Technical Note



# A Self-Stabilizing Method for Dynamically Redundant Autonomously Distributed Real-Time Video

Daisuke Sugisawa *

Posted Date: 16 October 2025

doi: 10.20944/preprints202510.1226.v1

Keywords: WebRTC; SFU; autonomous distributed topology; FSM; HALT metric; real-time streaming; scalability

 doi:10.20944/preprints202510.1226.v1



*Technical Note*

# A Self-Stabilizing Method for Dynamically Redundant Autonomously Distributed Real-Time Video

**Daisuke Sugisawa**

Xander, LLC. Shibuya, Tokyo, Japan; daisuke.sugisawa.xander@gmail.com

**Abstract**

This document is a **Technical Note** that reports on the design and proof-of-concept of a self-distributed Selective Forwarding Unit (SFU) architecture for real-time video streaming. Unlike a full research article, this note emphasizes the **practical motivation, implementation details, and observed behaviors** of the system. The central idea is to address the limitations of traditional prediction-based scaling by introducing a **finite state machine (FSM)-based self-stabilizing method** that enables autonomous scale-out and scale-in of distributed SFUs under dynamic traffic conditions.

**Keywords:** WebRTC; SFU; autonomous distributed topology; FSM; HALT metric; real-time streaming; scalability

---

## 1. Introduction

### *1.1. Background*

Scaling SFU-based real-time video streaming is challenging due to unpredictable and sudden spikes in user demand. Conventional centralized scaling approaches often depend on traffic prediction and controller-driven resource allocation, which fail to react quickly enough during extreme fluctuations.

### *1.2. Gap*

Prediction-based scaling cannot handle **instantaneous traffic surges**, leading to bottlenecks, instability, and degraded user experience.

### *1.3. Scope*

This Technical Note introduces a **self-distributed FSM-based SFU architecture** that autonomously handles replication and redundancy. It outlines the **design principles, FSM protocols, and practical implementation details**, with observations from proof-of-concept testing.

## 2. Design and Method

### *2.1. Self-Distributed Topology*

The architecture eliminates central bottlenecks by enabling each SFU node to participate in an **autonomous distribution topology**. Nodes coordinate locally rather than relying on a centralized orchestrator.

As shown in Figure 1, the topology organizes nodes into a decentralized structure where replication flows outward from load hotspots, ensuring redundancy without centralized coordination.

 doi:10.20944/preprints202510.1226.v1



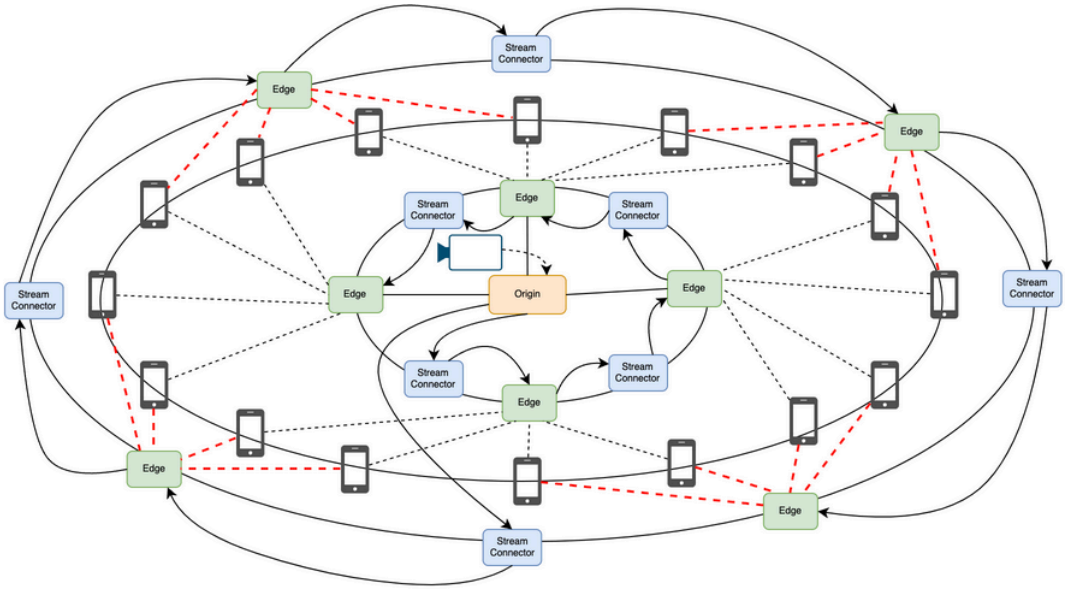


**Figure 1.** Cooperative autonomously distributed topology.

### *2.2. FSM-Based Protocols*

Scaling decisions are governed by a **finite state machine (FSM)** embedded within each node. FSM transitions define how nodes replicate under load and withdraw when demand decreases.

Figures 2 and 3 depicts the FSM transition model used for autonomous scale-out and scale-in.

