quality of Service data, incentivising high up-time and optimal service quality from node operators

The network functions as a multisided algorithmic marketplace:
Supply: Media Nodes providing network resources (bandwidth, compute).
Demand: Consumer and industrial applications built on the dRTC Network.

1.2.1 dRTC Protocol Sketch

Network

1. at request by registry (Smart Contracts):
(a) The Rewards Contract queries the Proof of
Resource Contract for node data
(b) Rewards for each Media Node are calculated
(c) The Rewards Contract requests token release
from the Reserve Contract
(d) Tokens are transferred to each Media Node's
Pool Contract
(e) The Pool Contract distributes rewards per
its rules

Media Node

1. Listen for requests from Orchestrator
2. Process RTC workloads
3. Perform QoS Tests on request from Registry Nodes

at each epoch t:

1. For each Client C, Report RTC Usage
2. Claim Rewards Earned for giving RTC Usage

Registry Nodes

at any time:
1. Perform QoS Tests on Media Nodes and collect analytics
2. Listen for Media Nodes entry requests to the network
3. Gossip New Media node entries to the network
4. Verify Media Nodes are Alive
5. Serve latest Media Node entries and metadata to clients

at each epoch t:

1. Update the reputation of Media Nodes based on
collected QoS analytics
2. Submit to Proof of Resource Contract

Orchestrator

at any time:
1. Request for Media Nodes from Registry:
(a) query Media Nodes from registry based on
input parameters
2. Allocate and Balance RTC Workload on Media Nodes

at each epoch t:
1. Data credits are Consumed for usage of RTC work
on the network

1.3 Protocol Diagram

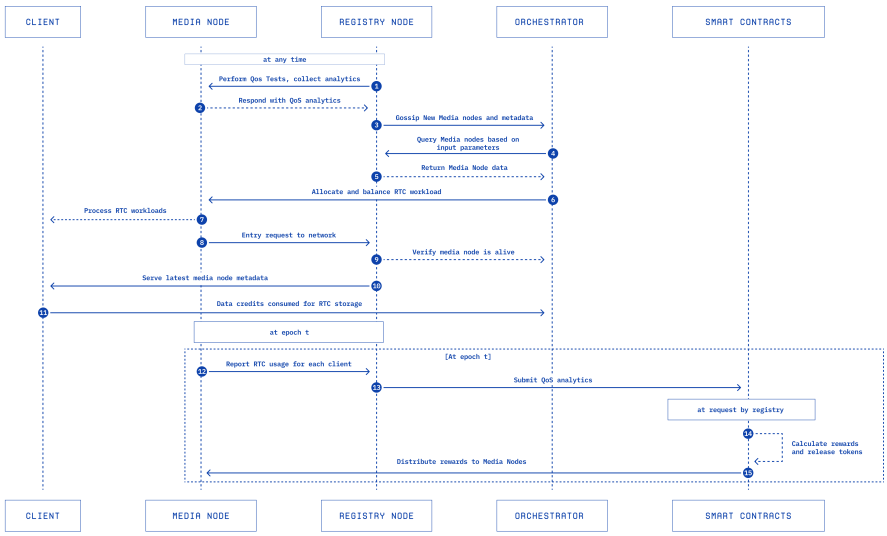


Figure 1: Protocol Diagram

2 The dRTC Protocol

2.1 Hyperorganic System Design Philosophy

Hyperorganic systems[12] implement coordination protocols where the policies are functions of the system state, instead of being prescriptive or predefined. The adaptability of a hyperorganic[12] system allows for a significant reduction in the number of assumptions that need to be made while designing the protocol. The economic dynamics of such systems are inherently free, dynamically adaptive, and emergent.

- Dynamically adaptive & perpetually relevant: Parameters and policies are not prescribed, rather they adapt as the system evolves. The protocol thus remains perpetually relevant to the current state, and to internal and external conditions.
- Free: Allow markets to operate freely whilst promoting healthy competition. With the capacity to self-update, its growth is not constrained by rigid or irrelevant predefined parameters, nor by the time to achieve consensus on protocol updates.
- Emergent: Enable positive macroscopic phenomena to arise from collective action.

This design philosophy ensures that the token value perpetually captures growth in the network's value, manages future risk[9], escapes local optima, and self-sustainably preserves the system's core functions.

Besides being hyperorganic[12], the dRTC network is robust and capable of withstanding long-tail events without compromising the system's fundamental functionality and integrity. We've built resilience into our system using fail-safes, mitigating single points of failure, multiple agent verification, and fault-tolerant architectures. dRTC protocol design adopts an iterative engineering design process of requirements definition, design, implementation, and verification & validation (V&V) [9].

2.1.1 Verification & Validation

V&V enables us to verify that the system meets our design specifications, and validate that the design specifications enable the system to achieve macro-level goals such as decentralisation, self-sustainability, and high performance.

We verify the system through analysis, audits, and rigorous network testing.

We validate the system via multi-agent simulations[2], modularised by subsystems each containing their sets of economic policies and parameters. Simpler validation methods such as Monte Carlo methods and parameter sweeps are applied to well-defined solution spaces.

2.2 The dRTC Network

The decentralized real-time communication (dRTC) network is a protocol to democratise synchronous connectivity over cyberspace.

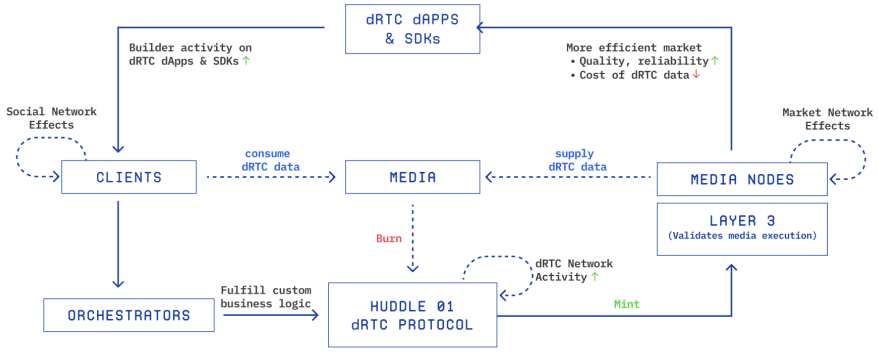


Figure 2: dRTC Network Flywheel

The dRTC network achieves this through a multi-sided prosumer marketplace of real-time data, providing anyone the ability to supply real-time data and earn rewards, or participate in secure, private, and competitively priced meetings, audio spaces, or other real-time connectivity apps via the dRTC network. The multi-sided marketplace comprises of Media Nodes who supply data and Users who consume data.

Orchestrators are the bridge between the Media Nodes and Users, acting as the brain and the control center for resource allocation decisions. Due to its modular nature, Users may choose to set up their own Orchestrators with custom resource allocation logic, or utilise the Orchestrator module outlined in Section 4.1: Orchestrator.

2.3 Media Nodes & Media Node Rewards

Media Nodes supply dRTC data so that Users can power their applications with meetings, audio spaces, and other synchronous data consuming products on the dRTC networks. They earn a reward for supplying real-time data on the dRTC network.

Media Node rewards are directly linked to their performance and contribution to the dRTC network. Factors influencing rewards include:

- Availability: The node's uptime and readiness to supply real-time data.
- Operational History: The node's track record of reliable service delivery.
- Location & Coverage: Proximity to areas with high demand or excess supply of dRTC resources.

- Data & Bandwidth Supply: The quantity of dRTC data provided and bandwidth made available.
- Quality of Service: Latency, latency fluctuations, and overall quality of the supplied bandwidth.

We define the following functional requirements in designing the Media Node reward function:

Functional requirement 1 The Media Node operator must be able to recoup their operation startup costs through their reward earnings within a certain period of time.

Functional requirement 2 The Media Node reward per unit data must be greater than the cost of regular operation of a Media Node per unit data.

Functional requirement 3 The Media Node reward function must satisfy Huddle01's margin condition, defined by the margin parameter μ over time.

Media Node unit reward We define unit reward r as the reward per dRTC bandwidth data supplied by a Media Node. The unit reward r is computed based on its corresponding reputational scores arising out of the pre-determined variables and the meeting scores arising out of post-determined variables.

The current pre-determined and post-determined variables are outlined below and the dRTC network may add more variables or redesign the existing ones to improve and evolve the dRTC network.

The unit reward for a Media Node is

$$r = \frac{r_{asym}}{\frac{1}{w_A A + w_H H + w_L L + w_c C} + e^{-k(V - \frac{r_{asym}}{2})}}$$

The following sections describe the definitions and derivations of the Media Node unit reward.

2.3.1 Pre-determined variables

These variables are aggregate normalized scores based on the history of operation of each Media Node up until the current state. Temporally, they possess a global scope, taking into consideration the node's overall operation rather than session-specific data. They can be alikened to objective reputation scores, as they are dependent on the Media Node's behaviour throughout its lifetime up until the current state.

Pre-determined variables	Symbol	Conditions
Availability score	A	$0 \leq A \leq 1$
Location score	L	$0 \leq L \leq 1$
Coverage score	C	$0 \leq C \leq 1$
History score	H	$0 \leq H \leq 1$
Quality of Service score	V	$0 \leq V \leq 1$

Availability Score This score describes the Media Node's up-time throughout its lifetime until the current state.

$$A = \frac{u}{l}$$

where u is the Media Node's up-time, and l is the Media Node's lifetime, i.e the total time that has elapsed since the Media Node commenced its operations

Location Score The location score describes the location of the Media Node with respect to participant demand. The closer a Media Node is to high participant density areas, the greater its location score.

The location score incentivises Media Nodes to operate in hotspots and in high-demand locales, ultimately increasing network availability of service and minimising latency.

Consider current participant density ρ_I at the locale at which a given Media Node is operating in -

$$L \propto \rho_I$$

Coverage score The coverage score describes the location of the Media Node with respect to the density of suppliers i.e Media Nodes in the vicinity. The further away a Media Node is to high Media Node density areas, the greater its coverage score.

We incorporate this score into the reward function to incentivise media nodes to operate in areas with sparse Media Node coverage, so as to improve the coverage of the Huddle01 network. This ties into our higher-level goal of providing thorough global coverage.

Consider current Media Node density ρ_M -

$$C \propto \frac{1}{\rho_M}$$

History Score This score describes the Media Node's proportion of service provided compared to the rest of the supply network, through its lifetime. In general, it can be considered an approximation of the Media Node's dedication to providing service for the network.

Arranging all Media Nodes in ascending order based on the amount of data supplied by them, we have a frequency distribution of Media Nodes with their respective data supplied.

The history score is the normalised percentile rank of the data supplied by the Media Node in the frequency distribution of data supplied by all Media Nodes in the network.

$$H_i = \frac{\sum_{j=0}^{i} n_j - (0.5 * s_i)}{N}$$

where s is the total amount of data supplied by a single media node throughout its lifetime, i is the Media Node under consideration, and $n_0 ... n_j$ is the count of all the nodes that have provided less or equal amount of data than node i

N is the total number of all Media Nodes in the network $N = \sum n$.

Crucially, the History Score impacts how new entrants are incentivised to join and continue to participate in the system.

To encourage new entrant participation, we can set the lower bound of the history score to be a positive value.

$$H_{min} = 0.5$$

Quality of Service Score The Quality of Service (QoS) score V describes the general quality of service supplied by a Media Node. Factors on which this score is approximated are the bitrate, the fraction of data lost, and the data packet round-trip time (RTT). The dRTC network employs a *challenger-prover model* to generate QoS scores where Media Nodes are randomly assigned the roles of *challenger* and *prover* to verify good service quality. This model is significantly more economically and resource efficient, and is robust against collusion attacks like game warping, compared to other verifier models in the industry.

$$V = F * \frac{B + L}{2}$$

where F is the fractional loss score, B is the bitrate score, and L is the latency score.

Refer to System Architecture: Registry to learn how these scores are computed.