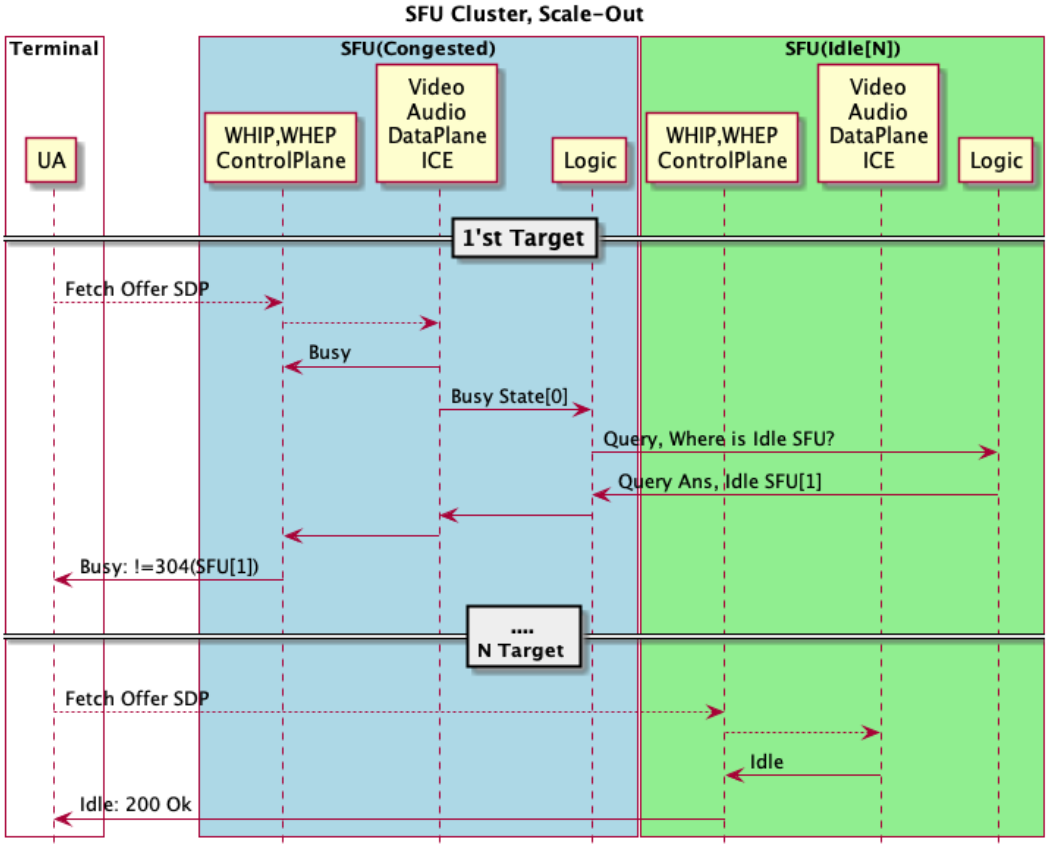


**Figure 2.** Scale-Out Protocol

 doi:10.20944/preprints202510.1226.v1



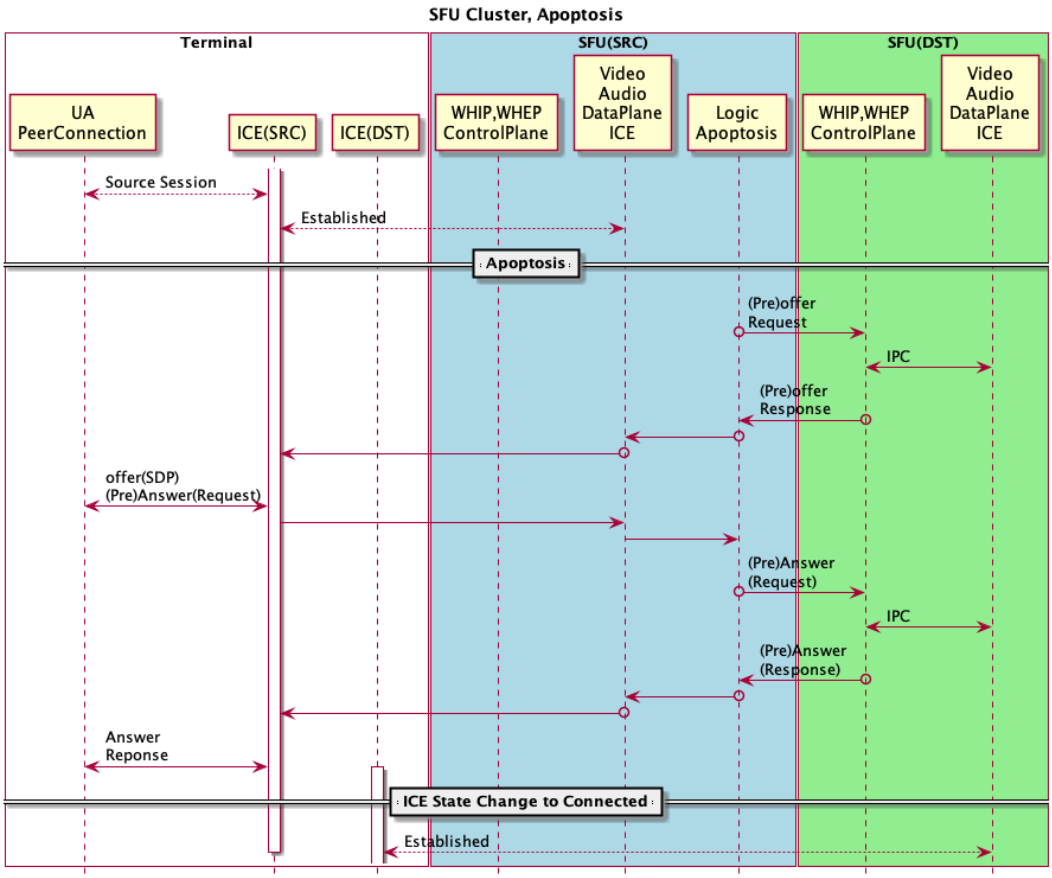


**Figure 3.** Scale-In Protocol

### *2.3. HALT Metric*

The **HALT metric** is introduced to estimate node capacity and trigger FSM transitions. HALT combines latency and throughput into a simple heuristic for load estimation.

## 3. Implementation

### *3.1. Frameworks and Environment*

The proof-of-concept implementation was built using

- **libdatachannel** for WebRTC signaling and media transport. [1]
- **MySQL BlackHole engine** for lightweight statistics handling.
- **MySQL Client Compatible** trilogy. [2]
- **Google Cloud Platform (GCP)** for deployment and distributed testing. Signaling([3])

### *3.2. Replication and Statistics Reporting*

Each node performs lightweight replication and periodically exchanges minimal statistics with peers. This minimizes coordination overhead and avoids central bottlenecks.

Figure 4 shows the replication flow and peer-to-peer exchange of statistics.

[1] https://github.com/paullouisageneau/libdatachannel
[2] https://github.com/trilogy-libraries/trilogy
[3] https://cloud.google.com/functions?hl=ja

 doi:10.20944/preprints202510.1226.v1



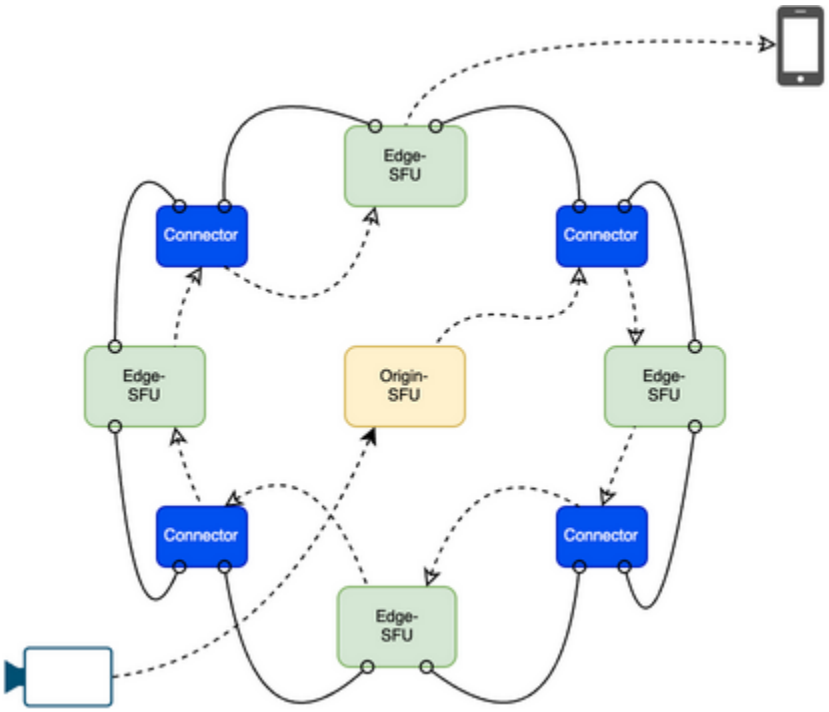


**Figure 4.** Neighbor-closed topology

## 4. Evaluation / Observations

### *4.1. RTT Measurements*

Table 1 presents round-trip time (RTT) measurements, demonstrating the effects of different edge node placements.

**Table 1.** measured RTT Tokyo - Region

| | |
|---|---|
| `EnterPrise LAN Tokyo-Tokyo` | 4.3 ms |
| `EnterPrise LAN Tokyo-Mumbai` | 135.2 ms |
| `EnterPrise Wifi Tokyo-Tokyo` | 7.8 ms |
| `EnterPrise Wifi Tokyo-Mumbai` | 138.5 ms |
| `Mobile Docomo Tokyo-Tokyo` | 43.0 ms |
| `Mobile Softbank Tokyo-Tokyo` | 43.0 ms |
| `Mobile Softbank Tokyo-Mumbai` | 180.7 ms |
| `Mobile AU Tokyo-Tokyo` | 43.0 ms |
| `Mobile AU Tokyo-Mumbai` | 176.2 ms |

### *4.2. Effects of Edge Placement (Origin, Proxy)*

Figure 5 shows the observed impact of edge node distribution on latency and stability. Placement closer to clients reduced RTT variance and improved resilience.

 doi:10.20944/preprints202510.1226.v1



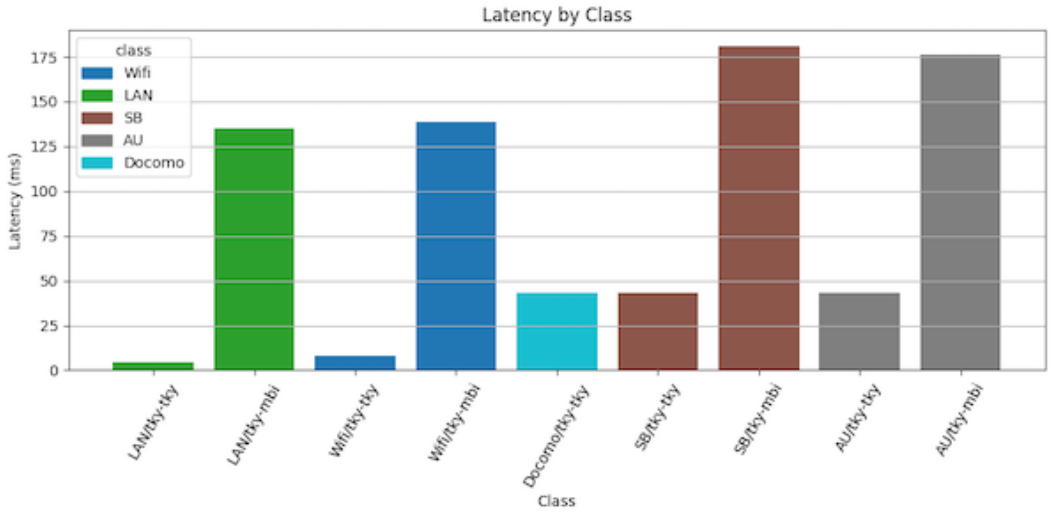


**Figure 5.** Inter-Region RTT

### *4.3. Scope of Results*

The evaluation presented in this Technical Note is limited to **proof-of-concept observations** rather than a full-scale statistical study. Specifically:

- **Round-trip time (RTT) measurements** were conducted in a limited set of deployment scenarios and summarized in Table 1.
- **Effects of edge placement** were observed in small-scale experiments, as illustrated in Figure 5
- The focus is on **highlighting qualitative behaviors** such as reduced RTT variance and improved stability when edge nodes are closer to clients.

A large-scale evaluation of **Quality of Experience (QoE)** across diverse network conditions and extended time frames has not yet been performed. Future work is required to validate these findings under production-scale workloads.

## 5. Discussion

### *5.1. Practical Implications*

- **Elimination of central bottlenecks** by relying on self-distributed coordination..
- **Lightweight statistics exchange** to keep overhead minimal.
- **Resilient operation** under unpredictable and dynamic traffic surges.

### *5.2. Limitations*

- Large-scale **Quality of Experience (QoE) evaluation** remains untested.
- Additional research is required to confirm stability at production scale.

### *5.3. Distributed vs. Autonomously Distributed*

"Distributed" $\neq$ "Autonomously Distributed."

- Distributed systems may still rely on controllers, coordinators, or managers.
- This creates bottlenecks:
  - Oversized statistics collection systems
  - High reporting costs

Our design eliminates central roles and achieves true self-stabilization.

## 6. Conclusions

This technical note introduced a self-stabilizing approach for dynamically redundant, autonomously distributed SFU clusters. By employing FSM-based cooperation and HALT-driven scaling

 doi:10.20944/preprints202510.1226.v1



decisions, the system avoids centralized bottlenecks and maintains fault tolerance through primary and redundant paths. A lightweight statistics design reduces cache interference and reporting overhead, enabling SFUs to focus on packet processing.

Our proof-of-concept deployments and RTT measurements highlight the importance of edge placement and distributed decision-making in practical cloud environments. In addition, the proposed architecture exhibits key self- properties* [13]:

### *6.1. Self-* Properties*

- Self-configuration: Edge-SFUs autonomously replicate and join clusters.
- Self-organization: routing and topology adapt dynamically to load and failures.
- Self-healing: nodes recover from crashes with seamless failover, Figure 15
- Self-management/optimization: HALT-driven adaptive policies (e.g., DSCP prioritization) sustain efficiency.

The contribution of this note is to provide architectural insights and implementation guidelines, rather than a full-scale evaluation, to support further development of scalable, cloud-native real-time video delivery.

## 7. Latency and Connection Stability

High RTTs (100–200 ms) between clients and distant Edge-SFUs cause ICE/STUN/TURN failures. Measured RTTs in Table 1 show severe degradation under mobile networks. To mitigate this, ICE timeouts must be tuned (e.g., iceGatheringTimeout, iceConnectionTimeout).

## 8. Related Work

**Table 2.** Related Work.

| | |
|---|---|
| Gossip [1]<br>Quorum Consensus<br>Figure 6 | Reduces monitoring cost, but only at the observation layer, not at control. |
| ENTS [2]<br>(Edge Task Scheduling)<br>Figure 7, Table 3 | Similar use case (video analytics) but focuses on centralized scheduling, not streaming. |
| DMSA [3]<br>Decentralized Micro-services<br>Table 4 | Decentralizes monitoring but retains centralized discovery queries. |
| P2P [12]<br>Unstructured/Structured<br>Figures 10 and 11 | Introduces hierarchical overlays with super-peers, which reintroduce centralization |
| Kademlia-based<br>WebRTC, [6]<br>Figure 8, Table 5 | Efficient DHT, but requires global structure and is redundant for cloud-native environments (regions/zones already managed). |
| SDN-based Scalable conferencing, [5] | Offloads SFU logic into P4 switches, skipping encryption/decryption, unsuitable for cloud cost-optimization |
| Enel<br>Graph Propagation Scaling<br>[7] | Uses graph propagation with coordinators; unlike our fully autonomous FSM-based approach. |
| US20210218681A1<br>Flash Crowd Management<br>In Real-Time Streaming<br>[11]<br>Table 6 | Central servers predict and handle spikes, scaling at the cluster level. In contrast, our method uses HALT-driven autonomous Edge replication. |
| AWS Chime<br>Media Pipelines<br>[9] | high abstraction and developer ease, but fully centralized, consuming extensive cloud resources. |
| Jitsi Conference<br>Focus<br>[10] | similarly centralized, constrained by heavy backend control. |