2.3.2 Post-determined variables

These variables are determined after each meeting happens. Temporally, they possess a local scope as they are only dependent on the characteristics of the individual meeting.

Post-determined variables	Symbol	Conditions
Minutes	m	$m \in \mathbb{Q}^+$
Data supplied	b	$b \in \mathbb{Q}^+$

Minutes Minutes m describes the number of minutes supplied throughout the meeting.

Data Supplied The data supplied through the course of a session takes into consideration the minutes consumed by the served participants, as well as the bitrate that was required to serve each minute. Bitrate is the amount of data transmitted per second in a media stream.

Typically, audio, standard/HD video, and screensharing have different bitrate requirements, averages outlined below:

1. Audio only: 64 Kbps
2. HD video: 650 Kbps
3. Audio + HD video: 700 Kbps

The unit bitrate b_{unit} per minute for a meeting with n_p media-producing participants, and n_c media-consuming participants is

$$b_{unit} = b_{a,v} * n_p[(n_p - 1) + n_c]$$

where $b_{a,v}$ is the bitrate of an audio or video stream, expressed in per minutes.

The data supplied b is equal to the total utlised data for a meeting of m minutes.

$$b = b_{unit} * m$$

2.3.3 Media Node Unit Reward

We had defined the unit reward r as the reward per dRTC bandwidth unit of service provided.

$$r = f(A, L, C, H, V)$$

The unit reward for a Media Node is

$$r = \frac{r_{asym}}{\frac{1}{w_A A + w_H H + w_L L + w_c C} + e^{-k(V - \frac{r_{asym}}{2})}}$$

where k is the curvature of the sigmoidal S-curve and r_{asym} is the asymptote of the reward function.

The Media Node's general availability, history of operation, location and coverage scores are expressed by the Availability score A, History score H, and the Location score L, and Coverage score C respectively.

Based on the relative importance of each metric to the operation of the system, we assign weights to these scores. w_A, w_H, w_L and w_C scale the availability, history, location, and coverage scores such that

$$w_A + w_H + w_L + w_C = 1$$

The weights w_A, w_H, w_L and w_C will be initialized based on our learnings an expert judgements on the relative importance of each score to the healthy operation of dRTC network. After intialization, the weights will have an update process through governance and hyperorganic[12] updates similar to all other design parameters.

Maximum reward For a Media Node with ideal availability, history, location and coverage scores, $A = A_{max}$, $H = H_{max}$, $L = L_{max}$ and $C = C_{max}$. The maximum values these scores can take is 1, as $A, H, L, C \in [0, 1]$.

Thus, the maximum reward $r_{i,max}$ that a Media Node with ideal past operation metrics can earn is

$$r_{i,max} = \frac{r_{asym}}{1 + e^{-k(V - \frac{r_{asym}}{2})}}$$

2.3.4 Media Node Reward Function

The Media Node session reward R is the reward earned by a Media Node for hosting an entire session, where they serve b total utilised data to facilitate the session.

$$R = r * b$$

where r is the unit reward, defined by

$$r = \frac{r_{asym}}{\frac{1}{w_A A + w_H H + w_L L + w_c C} + e^{-k(V - \frac{r_{asym}}{2})}}$$

Thus, the total reward earned by a Media Node for each session it supplies is R.

3 Token Economics

The Huddle01 Network Token ("HUDL") is the native protocol token of the dRTC Network.

HUDL is the medium of exchange for all transactions on the dRTC network - supplying Media Node rewards, staking, and one of the avenues for customers and end users to make payments. Monetary policies within the Huddle01 token economic system govern the minting, burning, distribution, and allocation of tokens to builders, investors, and the community.

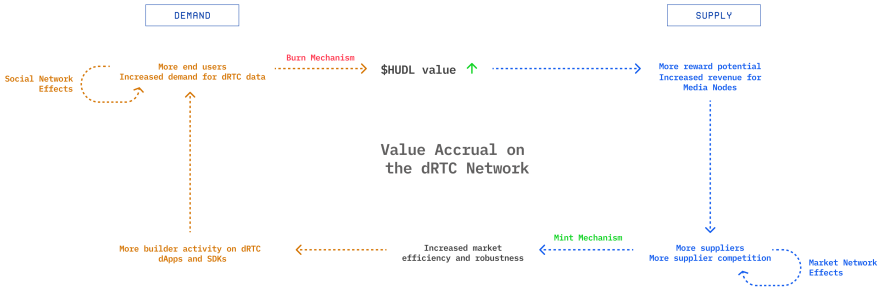


Figure 3: Value Accrual in dRTC Protocol

The system dynamics of the Huddle01 economy is inflationary. Although the economic system contains both inflationary and deflationary mechanisms built in, inflationary dynamics dominate thus making the overall system dynamically inflationary. The inflationary mechanism mints tokens to reward suppliers, adding to the circulating supply after each block. The deflationary mechanism burns tokens in proportion to the revenue generated from the dRTC network's usage.

The Huddle01 economic system utilises inflationary mechanisms to promote a high money velocity and incentivise supplier and Media Node participation. The inflation rate is set to strike a balance between providing sufficient rewards to suppliers while preventing excessive token value dilution. It also encourages token holders to maintain a healthy level of token circulation, ensuring sufficient velocity to support HUDL's utility as a medium of exchange on the dRTC network. The target annual inflation rate is generally between 2-15%.

3.1 Value Capture Mechanisms

Revenue is generated through the purchase and consumption of Data Credits by Users interacting with dApps or products built on the dRTC network. These interactions may involve direct use of the network or integration via the Huddle01 SDK. The Burn Mechanism captures incoming value from demand for real-time data. We are currently designing and developing several secondary value capture

mechanisms including staking, restaking, liquidity pooling, and a variety of primitives - both for the primary HUDL token, as well as incentivized testnet tokens.

3.2 Dynamic Minting

In the dRTC network, token minting incentivises supplier participation. Unlike static minting models which define a fixed minting schedule before launching the economy, the dynamic minting model is adaptive and state-dependent such that the amount of tokens to be minted is determined by the growth in the supply network's value. A dynamic minting model ensures that the token's circulating supply is perpetually commensurate with the actual value generated by supply-side network activities.

Media Nodes, acting as primary suppliers within the network, are rewarded with newly minted tokens in each epoch. The quantity of tokens received is directly proportional to the volume of data they contribute to facilitating meetings.

At the token generation event (TGE), denoted by $t = 0$, the system commences with an initial token supply of S_0. This initial supply precedes any subsequent minting activities such as those governed by the minting policy.

When a supplier earns a reward, new tokens are minted in accordance to the respective reward function, and disbursed to the supplier. The circulating supply increases by this amount.

For a Media Node, the reward function is

$$R = r * b$$

where b is the number of units of data supplied by the Media Node, and r is the unit reward for a single unit of data

$$r = \frac{r_{max}}{\frac{1}{w_A A + w_H H + w_V V} + e^{-k(v - v_{mid} - v_\tau)}}$$

The volume of tokens to be minted M in that block is

$$M = \alpha * \Sigma R$$

where α is the minting factor. It is a design parameter, and its value is currently set to 1.

The circulating supply S_C undergoes the state update function:

$$S_{C,t+1} = S_{C,t} + M$$

In alignment with the hyperorganic[12] systems design methodology, Huddle01 employs a dynamic minting mechanism to mint tokens in proportion to the

network's growth in value. When more data is supplied or more bandwidth and compute is onboarded to the dRTC network via increasing contributions from Media Nodes, this represents a growth in the supply side of the network. A growth in the supply side is always accompanied with a equivalent volume of minting.

3.3 Dynamic Burning

A burning policy is employed so as to capture the value generated due to dRTC network demand and product usage.

Mechanistically, it drives deflation of the HUDL token and increases its unit value in relation to the demand. Note that the inflationary dynamics still remain dominant, so the system overall remains inflationary.

End users engage with the dRTC network via apps, dApps, or products built on top of the network. These may be built and owned by any third-party developer (the dRTC builder). The dRTC builders are the ultimate prosumers of the dRTC network, both contributing value added services to the dRTC dApp ecosystem, while simultaneously deriving utility from them.

dRTC dApps may engage with the dRTC network in the following ways -

- Interface with the dRTC network directly, swapping out a centralized RTC provider for the dRTC network.
- Interact via the SDK by building an app utilizing the Huddle01 SDK. Any A/V communication on this app will be powered by the dRTC network.

We enable both cases while giving end-users and customers the freedom to engage with the dRTC network on a variety of terms, currencies, and models of their choice. To ensure this flexibility, we denominate all user/customer <-> dRTC interactions in terms of dRTC revenue generated.

The revenue generated from each customer <-> dRTC interface determines how many HUDL tokens will be burned in proportion. Revenue may be generated in a variety of modalities spanning currencies, frequency of payments etc. To ensure uniformity, we first denominate usage in terms of Data Credits.

3.3.1 Data Credits

Users must redeem Data Credits in order to consume real-time data on the dRTC network. Data Credits are pegged to $USD so as to ensure they are not subject to price volatility, thus keeping the price of data stable and predictable for the user. They are non-transferable, non-tradeable, and can only be used by their original owner.

The swap rate of Data Credit <> HUDL is determined using the spot price of HUDL <> USD at the time of burn. The burning is carried out as follows.

Given revenue generated per epoch p and current state circulating supply S_C, the amount of HUDL tokens to be burned in this block is

$$B = \beta * p$$

where β is the burning factor. It is a design parameter, and its value is determined based on parameter sweeps to ensure that the system maintains a desirable inflation rate. Based on our initial design considerations and given the early growth stage of the network, we have set $\beta = 0.6$. The remaining unburned amount is used to fund suppliers' and Media Node rewards and to fund further development of the protocol, including contributions to ecosystem growth programs. At all times, the burn rate is kept sufficiently low compared to the minting rate to keep the token inflationary, ensure the token's ability to act as a medium of exchange and encourage a high velocity of money.

For all design parameters, we employ multi-agent simulations and parameter sweeps alongside expert or collective judgement to determine β. In the future, we will also open up governance processes to enable the community to influence β.

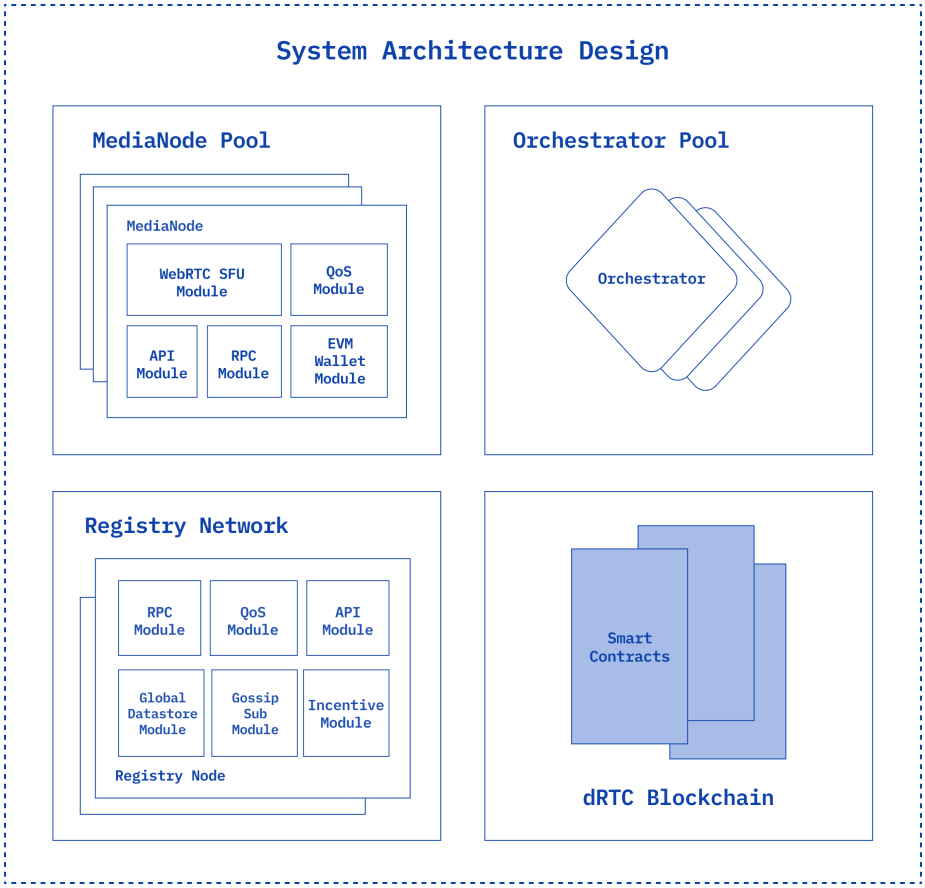
The circulating supply undergoes the following state update to remove the burned B amount of tokens out of circulation:

$$S_{C,t+1} = S_{C,t} - B$$

This model allows Huddle01 to keep the unit price of dRTC data stable with respect to USD or major stable currencies. On the other hand, HUDL tokens are subject to price movements due to market forces.