 doi:10.20944/preprints202510.1226.v1



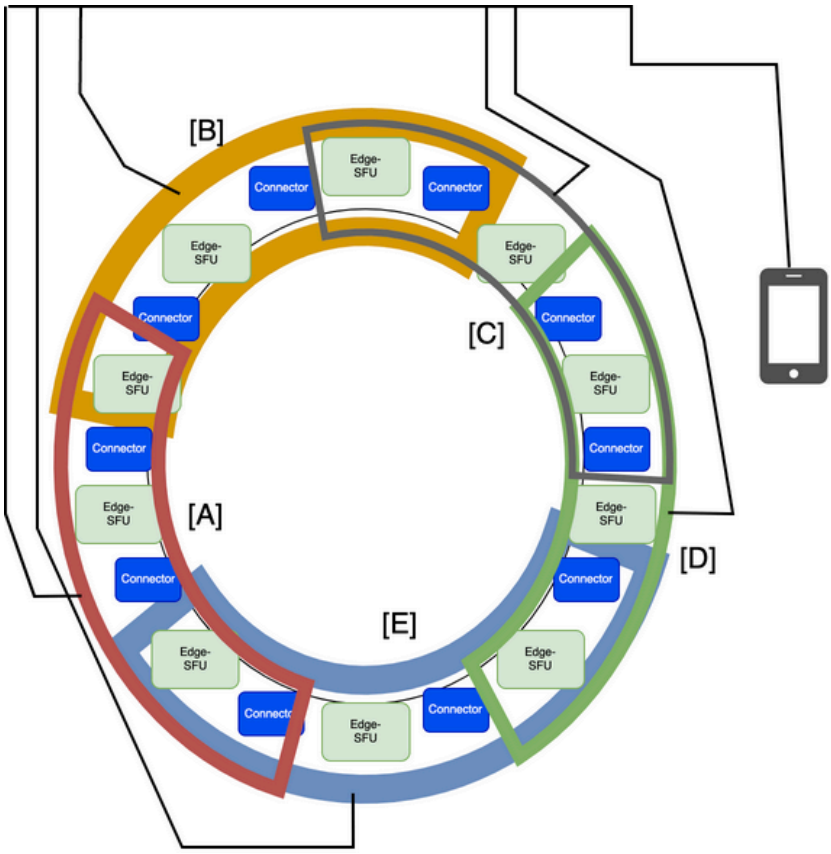


**Figure 6.** Self Distributed.

**Table 3.** ENTS , This Architecture.

| Item | ENTS | This Proposal |
|---|---|---|
| `Use Case` | Video Analysis: Data Stream Task | Video Delivery: WebRTC |
| `Scheduling` | Online Algorithm+Global Optimization | Scale-in/Scale-out via Local Decisions |

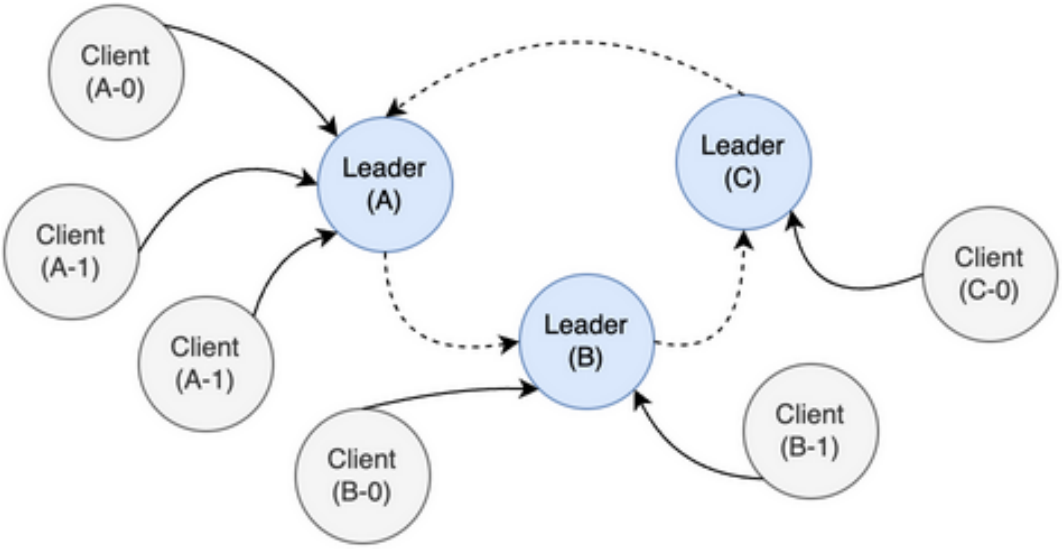


**Figure 7.** Semi-Intensive Conventional Monitoring.

 doi:10.20944/preprints202510.1226.v1



**Table 4.** DMSA vs This Architecture.

| Item | DMSA | This Proposal |
|---|---|---|
| Decentralization Depth | Depends on central list, API router nature | No central list needed,self-forming topology |
| Limit, Risk | Bottlenecks in reflecting changes to overall definition, redefinition of centralization | Minimal scale-in/out load |

**Table 5.** Kademlia , FSM/GCP.

| Item | Kademlia | This Proposal |
|---|---|---|
| `Overview` | Node search using XOR distance between node IDs | Selects optimal node using only local custom metrics per region/zone VM |
| `Placement` | Irregular and skewed node placement | Nodes structured logically and physically by region, zone, etc. |
| `Search` | O(log n) based on XOR distance | Real-time determination from RTT and HALT values of neighboring nodes |
| `FSM` | DHT distributed management | Nodes/devices manage only neighbor statistics |
| `Failure` | DHT redundancy | Nodes/devices manage only nearby statistics |
| `Delay` | k-bucket maintenance, DHT refresh | Node-specific local metric updates |
| Adaptability | Unspecified number of P2P nodes | GCP / Cloud, etc. Simple mechanism adapted to structured infrastructure |