In alignment with the hyperorganic[12] systems design methodology, the dynamic burn model builds a strong link between the demand for data on the underlying dRTC network, ensuring that the HUDL token continually captures the value of the amount reflective of this demand.

4 System Architecture



This section presents a comprehensive technical overview of the system architecture for a decentralized real-time communication (dRTC) network. The architecture consists of four core components: Orchestrators, Registry, Media Nodes and dRTC Chain.

- **Orchestrators** act as the control center, optimising resource allocation and managing application-specific features to ensure efficient operation and routing, and can distribute workloads among themselves.
- **Registry** ensures system components can find each other and maintains observability across the distributed infrastructure.
- **Media Nodes** form the backbone of the dRTC network, processing, routing, and managing audio, video, and data streams.
- **dRTC Chain** functions as a Layer 3 blockchain, hosting the on-chain components of the dRTC Network. It handles key processes such as incentivization, consensus, and verification mechanisms

The following sections delve into the technical details of each component, their submodules, and the interactions between them that enable the dRTC network to function effectively and scale efficiently.

4.1 Orchestrator

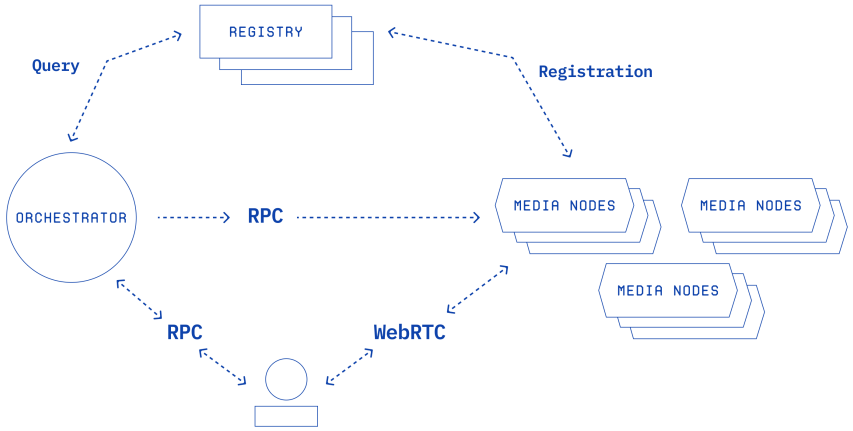


Figure 4: Overview of the Orchestrator and its Interactions

The Orchestrator component serves as the "brain" of the dRTC network, acting as a coordinator, managing, and controlling the flow of business logic and facilitating communication between Clients and Media Nodes.

Orchestrator contains the following features:

1. **Media Node RPC**: Managing the remote procedure call (RPC) interface between the Orchestrator and Media Nodes.
2. **Media Node Allocations**: Responsible for assigning Media Nodes to Clients based on various factors which ensures optimal resource allocation and load balancing across the network.
3. **Custom Application Features**: Gives the ability to developers to implement custom features and tailor the dRTC network to specific use cases.

4.1.1 Media Node RPC

The Media Node RPC Module facilitates communication between Orchestrators and Media Nodes through a standardised interface. This interface defines protocols, message formats, and supported operations, enabling Orchestrators to exert precise control over network resources via HTTP/2 RPC connections.

4.1.2 Connection Authentication

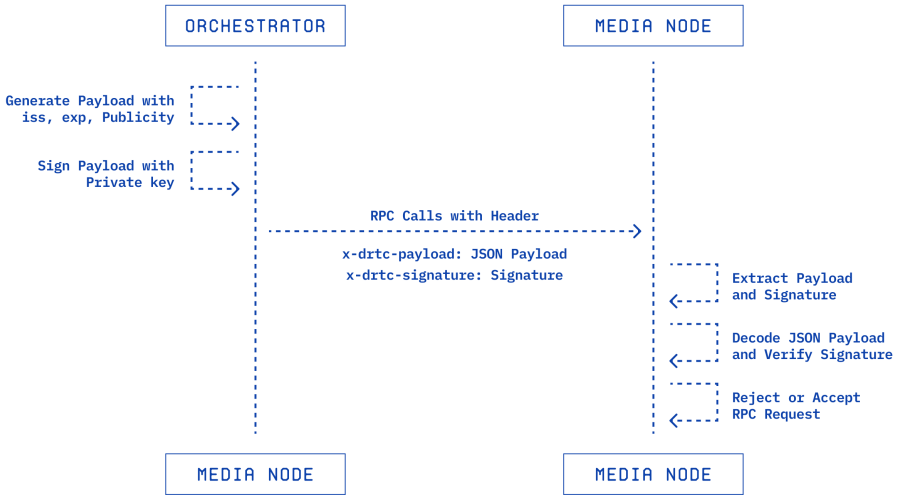


Figure 5: Media Node Connection Authentication

To ensure secure communication, each Orchestrator generates and maintains a Public-Private Key Pair.
For each RPC call to a Media Node, the Orchestrator creates a custom payload. This payload contains time-sensitive information and the Orchestrator's public key, serving as a secure, short-lived credential for the RPC call.

The payload is structured as follows:

```
iss := time.Now().UTC().Format(time.RFC3339)

exp := time.Now().Add(2 * time.Minute).UTC().Format(time.RFC3339)

payload := map[string]string{
    "iss": iss,
    "exp": exp,
    "publicKey": "<PUBLIC_KEY>"
}
```

This payload includes:

- `iss` (Issuer): The current timestamp in UTC
- `exp` (Expiration): A timestamp 2 minutes in the future
- `publicKey`: The Orchestrator's public key

Once created, the Orchestrator signs this payload using its private key. The payload and its signature are then transmitted to the Media Node as part of the RPC call using custom HTTP headers:

- `x-drtc-payload`: Contains the JSON-encoded payload
- `x-drtc-signature`: Contains the signature of the payload

Upon receiving an RPC call, the Media Node performs the following steps to authenticate the request:

Algorithm 1 Media Node RPC Call Authentication

procedure AUTHENTICATERPCCALL(*request*)
  payload ← ExtractHeader(*request*, 'x-drtc-payload')
  signature ← ExtractHeader(*request*, 'x-drtc-signature')
  $\{publicKey, iss, exp\}$ ← DecodePayload(*payload*)
  $currentTime$ ← GetCurrentTime()
  if $currentTime < iss$ **or** $currentTime > exp$ **then**
    return FALSE
  end if
  $isValid$ ← VerifySignature(*payload*, *signature*, $publicKey$)
  if $isValid$ **then**
    return TRUE
  else
    return FALSE
  end if
end procedure

If all these steps are successful, the Media Node considers the RPC call authenticated and processes the request.

4.1.3 Media Node Allocations

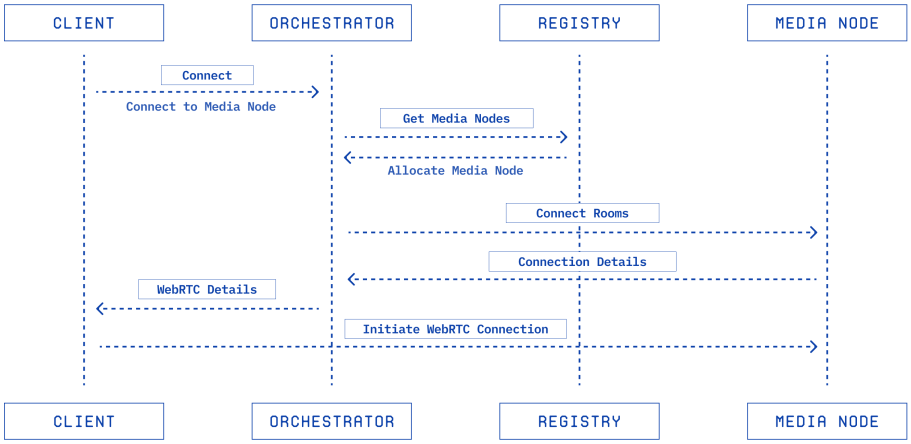


Figure 6: Allocation Schematic Diagram

The Media Node Allocations module is responsible for assigning Media Nodes to Clients and managing connections between Media Nodes in case of cascading. For detailed information on cascading, refer to Cascading Section in the section of Media Nodes.

When a client requests RTC services, the allocation process follows these steps:

1. The Orchestrator queries the Registry for available Media Nodes and their associated details.
2. Using information such as geographical location and bandwidth, the Orchestrator decides which Media Node is allocated to the client.

4.1.4 Custom Application Features

Clients may develop their own implementation of an Orchestrator with custom business logic. The Orchestrator offers a versatile framework for developing custom features and implementations which allows them to use the dRTC network to specific use-cases. By controlling Media Nodes and managing application-specific logic, developers can create optimised solutions for various scenarios, such as:

- **Host Assignment in Video Meetings:** In a video conferencing application, the Orchestrator can intelligently assign the role of "host" to a specific participant. This host can be granted special privileges, such as the ability to mute other participants, control screen sharing, or manage the overall meeting. The Orchestrator can use custom logic to determine the host based on factors like the order of joining, user roles, or explicit selection.

- **Content Delivery Network (CDN) Optimisation:** The Orchestrator can optimise the dRTC network for content delivery by strategically allocating Media Nodes[16] and controlling media stream flow. It considers factors like viewer geographic distribution, network conditions, and content popularity to make intelligent routing and distribution decisions, ensuring optimal performance and scalability.
- **Audio-Only Transmission:** For audio-focused scenarios like audio spaces, voice calls, or podcasting, the Orchestrator can prioritise audio transmission by allocating Media Nodes specifically for audio processing and adapting the media pipeline to minimise latency and ensure high-quality delivery. It can also reduce bandwidth usage by disabling video transmission when not required.
- **Adaptive Cascading Strategies:** The Orchestrator can dynamically select and apply different cascading[**?**] strategies (e.g., Proximity-Based, Regional, or Fanout) based on application requirements and network conditions. By adapting the cascading[**?**] strategy on the fly, the Orchestrator optimises factors like latency, load balancing, and content distribution, ensuring efficient resource utilisation and optimal performance for each use case.

  Note: For detailed information Cascading[16] Strategies, refer to Cascading[**?**] section of Media Node.

The dRTC Network's RPC driven architecture enable seamless integration with existing different application logic, allowing Clients and developers to define custom behaviours and policies. By leveraging the Orchestrator's capabilities, applications can use the dRTC network and deliver tailored, high-performance experiences to their users.