**Table 6.** IS20210218681A1 vs This Architecture.

| Item | US20210218681A1 | This Proposal |
|---|---|---|
| Traffic Spike Prediction | Centralized, Central Server + Management System | Strongly Autonomous Distributed: Each Node Independently Decides, Acts |
| Scaling Granularity | Scale-out at cluster level | Self-replication at local/neighboring Edge level |

 doi:10.20944/preprints202510.1226.v1



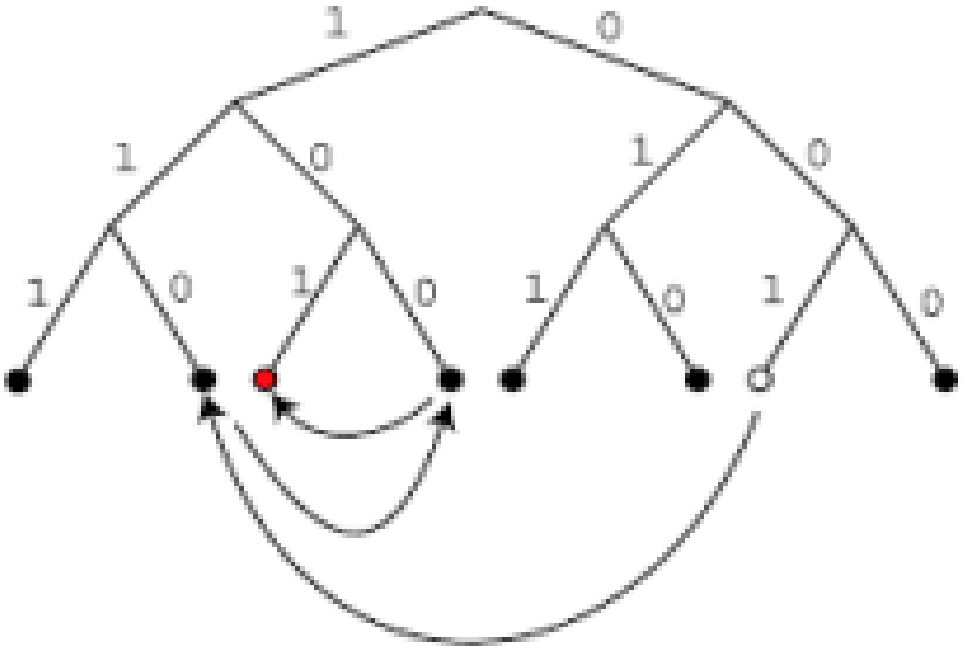


**Figure 8.** Kademlia XOR Metric.

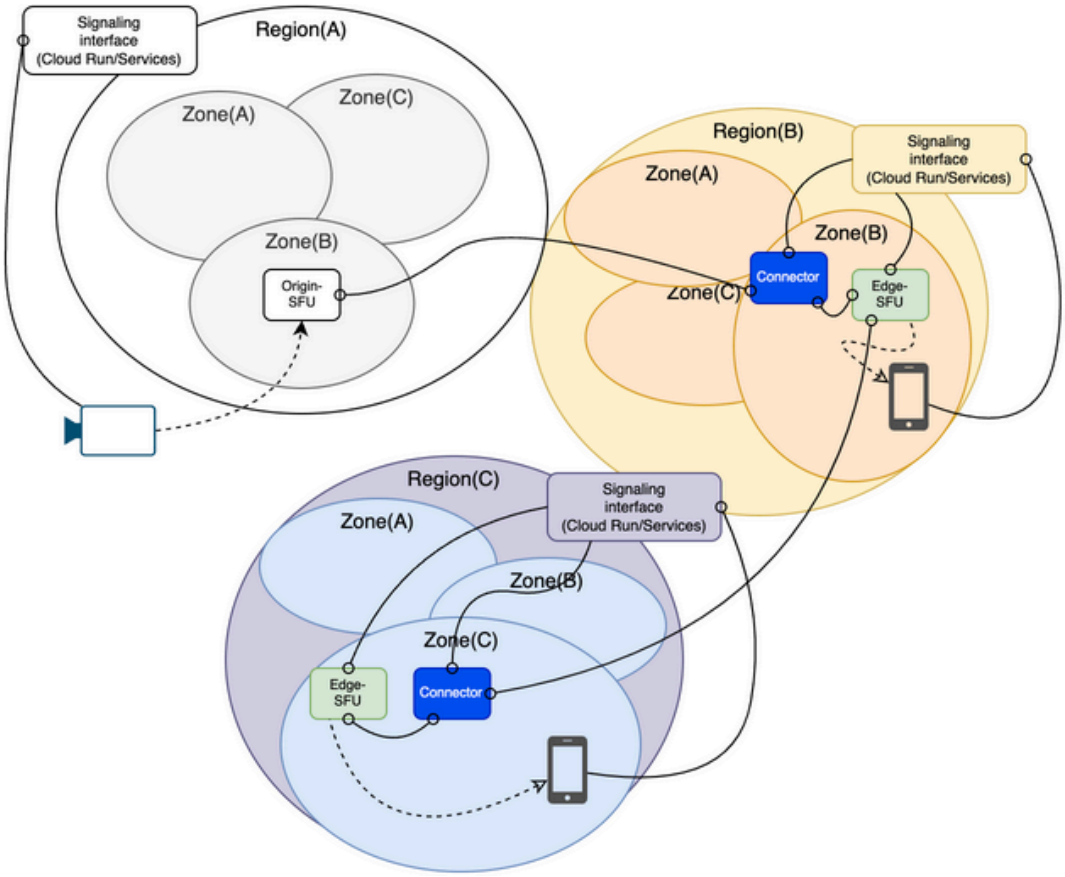


**Figure 9.** Cloud Native Edge-SFU Selection.

 doi:10.20944/preprints202510.1226.v1



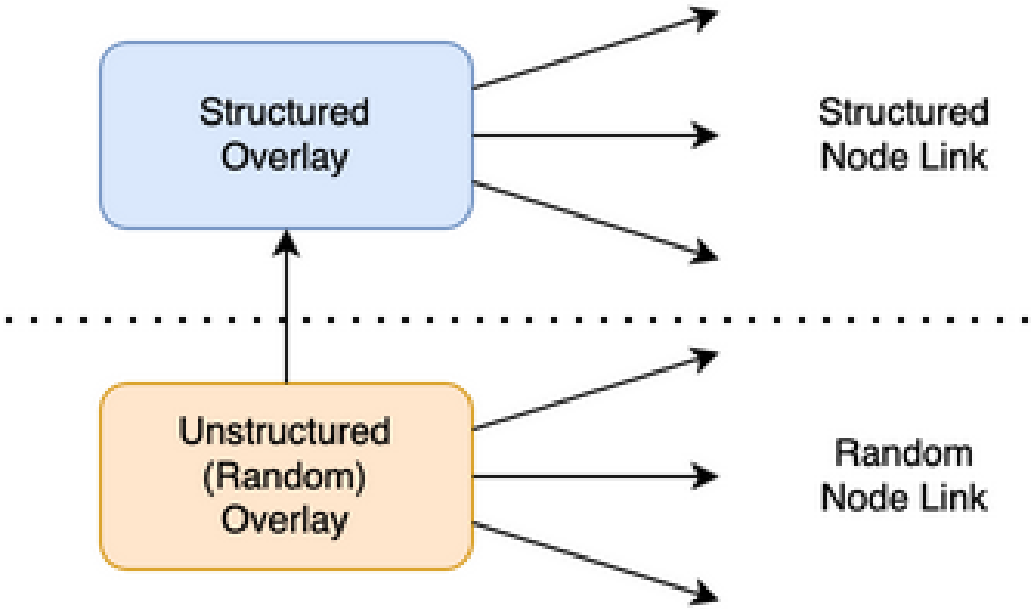


**Figure 10.** Unstructured Structured 2-Layer Topology.

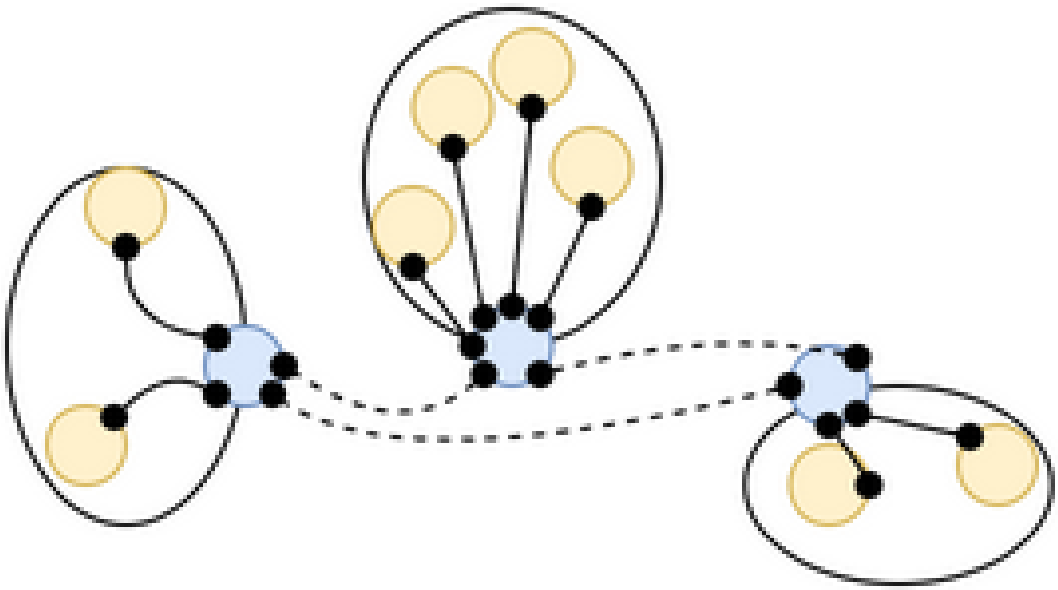

**Figure 11.** Super Peer Networ.k

Listing 1: Naming system.

```
{region}-{module-type}-{uuid}-{suffix}

# module-type = origin, edge, connector
# suffix = user-hash
```

Shows naming and query systems used to select Edge-SFUs dynamically with HALT-based filtering.