4.2 Registry

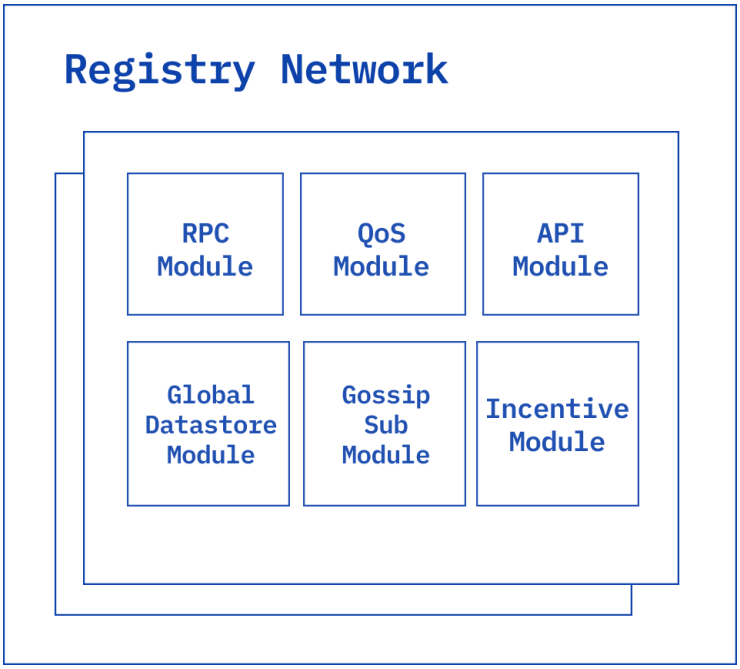


Figure 7: Schematic overview of the Registry Modules

The Registry is the entry point for Media Nodes and Orchestrators in the dRTC Network. It ensures system components can find each other and maintain observability across a distributed infrastructure. It serves as a data store and conducts node management, data storage, and network operations.

4.2.1 PubSub Module

The PubSub module for the Registry Network uses a gossip[19][11][**?**] protocol to propagate messages and is responsible for two primary functions:

1. **Bootstrapping new registry nodes:** When a new registry node joins the network, the PubSub[19][11][**?**] Module facilitates its integration by connecting it to existing nodes and sharing essential network information
2. **Facilitating communication between registry nodes**: ensuring that all nodes have up-to-date information about the network's state which is important for maintaining an accurate and current list of Media Nodes.

PubSub Module allows seamless scalability as new nodes join and helps maintain network consistency by propagating updates efficiently across all registry nodes.

4.2.2 DataStore Module

The DataStore Module stores information about Media Nodes and their attributes. It uses Conflict-Free Replicated Data Types (CRDTs) to ensure data consistency across the network and uses distributed data storage across the network, enhancing data backup, availability, and resilience.

Data Consistency and Storage

- **CRDTs for Consistency:** CRDTs prevent conflicts during concurrent updates, maintaining data integrity in the distributed environment.
- **High-Performance Storage:** A key-value store optimized for fast operations.
  - **Key-Value Structure:** Efficient data organisation and access.
  - **Index-Based Queries:** Speeds up data retrieval.
  - **Optimised for Registry:** Supports rapid lookups of Media Node [information].

4.2.3 RPC Module

The RPC Module executes two functionalities:

1. **Media Node Registration:** It processes requests from new Media Nodes seeking to join or leave the network.
2. **Node Discovery Interface:** It provides an interface for Orchestrators to discover and fetch the list of registered Media Nodes.

4.2.3.1 Media Node Registration

When a new Media Node joins the network, it registers with the Registry using gRPC. The registration process works as follows:

Algorithm 2 Media Node Registration Process

```
1:  procedure RegisterMediaNode(mediaNode)
2:      if connection is successful then
3:          nodeData ← PrepareNodeData(mediaNode)
4:          response ← InvokeRegisterEndpoint(connection, nodeData)
5:          if response is successful then
6:              GossipPropagation(nodeData)
7:              return SUCCESS
8:          else
9:              return REGISTRATION_FAILED
10:         end if
11:     else
12:         return CONNECTION_FAILED
13:     end if
14: end procedure
```

If a Media Node needs to leave the network, it can use the same `deregister` endpoint to deregister itself.

The data submitted during registration is encapsulated in the `NodePayload` struct:

```
type NodePayload struct {
    NodeId       string `json:"nodeId,omitempty"`
    Region       string `json:"region,omitempty"`
    Geolocation  string `json:"geolocation,omitempty"`
    Url          string `json:"url,omitempty"`
    DiscoveryUrl string `json:"discoveryUrl,omitempty"`
    Ip           string `json:"ip,omitempty"`
    MetricsUrl   string `json:"metricsUrl,omitempty"`
    PubKey       string `json:"pubKey,omitempty"`
    Version      string `json:"version,omitempty"`
}
```

The `NodePayload` struct contains the following fields:

- `NodeId`: A unique identifier for the Node.
- `Region`: The geographical region where the Media Node is located. `( NAM, ASI, EU )`
- `Geolocation`: The specific geolocation of the Media Node.
- `Url`: The URL of the Media Node.
- `DiscoveryUrl`: The URL used by Registry to communicate with the Node.
- `Ip`: The IP address of the Media Node.
- `MetricsUrl`: The URL where metrics related to the Media Node can be accessed.
- `PubKey`: The public key of the Node.
- `Version`: The version of the Node.

4.2.3.2 End-User Interface for Node Discovery

The Registry is a central component in our media management system. It provides a REST-based interface that allows various entities - including end-users, client applications, and other network components - to query and retrieve information about Media Nodes. This interface includes endpoints such as:

- `/getMediaNodes`: Retrieves a list of available Media Nodes based on specified criteria.
- `/nodeById`: Fetches information about a specific Media Node using its unique identifier.
- `/listMediaNodes`: Returns a list of all available Media Nodes in the network.

These endpoints can be accessed using standard HTTP GET requests, allowing easy integration with various client applications and services.

4.2.4 QoS Module

The QoS Module monitors the Quality of Service (QoS) provided by Media Nodes and perceived by end users.

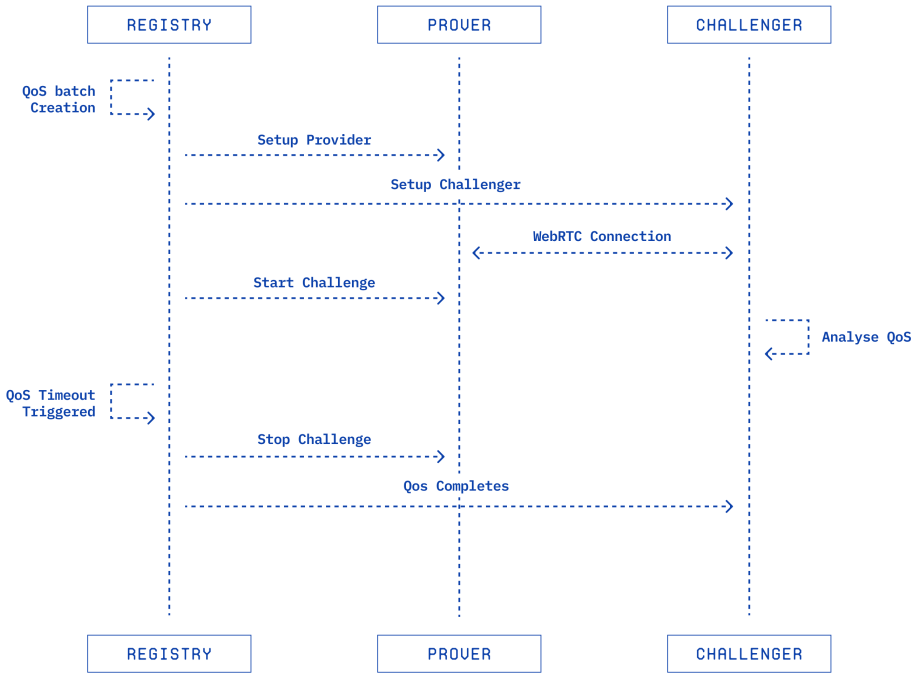


Figure 8: QoS Schematic Diagram

QoS Metrics The following metrics are used to approximate the level of QoS:

- **Bitrate** is the amount of data transmitted per second in a media stream.
- **Fraction Lost** is the proportion of data packets lost in a network transmission.
- **RTT (Round-Trip Time)** is the time it takes for a data packet to travel from the source to the destination and back.

These metrics are used to approximate media streaming quality. Bitrate affects media clarity, Fraction Lost indicates connection stability, and RTT (Round-Trip Time) measures latency, impacting real-time communication (RTC) responsiveness. Together, they can provide a high-level overview of Media Node performance.

QoS Tests

The Registry initiates QoS tests on Media Nodes to validate bandwidth requirements and assess the network condition between them.

The QoS test process designates one Media Node as the `Challenger` and the other as the `Prover`. The `Challenger` initiates the test by connecting with the `Prover` and assessing QoS metrics. The Prover's role is to demonstrate its ability to maintain good service quality.

The selection of challenger and prover nodes for QoS tests follows a randomised process based following algorithm:

- **Randomness**: The selection is primarily random, ensuring fairness and unpredictability.
- **Bias for Untested Nodes**: The algorithm prioritises nodes that haven't been tested recently to ensure full network coverage.
- **Repeat Testing**: Any node can be selected at any time, maintaining unpredictability.
- **Adaptability**: The process adjusts to the network's size and conditions as they change.
- **Role Rotation**: Nodes switch between challenger and prover roles in different tests to provide a broader performance evaluation.

Each QoS test runs for a duration of 20 minutes, with a 2-minute setup phase. After the test, the Challenger reports the measured QoS metrics to the Registry.

By regularly conducting these tests, the QoS Module can identify potential issues and take corrective actions to optimise network performance and ensure a high-quality user experience.

QoS Scores After a QoS test is completed, the Registry requests the Challenger to provide the following QoS Scores:

- **Bitrate Score (B)**
- **Fraction Lost Score (F)**
- **RTT Score (R)**

In addition to the final scores for each metric, the Challenger also sends the raw data collected during the test using which the scores were determined. This raw data can be used for further analysis or to verify the accuracy of the calculated metrics.

Note: For detailed information on how the Bitrate (B), Fraction Lost (F), and RTT (R) metrics are calculated, Refer to QoS Module section of Media Node.

Final QoS Score Calculation

The Quality of Service (QoS) score for a Media Node is calculated using a formula that combines three critical performance metrics:

$$V = F * \frac{B + L}{2}$$

Where F is the Fractional Loss Score, B is the Bitrate Score, and L is the Latency Score.

Each component is positioned to reflect its impact on real-time communication quality, based on the rationale:

1. Fractional Loss (F) serves as a multiplier, reflecting its importance in maintaining media quality. Packet losses can significantly degrade audio and video quality, causing noticeable interruptions. By using F as a multiplier, we ensure that any substantial packet loss heavily penalises the overall QoS score, regardless of performance in other metrics.

2. Bitrate (B) and Latency (L) scores are averaged in the formula, because of their interdependent nature in real-world scenarios. By averaging these scores, we create a balanced representation that prevents overemphasis on raw throughput at the expense of responsiveness, or vice versa.

   Interpretation of the (B + L) / 2 component:

   - Scores > 0.8: Indicate performance suitable for most applications
   - Scores < 0.6: Suggest potential issues in either bitrate or latency, warranting further investigation

4.3 Media Nodes

Media Nodes route and manage audio, video, and data streams. These nodes are the core components of the dRTC Network. They handle encoding, decoding, mixing, and routing, providing essential CPU and bandwidth resources.

Every Media Node contains the following modules:

- **Media Engine**: Runs the Selective Forwarding Unit (SFU)[6], which manages WebRTC[13] connections and routes media packets. The next section will cover the media engine in detail.
- **RPC Module**: Handles communication between the Media Node and the Orchestrator. Media Nodes act as workers, following instructions from the Orchestrator.
- **Registry Module**: Manages communication with the Registry Network. Media Nodes advertise their available resources and capabilities to the network when ready to participate.
- **QoS Module**: Oversees the Quality of Service, ensuring the Media Node delivers resources as advertised to the network.

A single Media Node may encounter bandwidth and compute limitations as the number of meeting participants increases. To resolve this, Orchestrators can distribute workloads across Media Nodes, effectively reducing CPU and bandwidth requirements on one node, scaling workload across multiple nodes. This process is referred to as Cascading[**?**].