 doi:10.20944/preprints202510.1226.v1



## 9. Implementation

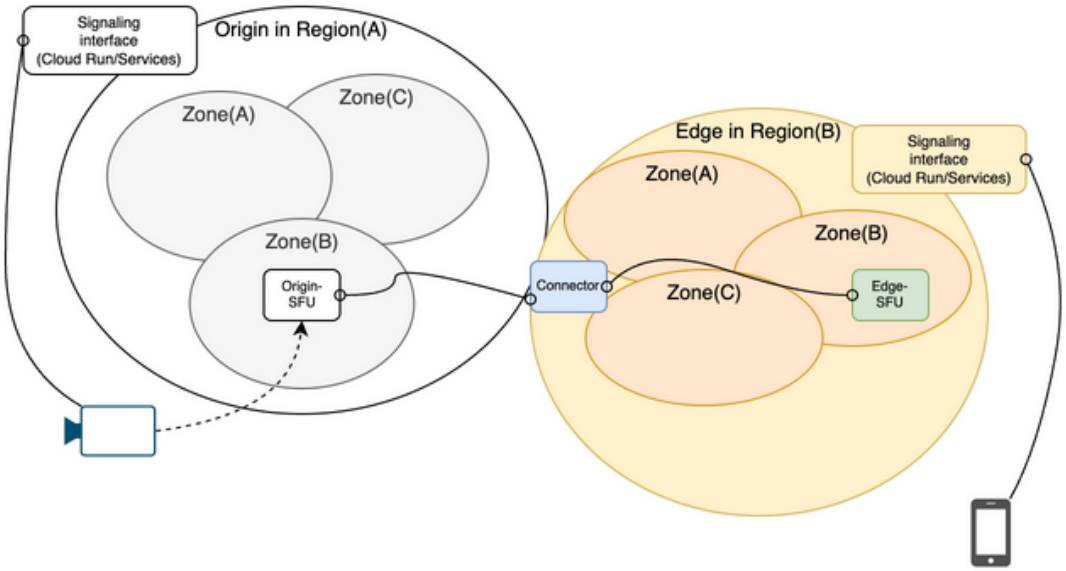


**Figure 12.** Other Region.

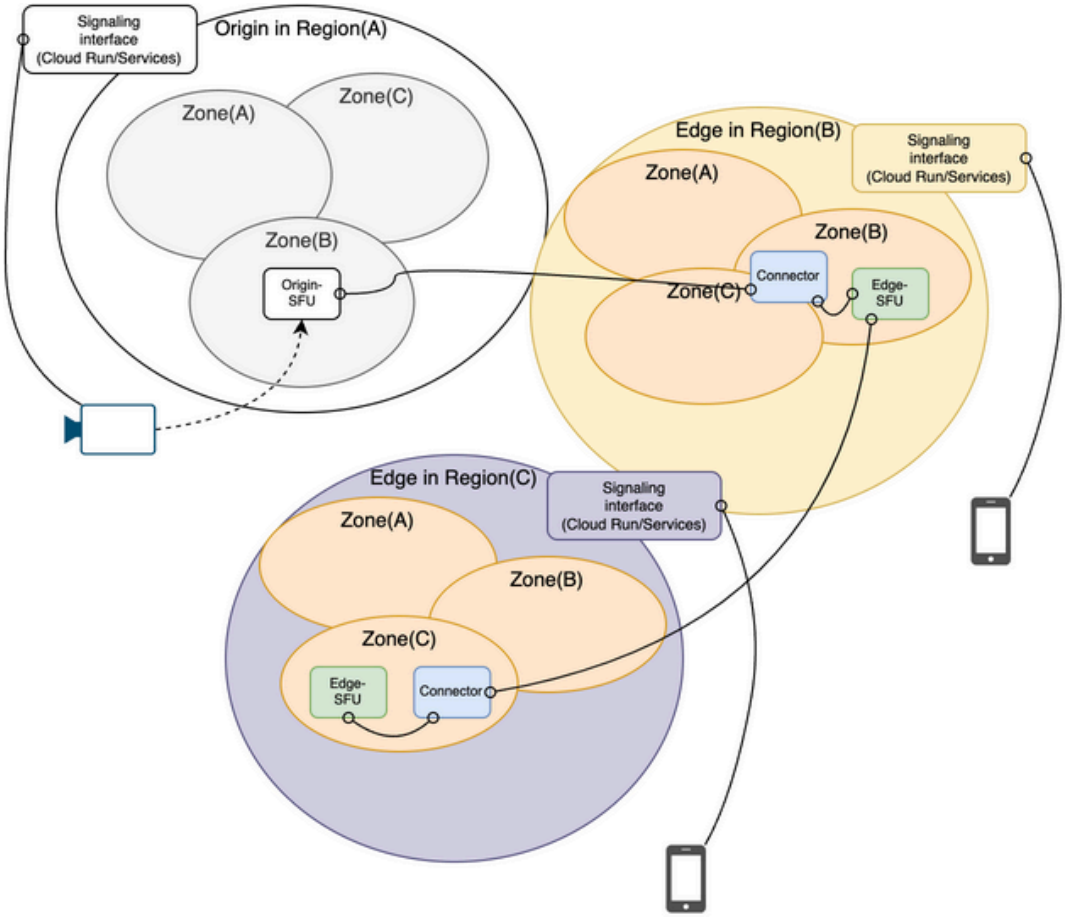


**Figure 13.** Multi Region.

 doi:10.20944/preprints202510.1226.v1



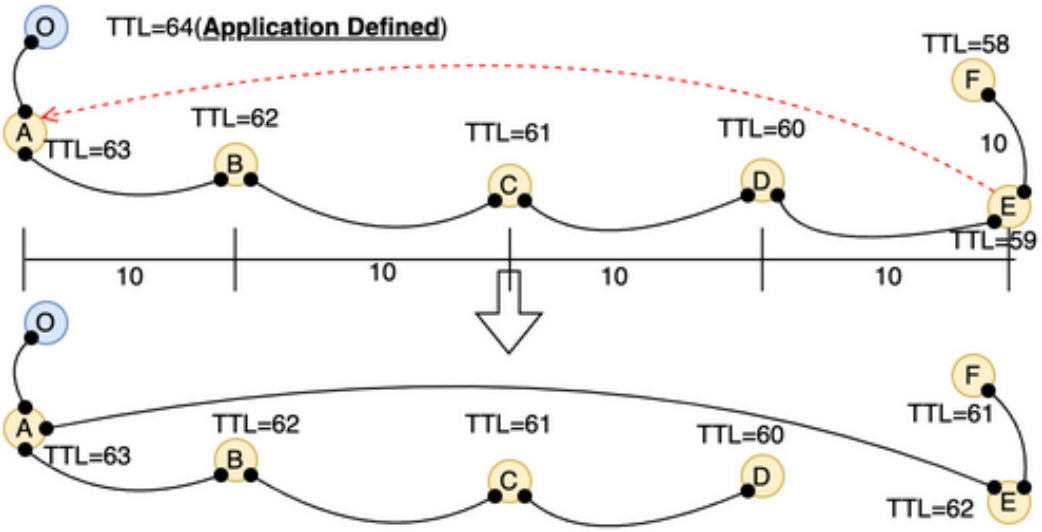


**Figure 14.** Optimization and Reconfiguration of Communication Paths Based on Local Judgment.

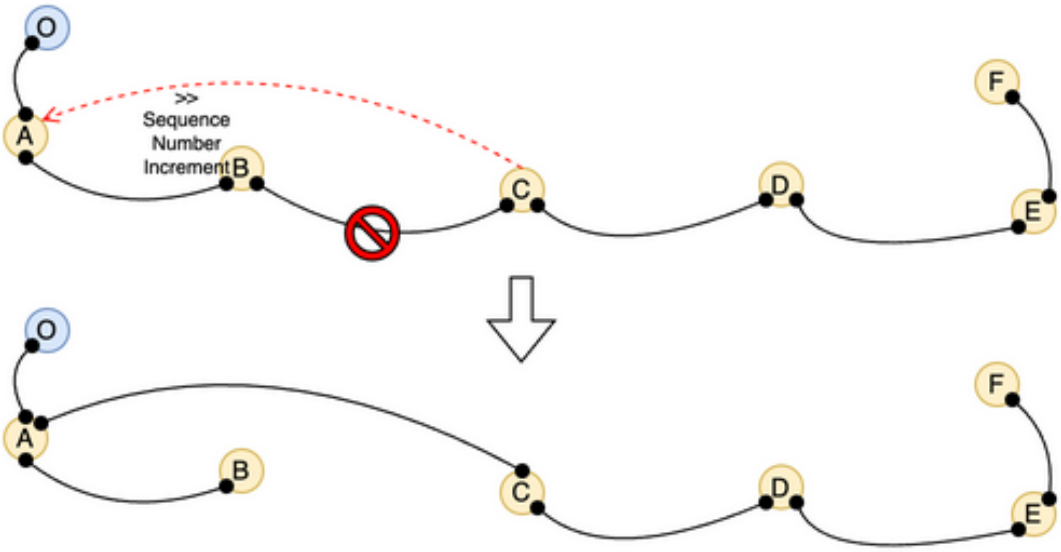


**Figure 15.** Auto Failover based on local determination.

## 10. Appendix

### *10.1. Code Listings*

Listing 2: Neighbor Edge Node Selection Algorithm.

```
def neighbor_edge(client_location, edge_stats):
    candidate_edges = []
    for edge in edge_stats:
    if is_in_region(edge.location, client_location):
        halt_per_session = edge.halt_value / edge.session_count
        rtt = estimate_rtt(client_location, edge.location)
        candidate_edges.append((edge, halt_per_session, rtt))

    if not candidate_edges:
        new_edge = create_edge_instance_neighbor(client_location)
        parent_edge = find_neighbor_existing_edge(client_location)
        connect_edges_via_connector(parent_edge, new_edge)
        return new_edge
```

 doi:10.20944/preprints202510.1226.v1



```
    candidate_edges.sort(key=lambda x: (-x[1], x[2]))
    return candidate_edges[0][0]
```

Listing 3: Statistics Report

```
INSERT INTO stats
    (`session_id`, `event`, `option`, `created_at`)
VALUES
    ({session_id}, {event}, {option}, {created_at}),
    ({session_id}, {event}, {option}, {created_at}),
    ....( x BULK SIZE )
    ({session_id}, {event}, {option}, {created_at}),
    ;
```

Listing 4: CPU-Core-Mask.

```
cpu_set_t cpumask;
CPU_ZERO(&cpumask);
CPU_SET(cpuid, &cpumask);
//
if (cpuid >= 0){
    if (pthread_setaffinity_np(pthread_self(), sizeof(cpumask), &cpumask) != 0) {
        PANIC("Unable to set affinity: %s", strerror(errno));
        return(ERR);
    }
    return(OK);
}
```