4.3.1 Cascading

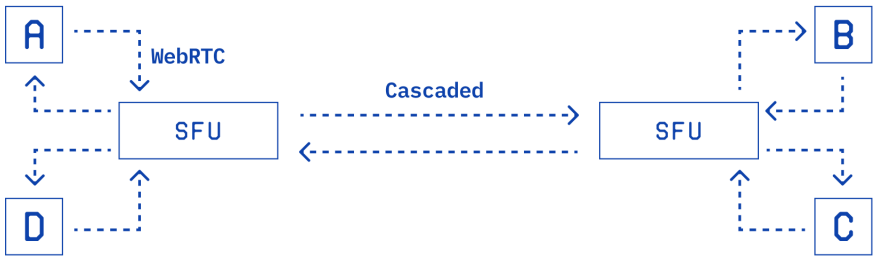


Figure 9: Two SFUs[6] cascaded for four clients (A,B,C,D)

Figure 9 shows Media Nodes receive media streams from one WebRTC[13] peer and forward them to others, requiring significant bandwidth. As the number of participants grows, a single Media Node can not handle the load, presenting a scalability challenge. To manage this, Media Nodes use cascading[**?**], similar to database sharding to distribute the load across multiple nodes.

Cascading[**?**] strategies:

1. **Proximity-Based Cascading[?]:** Clients connect to the nearest Media Node, which relays streams to the node closest to the recipient, optimising for low latency.
2. **Regional Cascading[?]:** Clients within the same region connect to different Media Nodes, which exchange streams before forwarding to recipients, potentially improving load balancing in a region
3. **Fanout Cascading[?]:** Clients in multiple regions connect to Media Nodes in their respective regions, which receive a stream from a primary Media Node acting as the source. This setup ensures low-latency content delivery, making it ideal for live streaming and similar use cases. ****

4.3.2 Example of Proximity-Based Cascading

Participants: Alice, Bob, Dan, and Eve are located in different parts of the world.

Let's assume:

- Alice is in New York, USA
- Bob is in Tokyo, Japan
- Dan is in Tallinn, Estonia
- Eve is in Sydney, Australia

Without Cascading[?] (Hypothetical)

- **Centralised SFU[6]:** Imagine a scenario where all participants connect to a single, centralised SFU[6], most likely located in a data centre in North

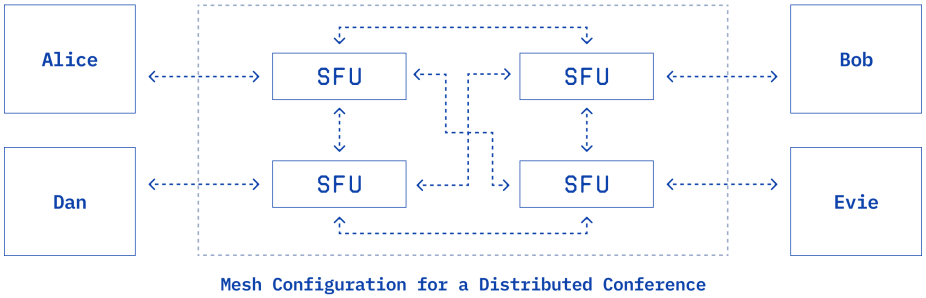


Figure 10: Mesh Configuration for a distributed workload

America.

- **Long-Distance Transmission:** In this case, media streams from participants like Bob (in Tokyo) and Eve (in Sydney) would need to traverse vast distances to reach the centralised SFU[6] and then back to other participants.
- **Consequences:**
  - **Increased Latency:** The long-distance transmission would introduce significant latency, causing delays in audio and video, and hindering natural conversation flow.
  - **Potential Packet Loss:** The greater the distance, the higher the chances of packet loss due to network congestion or disruptions, leading to audio/video glitches and a degraded experience.

With Cascading[?] on dRTC Network

- **Geolocation-Based Connection:**
  - Alice connects to the Media Node in North America, near New York, USA.
  - Bob connects to the Media Node in Asia, near Tokyo, Japan.
  - Dan connects to the Media Node in Europe, near Tallinn, Estonia.
  - Eve connects to the Media Node in Oceania, near Sydney, Australia.
- **Mesh Network of Media Nodes:** These geographically distributed Media Nodes are interconnected in a mesh topology via the dRTC backbone.

Benefits:

- **Reduced Latency:** Participants connect to nearby Media Nodes, minimising the distance their media streams travel, leading to significantly lower latency and a more real-time communication experience.
- **Minimised Packet Loss:** Shorter distances reduce the likelihood of packet loss, ensuring smoother audio and video quality.
- **Efficient Distribution:** The mesh network ensures efficient distribution of media streams between Media Nodes, regardless of the participants' locations.

4.3.3 Example Fanout Cascading

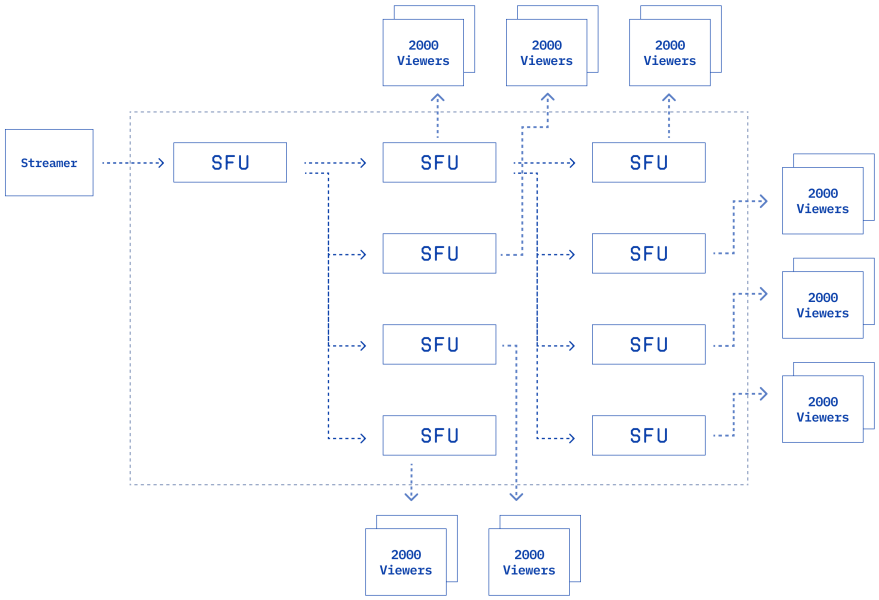


Figure 11: Cascaded System for Livestreaming

Live-streaming prioritises efficient distribution of a single media stream from the streamer to a large audience

Figure 11 shows the process initiates with the "Streamer" sending their live media stream to a single, primary Media Node. This Media Node acts as the central point of ingestion for the livestream. The primary media node then "fans out" the received stream, distributing it to multiple other Media Nodes strategically located across the globe. This distribution leverages the "dRTC backbone," ensuring the stream's availability to a geographically diverse audience. Each of these distributed Media Nodes caters to a group of "Viewers" within their respective regions. This localised delivery model reduces latency and improves the overall viewing experience for the audience, as the streams travel shorter distances.

Without Cascading[?] (Hypothetical)

- **Centralised Streaming:** In a traditional livestreaming setup without cascading[**?**], all viewers would directly connect to a single, centralised Media Node, regardless of their geographical location. This central node would be responsible for receiving the streamer's feed and transmitting it to every viewer.
- **Bandwidth Bottlenecks at the Central Node:** The central node

would need to handle the massive upstream bandwidth from the streamer and simultaneously transmit the stream to potentially thousands or even millions of viewers. This can create a significant bottleneck, leading to buffering, dropped frames, or even complete service outages.
- **Single Point of Failure:** The reliance on a single central node creates a single point of failure. If this node experiences any technical issues or outages, the entire livestream is disrupted for all viewers.
- **Limited Scalability:** The central node's capacity to handle a large number of viewers is limited. As the audience grows, the system may struggle to maintain performance and reliability.

In contrast, the cascading[**?**] approach illustrated in Figure 11 , mitigates these challenges by distributing the load across multiple Media Nodes, ensuring a smoother and more scalable live-streaming experience for a global audience

Benefits of Cascading[?]

- **Scalability:** The fan-out mechanism allows the system to scale effortlessly, accommodating a massive number of viewers across the globe.
- **Reduced Latency:** By serving viewers from geographically closer Media Nodes, the latency associated with media stream delivery is minimised.
- **Efficient Bandwidth Utilisation:** The centralised ingestion model reduces the upstream bandwidth requirements for the streamer, as they only need to send a single stream to the primary Media Node.
- **Fault Tolerance:** If one of the distributed Media Nodes encounters an issue, viewers connected to it can be seamlessly redirected to other operational Media Node, ensuring uninterrupted streaming.

4.3.4 Media Node Modules

4.3.5 Media Engine

The Media Engine is a WebRTC[13] SFU (Selective Forwarding Unit)[6] responsible for managing different WebRTC[13] connections and routing media packets across these connections.

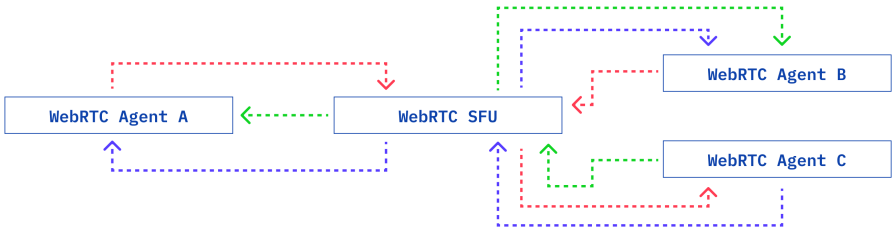


Figure 12: Clients connected to an SFU

Every Client creates a WebRTC[13] Peer Connection with the SFU[6] and sends

their media stream to the SFU[6] and the SFU[6] forwards the media stream to the other participants(Fig 4).

Advantages of using SFUs[6]:

- **Bandwidth Efficiency:** Each participant sends their media stream only once, reducing overall upload bandwidth requirements.
- **Scalability:** The model supports a large number of participants through efficient stream handling.
- **Adaptive Streaming:** Techniques like Simulcast[5] enable adaptation to varying network conditions, ensuring a smooth experience for all participants.

4.3.6 RPC Module

The RPC Module enables Media Nodes to receive and execute instructions from Orchestrators within the network, facilitating communication between Media Node and Orchestrator.

It has the following functions:

- The RPC Module exposes a set of RPC Methods for the Orchestrator to utilise to manage RTC sessions which include methods like creating rooms, consuming and producing media streams, etc
- The RPC Module listens for requests from Orchestrators, processes them and relays them to the Media Engine Module

4.3.7 Registry Module

The Registry Module handles the communication between the Media Node and the Registry Network.

It has the following functions:

- **Advertisement of Capabilities:** It broadcasts the Media Node's available resources to the network, including bandwidth, compute capabilities and QoS (Quality of Service) reports.
- **QoS Test Facilitation:** It actively listens for QoS test requests from the Registry Network and subsequently relay these requests to the QoS module within the Media Node for execution.

4.3.8 QoS Module

The QoS (Quality of Service) Module verifies the resources provided by individual nodes in dRTC networks by enforcing resource accountability for Media Nodes. It offers a set of APIs that enable resource policing, preventing issues like bandwidth hogging, CPU overuse, and latency problems that could occur if any of the nodes fail to adhere to their advertised capabilities.

4.3.2.4.1 QoS Process

The Registry Network selects two Media Nodes for testing: a Prover, which provides resources, and a Challenger, which tests those resources. The Prover and Challenger then establish a WebRTC[13] connection.

The Prover generates and sends a Video Media Stream to the Challenger. The Challenger receives this stream and calculates various QoS scores:

- **Bitrate Score:** Measures the Prover's ability to maintain a consistent bitrate.
- **Fractional Loss Score:** Evaluates the Prover's packet loss rate.
- **Latency Score:** Assesses the Prover's latency performance.

Note: For a more detailed explanation of the node selection process and the rationale behind these QoS metrics, please refer to the QoS Module section in the Registry documentation.

4.3.2.4.2 QoS Score Calculations

The QoS score calculation process is periodic where the Challenger periodically checks the WebRTC[13] Connection stats at 5-second intervals.

```
{
  timestamp: 861130149,
  ssrc: 805818640,
  kind: 'video',
  mimeType: 'video/H264',
  roundTripTime: 0.0152587890625,
  type: 'inbound-rtp',
  jitter: 86,
  byteCount: 649149,
  packetCount: 527,
  bitrate: 743117,
}
```

[15]

Using these stats, the Challenger calculates different scores that are essential for determining the QoS of the Media Node, the calculations for these scores are as follows:

Bitrate Score Calculation

The Bitrate Score measures how well a media stream maintains a consistent and adequate data transfer rate. The calculation process involves the following steps:

EWMA Calculation:

For each bitrate value, calculate the new EWMA using the formula:

$$\text{EWMA} = \alpha \cdot \text{new\_bitrate} + (1 - \alpha) \cdot \text{previous\_EWMA}$$

where:

- α is the weighting factor, $0 < \alpha \leq 1$
- new_bitrate is the new observation of bitrate
- previous_EWMA is the previous EWMA value

Categorisation:

Classify each bitrate value as 'good' or 'bad' based on its deviation from the EWMA. Here's the Go implementation:

Algorithm 3 Bitrate Bucket Calculation

```
1: function BitrateBucketCalculation(value, ewma)
2:     if ewma = nil then return ""
3:     end if
4:     diff ← |value - *ewma|
5:     threshold ← max(100, *ewma * 0.15)
6:     if threshold < diff then return "bad"
7:     elsereturn "good"
8:     end if
9: end function
```

This function does the following:

- Calculates the absolute difference between the new bitrate value and the EWMA, normalising the value between 0 and 1, indicating overall bitrate stability.
- Determines a threshold, which is the maximum of 100 or 15% of the EWMA.
- If the difference exceeds the threshold, the bitrate is categorised as 'bad'; otherwise, it's 'good'.

Score Calculation

The Bitrate Score is the ratio of 'good' values to the total number of values:

$$\text{Bitrate Score} = \frac{\text{Number of 'good' bitrate values}}{\text{Total number of bitrate values}}$$

This approach allows for adaptive evaluation of bitrate performance, taking into account both absolute and relative changes in the bitrate. The use of EWMA ensures that the score reflects recent trends while smoothing out short-term fluctuations.

Fractional Loss Score Calculation

The Fractional Loss Score assesses the reliability of data transmission by measuring packet loss.

Categorisation

Classify each fractional loss value as 'good' or 'bad' based on a fixed threshold. Here's the Go implementation:

Algorithm 4 Fractional Loss Bucket Calculation

```
1: function FractionalLossBucketCalculation(value)
2:     if value > 0.05 then return "bad"
3:     elsereturn "good"
4:     end if
5: end function
```

This function does the following:

- Compares the fractional loss value to a fixed threshold of 0.05 (5%), normalises value between 0 and 1, indicating the reliability of packet transmission.
- If the fractional loss exceeds 5%, it's categorized as 'bad'; otherwise, it's 'good'.

Score Calculation

The Fractional Loss Score is the ratio of 'good' values to the total number of values:

$$\text{Fractional\_Loss\_Score} = \frac{\text{Number of 'good' fractional loss values}}{\text{Total number of fractional loss values}}$$

This approach provides a straightforward evaluation of network reliability:

- A fractional loss below 5% is considered acceptable for most real-time communications.
- Consistently high fractional loss (above 5%) indicates potential network issues that could affect the quality of communication.

Latency Score Calculation

The RTT Score measures the network latency between the Prover and Challenger. The calculation process involves the following steps:

EWMA Calculation

Maintain an Exponentially Weighted Moving Average (EWMA) of the RTT:

$$\text{EWMA} = \alpha \cdot \text{new\_value} + (1 - \alpha) \cdot \text{previous\_EWMA}$$

Where α (alpha) is typically 0.1, giving more weight to recent data.

Categorisation

Classify each RTT value as 'good' or 'bad' based on its deviation from the EWMA[**?**]. Here's the Go implementation:

Algorithm 5 Latency Bucket Calculation

```
1: function LatencyBucketCalculation(value, ewma)
2:     if ewma = nil then return ""
3:     end if
4:     if |value - *ewma| > 20 then return "bad"
5:     elsereturn "good"
6:     end if
7: end function
```

This function does the following:

- Calculates the absolute difference between the new RTT value and the EWMA[**?**], normalises value between 0 and 1, representing the consistency of round-trip time.
- Compares this difference to a fixed threshold of 20ms.
- If the difference exceeds 20ms, the RTT is categorised as 'bad'; otherwise, it's 'good'.

Score Calculation

The Latency Score is the ratio of 'good' values to the total number of values:

$$\text{Latency Score} = \frac{\text{Number of 'good' RTT values}}{\text{Total number of RTT values}}$$

This approach allows for adaptive evaluation of network latency:

- It considers the relative change in RTT rather than absolute values, making it adaptable to different network conditions.
- The 20ms threshold for deviation provides a balance between allowing for normal fluctuations and detecting significant latency spikes.
- Using EWMA[**?**] ensures that the score reflects recent trends while smoothing out short-term fluctuations.

4.3.2.4.3 QoS Reporting

At the end of the QoS process, the Media Node compiles a comprehensive report of the Prover's performance metrics. This report, structured as a JSON payload, provides a detailed snapshot of the node's quality metrics. The report includes the final scores as well as the snapshots of raw data for each metric.

Example QoS Report:[15]


```
{
  "scoreSnapshot": {
    "bitrateScore": 0.6153846153846154,
    "bitrateSnapshot": [
      64.614, 267.958, 599.459, 742.682, 818.534,
      736.214, 834.406, 743.117, 811.645, 730.448,
      822.909, 741.19, 834.986
    ],
    "latencyScore": 1,
    "latencySnapshot": [
      0, 0, 0, 1.007080078125, 1.007080078125,
      0.0152587890625, 0.0152587890625, 0.0152587890625,
      ↪ 0.0152587890625, 0.0152587890625,
      0.0152587890625, 0, 0
    ],
    "fractionalLossScore": 0.982,
    "fractionalLossSnapshot": [
      0.01, 0.005, 0.02, 0.015, 0.03,
      0.01, 0.025, 0.02, 0.035, 0.015,
      0.005, 0.01, 0.02
    ]
  }
}
```

5 dRTC Chain

Transparent and equitable operations are essential for the success of the dRTC Network. Key aspects of this are fair and open compensation of node operators, which play a crucial role in maintaining and supporting the network. The immutability of Proof of Resource metrics over time is fundamental to the reputations of nodes in the network, enforcing provable meritocracy across the network. Equally important is transparency in global network state changes. The dRTC Network's core values include openness, decentralization, neutrality, and resistance to censorship. Upholding these values is important when developing any component of the network.

A "blockchain" has proven to be an ideal solution to address these challenges. However, given the limitations of existing blockchains, the dRTC Network has opted for a Layer 3 blockchain architecture called the 'dRTC Chain'. The dRTC Chain is a blockchain that settles transactions on an existing Layer 2 blockchain i,e Arbitrum One. By implementing a Layer 3 blockchain, the dRTC Network creates a solid foundation for the network's future growth and success.

We chose to build on top of the Ethereum ecosystem to leverage the security, scalability, and adoption of its existing solutions because it aligned with our ethos. This allowed us to focus on optimizing the dRTC Network and the dRTC Chain rather than reinventing the wheel while benefiting from the already established infrastructure and developer community. This also presents an opportunity for us to contribute to a value-aligned ecosystem.

5.1 Our Ecosystem of choice: Ethereum

Ethereum, launched in 2015, is a decentralized, open-source blockchain platform that enables the creation of smart contracts and decentralized applications (dApps). As the second-largest blockchain by market capitalization, Ethereum has been at the forefront of the blockchain industry, driving innovation and adoption across various sectors[**?**].

5.1.1 Current Ethereum Landscape

Scalability Challenges and Layer 2 Solutions
As Ethereum's popularity has grown, the network has faced scalability challenges. To address these issues, the Ethereum community developed Layer 2 scaling solutions, such as rollups. Rollups process transactions off-chain and periodically commit data to the Ethereum mainchain, thereby increasing throughput and reducing gas fees while maintaining the security of the Ethereum network. As of March 2024, the Ethereum ecosystem has seen significant growth in Layer 2 solutions, with 45 rollups launched and another 34 projects in development.

Market Share of Leading Layer 2 Solutions
Among the various Layer 2 solutions, Arbitrum and Optimism have emerged as

market leaders, capturing 42% and 24% of the market share, respectively. Other notable projects include Starknet, Base, and zkSync, which have also gained traction in the Ethereum community .

5.2 Challenges for the dRTC Network

While Ethereum and its Layer 2 solutions[**?**] align with the values of the dRTC Network's, the extensive scale at which the dRTC Network is expected to operate poses some challenges.

1. **Scalability:** The ability to handle a high volume of transactions and state updates without compromising performance.
2. **Cost-effectiveness:** Ensuring that transaction costs remain low enough to make the protocol economically viable for all participants.
3. **Isolated state updates:** The capability to update the state of the protocol without being affected by unrelated applications.
4. **Ethereum compatibility:** Despite the need for a custom solution, maintaining compatibility with the Ethereum ecosystem remains a priority.

5.3 The dRTC Chain

To meet the unique needs of the dRTC Network, a dedicated blockchain called the dRTC Chain has been developed. The dRTC Chain is designed to:

1. Support high-frequency state updates required for real-time communication services.
2. Implement efficient incentivization and consensus mechanisms without prohibitive gas costs.
3. Allow for isolated state updates, ensuring the stability and independence of different applications within the ecosystem.
4. Maintain compatibility with Ethereum, enabling integration with existing tools and allowing for future interoperability[**?**].

5.3.1 Our Choice of Layer 2: Arbitrum

Arbitrum is a Layer 2 scaling solution[7] that utilizes optimistic rollups to enhance transaction efficiency and reduce costs associated with transactions on the Ethereum. As of July 2024, Arbitrum commands 39.36% of the Layer 2 market and secures $16.90 billion in Ethereum escrow[**?**]. Arbitrum Orbit allows for the creation of customizable chains that interoperate seamlessly with the broader Ethereum network, and it serves as the foundation for the dRTC Chain. By leveraging Arbitrum Orbit, the dRTC Chain is optimized for scalability, cost-efficiency, and state management. This custom-built chain ensures smooth interoperability with Ethereum while preventing liquidity fragmentation.

5.3.2 dRTC Chain: Deep Dive

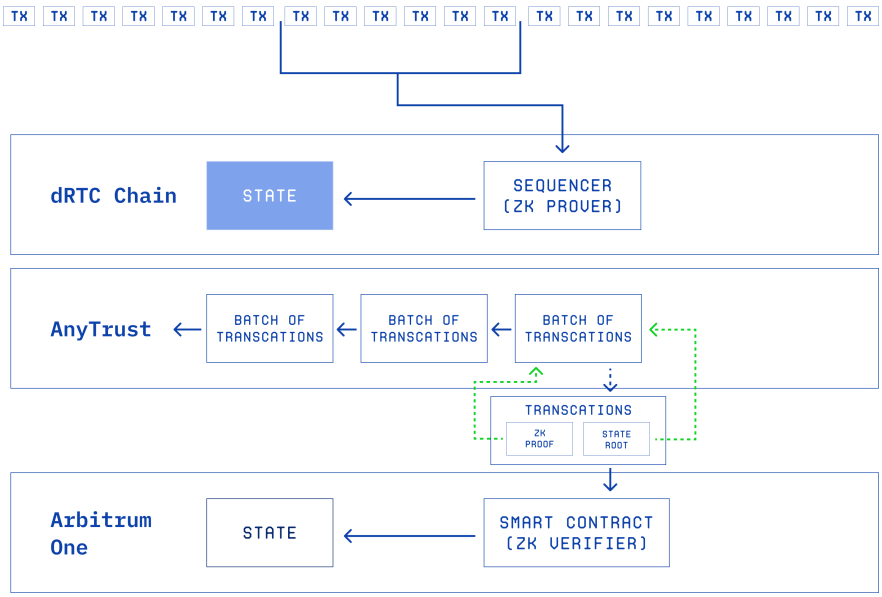


Figure 13: dRTC Chain Transaction Flow

The dRTC Chain utilises AnyTrust protocol by Arbitrum which settles on Arbitrum One to meet all previously mentioned requirements using Ethereum (ETH) as gas fees[**?**]. It employs a permissioned Data Availability Committee (DAC) to store transaction data off-chain, instead of on the Ethereum or Arbitrum Rollup[3] chains. The DAC, a permissioned group of nodes, ensures data availability for the blockchain and operates with the premise that at least two of its members remain honest. The Data Availability Committee (DAC) enhances the performance of the dRTC Chain by ensuring off-chain data storage through a trust-minimised design, requiring a quorum of honest parties to confirm data availability. This allows the dRTC chain to achieve the desired scalability while maintaining security.

The dRTC chain processes transactions through a sequencer, which collects and orders transactions before they are submitted to the Data Availability Committee (DAC). This sequencer helps maintain efficient throughput and reduce latency in transaction processing.

DACs consist of a specified number (N) of parties, operating under the premise that at least a configurable number (H) of these members are honest. Each DAC member runs a Data Availability Server (DAS) to store transaction data off-chain. To confirm storage, a quorum of `K = (N + 1) - H` members must sign a certificate. The chain's sequencer sends data batches to the DAC, which issues

signed certificates guaranteeing data storage for a designated period. Nodes can later verify data availability through these certificates, ensuring access as long as the minimum number of honest members (H) is maintained.

Transactions that are batched together and sent to the DAC, confirms their availability. The sequencer also manages the creation of Data Availability Certificates (DACerts) that are posted to Ethereum[**?**] to signal that the data is available and valid.

5.4 Core Components of the dRTC Network

The dRTC Network employs the dRTC Chain through a system of smart contracts to manage its core operations. These contracts work in concert to facilitate node registration, manage delegation processes, distribute incentives, and maintain the network's global state.

5.4.1 Core Smart Contracts

The network's functionality is underpinned by several key smart contracts:

1. Pool Manager: Oversees the creation and management of node pools.
2. Pool Contract: Handles the operations within individual node pools.
3. Media Node Key Contract (ERC-1155): Manages the unique identifiers for Media Nodes.
4. veHUDL Token Contract (ERC-20): Non tradeable, non transferable token that can only be procured as rewards
5. HUDL Token Contract (ERC-20):
6. Proof of Resource Contracts: Store records of the resources provided by nodes.
7. Rewards Contract: Calculates and distributes rewards to participants.
8. Reserve Contracts: Manage the network's financial reserves.

These contracts form the backbone of the dRTC Network, ensuring its efficient and secure operation.

5.5 Media Node Participation

Participation in the dRTC Network as a Media Node is contingent upon ownership of a "Media Node Key". This is a non-fungible token that serves as a license, granting the holder the right to:

- Operate a Media Node within the network
- Provide dRTC data to Clients and their applications
- Earn rewards proportional to their contributions to the network

The Media Node Key is a crucial component in the network's architecture, aligning incentives and ensuring committed participation.

5.6 Media Node Key Delegation

Delegation is key enabler of accessibility to participating as a Media Node on the dRTC Network. It serves as a mechanism to optimise resource allocation and broaden network participation. The delegation process unfolds in two primary ways:

1. Self-Delegation: Node Key owners who operate their own nodes can delegate their keys to themselves. This process not only confirms their commitment to active participation but also allows them to bring higher-quality nodes into the system and reduce operational expenses by factors. By maximizing node performance, these owners can increase their potential rewards, as better quality nodes contribute more effectively to the network's stability and performance.
2. Third-Party Delegation: Key owners who are either unable or unwilling to operate their own nodes can delegate their keys to active node operators. This allows them to contribute to the network and earn rewards without the technical overhead of running a node. For node operators, receiving such delegations is a strong signal of trust and reputation, affirming their reliability as a high-performing entity.

Nodes can accept delegations from multiple key holders, with the number of delegations proportional to the node's contributed resources. This system incentivizes node operators to provide high-quality service and contribute significant resources to the network. In return, they can earn a commission from the rewards generated by delegated keys. Delegation reduces operational expenses for key holders, as they can manage multiple keys on a single node or delegate their keys to quality nodes instead of running individual nodes for each key. This creates a competitive marketplace for delegation services, driving improvement and efficiency in the network.

The delegation system offers several key benefits to the dRTC Network:

1. Quality Assurance: By allowing key holders to delegate to high-performing nodes, the system naturally selects for and rewards quality node operators.
2. Lower Barrier to Entry: Delegation lowers the barrier to network participation, allowing key holders to contribute without the need for direct infrastructure management. It enables a broader range of individuals and entities to contribute to and benefit from the network.
3. Lower Operational Costs: Node operators can accept delegations to offset costs through commissions or efficiently manage multiple keys on a single node, optimizing their resource utilization.
4. Reputation Building: The delegation process serves as a foundation for a reputation system among nodes, where consistently high-performing nodes can attract more delegations.
5. Enhanced Security: Similar to the concept of Total Value Locked (TVL) in DeFi protocols, the delegation system contributes to the overall security and stability of the network by ensuring a high stake in its proper functioning.

Similar to Ethereum's 32 ETH staking limit[**?**], the dRTC Network also implements a maximum resource cap per node. This cap ensures that no single node can dominate the network, promoting a more distributed and resilient system. Additionally, there's a limit on the total number of active nodes, which helps maintain network efficiency and manageability.

5.6.1 Delegation Lifecycle

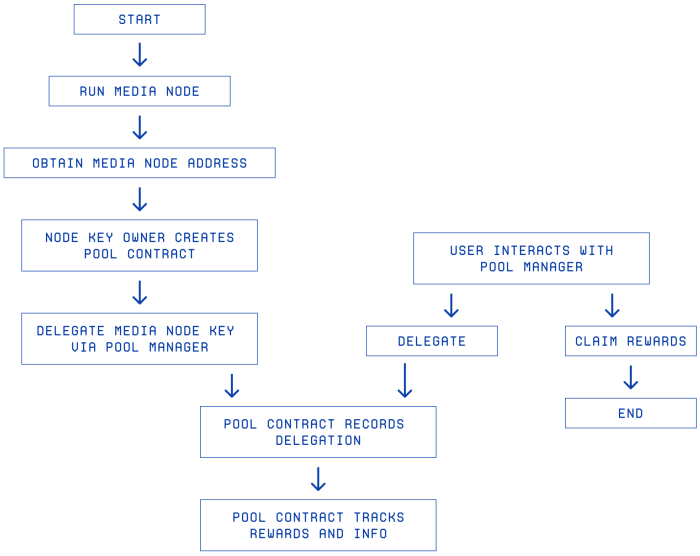


Figure 14: Delegation Lifecycle

The delegation process in the dRTC Network involves several steps and key components:

1. Media Node Initialisation: When a Media Node is initialized, it is immediately provided with a unique Media Node Address. This address serves as the identifier for that specific node in the network.
2. Pool Contract Creation: A Node Key owner must create a "Pool Contract" for the Media Node they wish to delegate to. This is done by using the Media Node Address obtained in step 1. The creation process is facilitated through the "Pool Manager" contract, which acts as an interface for interacting with Pool Contracts.
3. Delegation: The Node Key owner delegates their Media Node Key to the newly created Pool Contract. This delegation is also performed through the Pool Manager contract, which ensures proper recording and management of the delegation.
4. Pool Contract: The Pool Contract serves as a record-keeper for the Media Node. It maintains a comprehensive list of all delegations made to that specific Media Node. Additionally, it tracks other crucial information such

as:

- Rewards earned by each delegated key
- Identity of key owners
- Other relevant data pertaining to the node's operation and performance

5. Ongoing Management: The Pool Manager contract continues to serve as the primary interface for anybody to interact with any Pool Contract. Through Pool Manager, one can:
   - Make additional delegations to the Media Node
   - Claim rewards earned from their delegations
   - Access other functions related to their participation in the node's operations

In the dRTC Network, delegation is a process that doesn't involve the transfer of ownership of the Media Node Key NFT. Instead, it creates a record in the smart contracts through mapping structures.

5.7 Proof of Resource

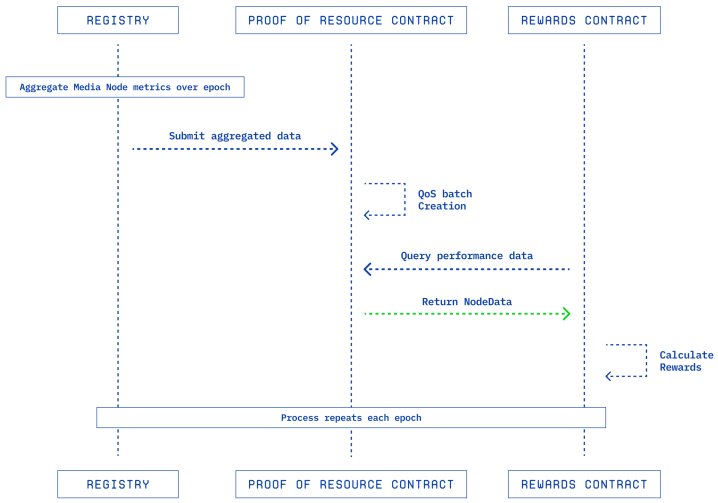


Figure 15: Proof of Resource

The Proof of Resource contract is a component of the dRTC Network, serving as the on-chain repository for Quality of Service (QoS) and other important data collected by the Registry. This contract bridges the gap between off-chain performance monitoring and on-chain reward distribution.

As outlined in the QoS Module , the QoS test results are aggregated by the Registry, along with other important operational metrics, which are defined over epochs and then committed to the Proof of Resource contract.. The data is

structured in the following format:

```
struct NodeData {
    uint256 totalBandwidth;
    uint256 lastRewardedBatch;
    ... other important metrics
}
```

This data serves multiple purposes:

1. It provides the foundation for the Rewards Contract to calculate fair and accurate compensation for each Media Node.
2. It establishes a verifiable history of node performance.

5.8 Implementation of Media Node Rewards

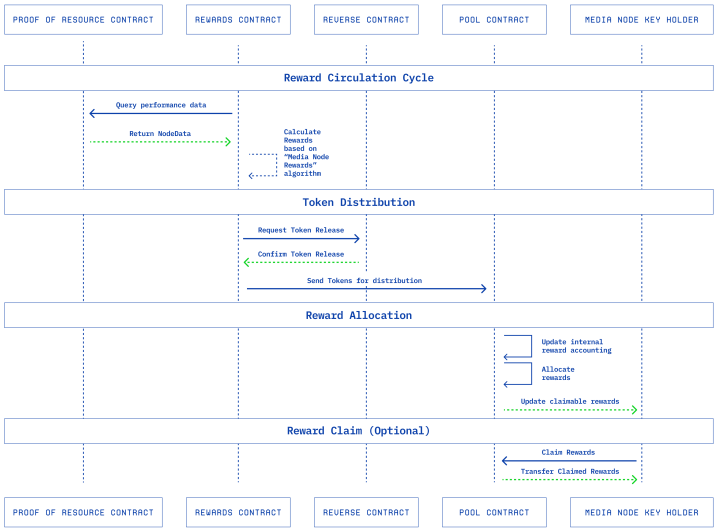


Figure 16: Implementation of Media Node Rewards

Building upon the Proof of Resource contract's role in storing and managing QoS data, the dRTC Network implements a data-driven incentivization mechanism. This system coordinates interactions between the Rewards Contract, Proof of Resource Contract, and Reserve Contracts to fairly compensate Media Nodes for their contributions.

The process works as follows:

1. The Rewards Contract queries the Proof of Resource Contract to retrieve node data.
2. Based on this data, it calculates the appropriate rewards for each Media Node, as outlined in the Dynamic Minting section.

3. The Rewards Contract determines the total token amount for the current epoch.
4. It calls a Reserve Contract to release the required tokens.
5. These tokens are transferred to the respective Pool Contract associated with each Media Node.
6. The Pool Contract distributes the rewards according to its predefined rules.

6 Applications on the dRTC Ecosystem

The dRTC Ecosystem, with its unique implementation of decentralized real-time communication infrastructure on a blockchain, has already demonstrated its efficacy in powering successful video conferencing, streaming, and audio-only applications across various ecosystems. Notable implementations include decentralized alternatives to traditional video meeting platforms and audio spaces, showcasing the system's ability to deliver high-quality, low-latency communication. Building on this foundation, the dRTC Ecosystem presents significant potential for further transformative applications across multiple domains, such as:

1. **Metaverse and Virtual Worlds**: The dRTC Network could provide the communication backbone for decentralized metaverse platforms, enabling high-quality, low-latency audio and video interactions. The dRTC Chain could manage digital asset ownership and governance, potentially revolutionizing virtual economies and social spaces.
2. **Decentralized Finance (DeFi) and High-Frequency Trading (HFT)**: The dRTC Ecosystem's real-time, secure and low-latency capabilities combined with the security of the dRTC Chain, could facilitate decentralized trading rooms, real-time market analysis, cross-chain arbitrage, and AI-driven trading strategies potentitally paving way for new primitives of financial instruments with near-instant settlement.
3. **Gaming and E-sports**: The dRTC Ecosystem has the potential to enhance online gaming experiences. Its low-latency network could enable seamless, decentralized multiplayer gaming supported by in-game multimedia capabilities. The dRTC Chain could facilitate in-game asset ownership and trading. This infrastructure could support new models for e-sports and player-owned game economies.
4. **SocialFi**: The dRTC Ecosystem could usher in a new era of social applications by combining real-time multimedia capabilities with dRTC Chain functionalities. This could enable SocialFi features like tokenized social interactions and decentralized content monetization, reshaping creator economies with more equitable, user-centric platforms.
5. **AI Agents**: The dRTC Network could serve as a communication layer for AI agents, enabling real-time interactions between multiple AIs or between AIs and humans. This could have applications in areas such as decentralized AI marketplaces, collaborative problem-solving, and advanced virtual assistants.
6. **Advanced Content Delivery Networks (CDNs)** : The dRTC Ecosystem could be utilized for a sophisticated, multi-protocol CDN infrastructure for serving both live and static content. Integrations of technologies like WebTransport could be combined with intelligent content routing based on viewer geography, network conditions, and content popularity. This approach could significantly enhance performance and scalability, resulting in a versatile, decentralized CDN that rivals traditional architectures in

efficiency, adaptability, and cost across real-time and on-demand scenarios.

The domains discussed represent key areas for future research and development within the dRTC Ecosystem. As this ecosystem evolves, it promises to enable innovative use cases that harness the synergy between the dRTC Network's real-time communication capabilities and the dRTC Chain's security, scalability, and transparency.

7 Future Work

The following topics represent future and on-going work:

7.1 Protocol Economics

- Develop secondary value accrual mechanisms, including staking, restaking, and liquidity pools
- Implement governance mechanisms for the dRTC network, enabling collective decision-making to influence hyperorganic[12] policy updates
- Design mechanisms to ensure geographical distribution of Media Nodes through strategic Node Key sales
- Refine the Media Node reward function to incorporate meeting-specific quality data
- Implement surge rewards based on temporal demand fluctuations and supply variations
- Develop an internal accounting mechanism using data credits
- Create financial instruments and DeFi primitives utilizing incentivized testnet tokens
- Validate and iterate protocol mechanics through multi-agent simulations[2]

7.2 System & Network Architecture

- Implement a secret leader election protocol for Expected Consensus, ensuring one elected leader per epoch
- Develop detailed specifications for various system components in the dRTC Network
- Complete the implementation of a Hierarchical Consensus protocol, enabling the dRTC network to identify and penalize malicious actors
- Design the protocol that uses Quality of Service (QoS) to construct reputation systems. This system will be developed and refined through extensive testing on a dedicated testnet environment.

7.3 General Improvements

- Conduct comprehensive verification and network testing
- Develop systems design and protocol economics for Layer01, an execution validation layer
- Optimize Media Node delegation to address the requirements of the dRTC network's live phase, with a focus on long-term sustainability
- Enhance the dRTC Chain to become the preferred connectivity solution for SocialFi, GameFi, and applications across various domains

8 Glossary

- **Prosumer** : A marketplace agent who can simultaneously contribute as a supplier/producer and as a consumer of the commodity or services on the market. In the context of the dRTC network, a prosumer is a role of an agent who may participate as a Media Node supplying real-time data and/or as a Client consuming real-time data via applications.
- **Hyperorganic Systems**: hyperorganic[12] systems implement coordination protocols where the policies are functions of the system state, instead of being prescriptive or predefined. This allows for their economic dynamics to be inherently free, dynamically adaptive, and emergent.
- **Conflict-free Replicated Data Types (CRDTs)** : A data structure that allows multiple nodes to concurrently modify shared data without requiring strict coordination. CRDTs ensure data consistency and enable seamless data replication across the network.
- **WebRTC (Web Real-Time Communication):** A free, open-source project that enables real-time communication capabilities in web browsers and mobile applications.
- **SFU (Selective Forwarding Unit)**: A type of video routing mechanism in WebRTC[13] that receives multiple media streams and selectively forwards them to the intended recipients.
- **RTP (Real-Time Transport Protocol)**: is a network protocol for delivering audio and video over IP networks. RTP[22] is used in communication and entertainment systems that involve streaming media, such as telephony, video teleconference applications including WebRTC[13].
- **Simulcast**: A technique in WebRTC that allows a sender to transmit multiple versions of the same video stream at different resolutions and bitrates, enabling adaptive streaming.
- **EWMA (Exponentially Weighted Moving Average)**: A type of moving average that gives more weight to recent data points, making it more responsive to changes in data.
- **Gossip Protocol**: A peer-to-peer communication protocol in which nodes randomly exchange information with each other to efficiently propagate data across a network.
- **RTT (Round-Trip Time)**: The time it takes for a data packet to travel from a sender to a receiver and back again, used as a measure of latency in network communications.
- **Blockchain**: A distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes.
- **Consensus Mechanism**: The mechanisms used in blockchain networks to achieve agreement among distributed nodes about the current state of the ledger.
- **Ethereum**: A decentralized blockchain that enables the creation and execution of smart contracts and decentralized applications (dApps).
- **EVM**: The Ethereum Virtual Machine (EVM) is the decentralized[**?**] runtime environment that executes smart contracts on the Ethereum

blockchain.

- **Layer 2 (L2)**: Refers to a set of scaling solutions built on top of the Ethereum blockchain[**?**] (Layer 1) designed to increase transaction throughput and reduce fees without compromising security.
- **Arbitrum One**: A leading Layer 2[7] scaling solution for Ethereum, employing optimistic rollups.
- **Liquidity Fragmentation**: Refers to a situation where available liquidity in a blockchain is dispersed across multiple blockchains rather than being concentrated in a single location.
- **Smart Contracts**: Self-executing programs that automatically enforce and execute the terms of an agreement based on predefined conditions, without the need for intermediaries.
- **Stylus**: Arbitrum Stylus[3] is an upgrade to Arbitrum[3], enabling developers to write smart contracts using multiple programming languages, including Rust, C, and C++.

References

[1] Alexiou, A., et al., "QoS for WebRTC-based Video Communication," *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1016-1017, 2016.

[2] Appiah, S., "Decentralized Organizations as Multi-Agent Systems - A Complex Systems Perspective," ResearchGate, Mar. 2017. [Online]. Available: https://www.researchgate.net/publication/319875145_Decentralized_Organizations_as_Multi-Agent_Systems_-_A_Complex_Systems_Perspective

[3] Kalodner, H., et al., "Arbitrum: Scalable, Private Smart Contracts," *USENIX Security Symposium*, 2018.

[4] Grozev, B., "Improving Scale and Media Quality with Cascading SFUs," Nov. 12, 2018. [Online]. Available: https://webrtchacks.com/sfu-cascading/

[5] Burman, B., et al., "Using Simulcast in SDP and RTP Sessions," IETF, Mar. 5, 2019. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-mmusic-sdp-simulcast

[6] Rescorla, E., "WebRTC Security Architecture," IETF, Jul. 21, 2019. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-rtcweb-security-arch

[7] Gudgeon, L., et al., "SoK: Layer-Two Blockchain Protocols," *Financial Cryptography and Data Security*, 2020.

[8] Buterin, V., "Why Proof of Stake," Nov. 6, 2020. [Online]. Available: https://vitalik.eth.limo/general/2020/11/06/pos2020.html

[9] Appiah, S., "Risk Adjusted Bonding Curves" YouTube, Jun. 14, 2021. [Online]. Available: https://www.youtube.com/watch?v=LfwgRSb-uG0

[10] Buterin, V., "An Incomplete Guide to Rollups," Jan. 5, 2021. [Online]. Available: https://vitalik.eth.limo/general/2021/01/05/rollup.html

[11] Dias, J. P., et al., "Libp2p: A Network Stack for Peer-to-Peer Applications," *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pp. 91-94, 2021.

[12] Appiah, S., "Hyperorganic Systems," YouTube, Huddle01, EthCC, Apr. 28, 2023. [Online]. Available: https://www.youtube.com/watch?v=MfCfwSSo35U

[13] W3C, "WebRTC: Real-Time Communication in Browsers," Mar. 6, 2023. [Online]. Available: https://www.w3.org/TR/webrtc/

[14] Buterin, V., "Layer 2s as Cultural Extensions of Ethereum," Vitalik.eth.limo, Jun. 19, 2024. [Online]. Available: https://vitalik.eth.limo/general/2024/05/29/l2culture.html

[15] W3C, "WebRTC Statistics API," Jul. 18, 2024. [Online]. Available: https://w3c.github.io/webrtc-stats/

[16] Kurundodi, A., "Scaling Decentralized Low Latency Infrastructure at Huddle01," YouTube, IPFS Camp 2024, Jun. 2024. [Online]. Available: https://www.youtube.com/watch?v=YhOt80N4fWQ

[17] Arbitrum, "Inside Anytrust," Arbitrum Documentation. [Online]. Available: https://docs.arbitrum.io/how-arbitrum-works/inside-anytrust

[18] Arbitrum, "Arbitrum Orbit," Ankr.com. [Online]. Available: https://www.ankr.com/docs/scaling-services-rollups/stacks/arbitrum-orbit/

[19] Protocol Labs, "libp2p Documentation," https://docs.libp2p.io/

[20] Larimer, D., "Delegated Proof-of-Stake (DPOS)," Bitcointalk.org, Apr. 3, 2014. [Online]. Available: https://bitcointalk.org/index.php?topic=558316.0

[21] Buterin, V., "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," Apr. 2014. [Online]. Available: https://ethereum.org/

[22] Schulzrinne, H., et al., "RTP: A Transport Protocol for Real-Time Applications," IETF, https://datatracker.ietf.org/doc/html/rfc3550

[23] Jansen, B., et al., "Performance Evaluation of WebRTC-based Video Conferencing," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 56-61, 2015.