Listing 5: Sample RTT measurement.

```
peer_connection = new RTCPeerConnection({iceServers: [], iceTransportPolicy: 'all'});
peer_connection.onicegatheringstatechange = (state) => {
    if (peer_connection.iceGatheringState === 'complete') {
        const ans = peer_connection.localDescription;
        answer({"type": ans.type, sdp: ans.sdp, link: ''});
    }
}
peer_connection.ondatachannel = (event) => {
    data_channel = event.channel;
    data_channel.onopen = () => {
        data_channel.send(JSON.stringify({"t0": nowJST()}));
    }
    data_channel.onmessage = (event) => {
        var jsn = JSON.parse(event.data);
        if ('t5' in jsn) {
            jsn['t6'] = nowJST();
            topology.value = JSON.stringify(jsn, null, 2);
        } else if ('t3' in jsn) {
            jsn['t4'] = nowJST();
            data_channel.send(JSON.stringify(jsn));
        } else if ('t1' in jsn) {
            jsn['t2'] = nowJST();
            data_channel.send(JSON.stringify(jsn));
        }
    };
};
peer_connection.setRemoteDescription(offer_json);
peer_connection.createAnswer().then(answer => {
    peer_connection.setLocalDescription(answer);
});
```

 doi:10.20944/preprints202510.1226.v1



*10.2. Cache Pollution Avoidance*

Edge-SFUs must prioritize packet forwarding over auxiliary tasks. Statistics collection, logging, or control messages can cause cache pollution and degrade forwarding throughput. Our implementation isolates forwarding-critical threads from reporting tasks, ensuring cache locality is preserved for real-time processing.

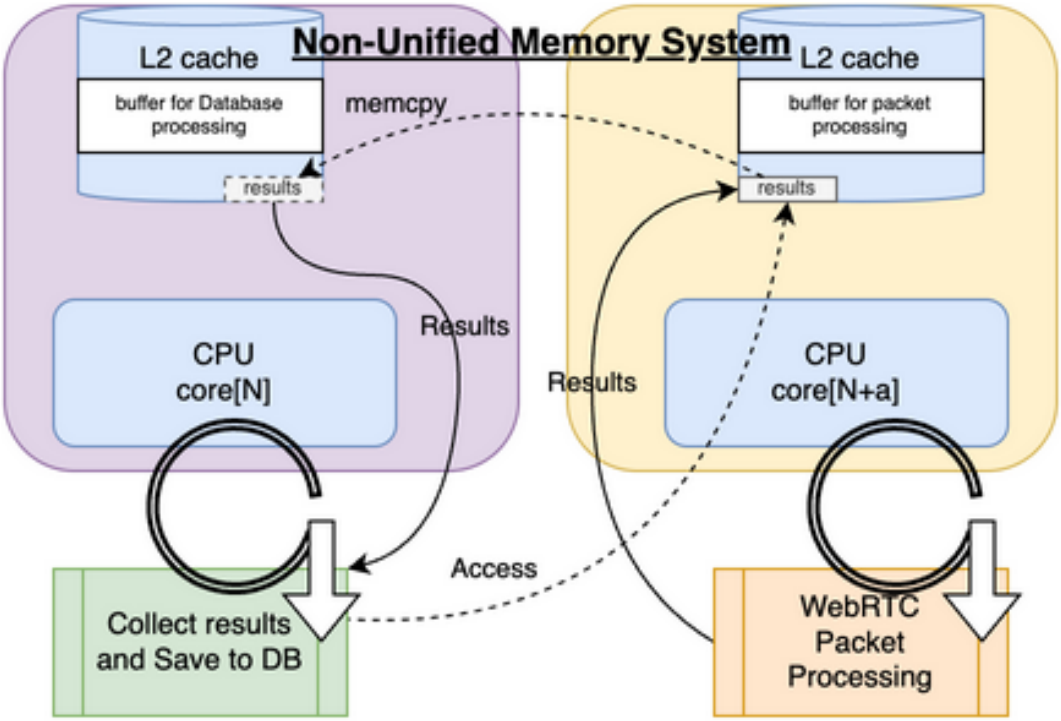


**Figure 16.** Parallel Processin on Many-Core CPUs.

*10.3. Origin-Edge Connectivity*

- Origin-SFUs and Edge-SFUs connect through NAT traversal connector modules.
- Clients search for nearby Edge-SFUs; if none exist, a new Edge-SFU (B) is spawned near the client.
- The new Edge-SFU automatically connects to an existing Edge-SFU (A) and integrates into the topology.

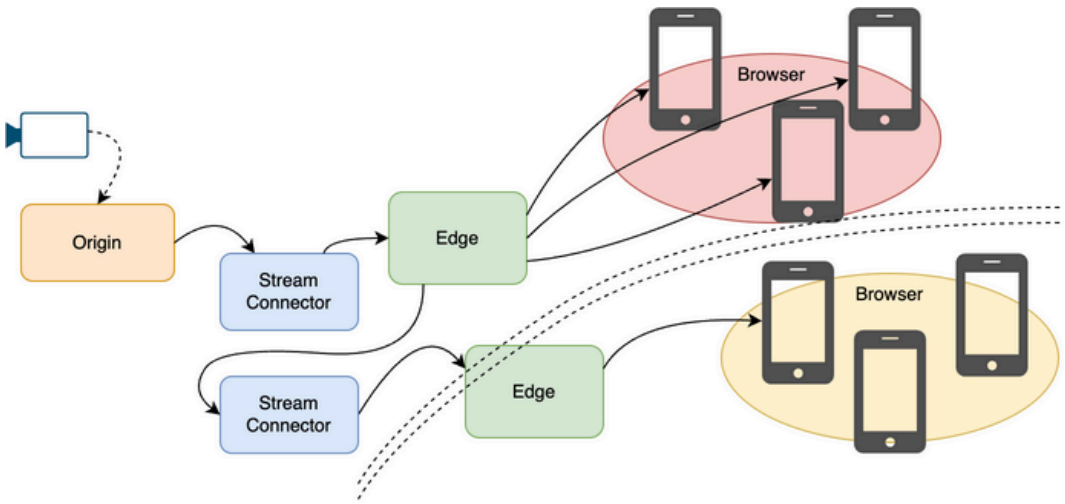


**Figure 17.** Origin Edge Connectivity.

*10.4. Improving Accuracy of Statistics – HALT Value*

Traditional metrics (session count, CPU usage, bandwidth) fail to measure actual spare capacity.

 doi:10.20944/preprints202510.1226.v1



- We introduce HALT (Hardware Available Load Threshold) as a real metric.
- HALT measures residual processing capacity after each loop, incorporating CPU idle, I/O waits, throughput, etc.
- This enables more reliable distributed decision-making

### *10.5. Oversized Statistics Systems*

Traditional systems often suffer from bloated statistics pipelines (HTTP APIs, external analytics). These consume CPU, memory, and network resources, ironically destabilizing the system. Our approach keeps statistics simple (local MySQL + replication), enabling SFUs to focus on packet processing.

### *10.6. Handover*

ICE/TURN negotiation introduces latency, Buffers synchronize 2–30 frames between source and target Edge-SFUs

**Table 7.** Handover Entities.

| Item | Overview |
|---|---|
| `Mobile Device` | UE(User Equipment) |
| `HandOver Origin` | Currently Connected Edge-SFU |
| `HandOver Dest` | Target Edge-SFU |
| `Connector` | Stream-Connector during handover |
| `Buffer` | Media(video) buffering during handover |

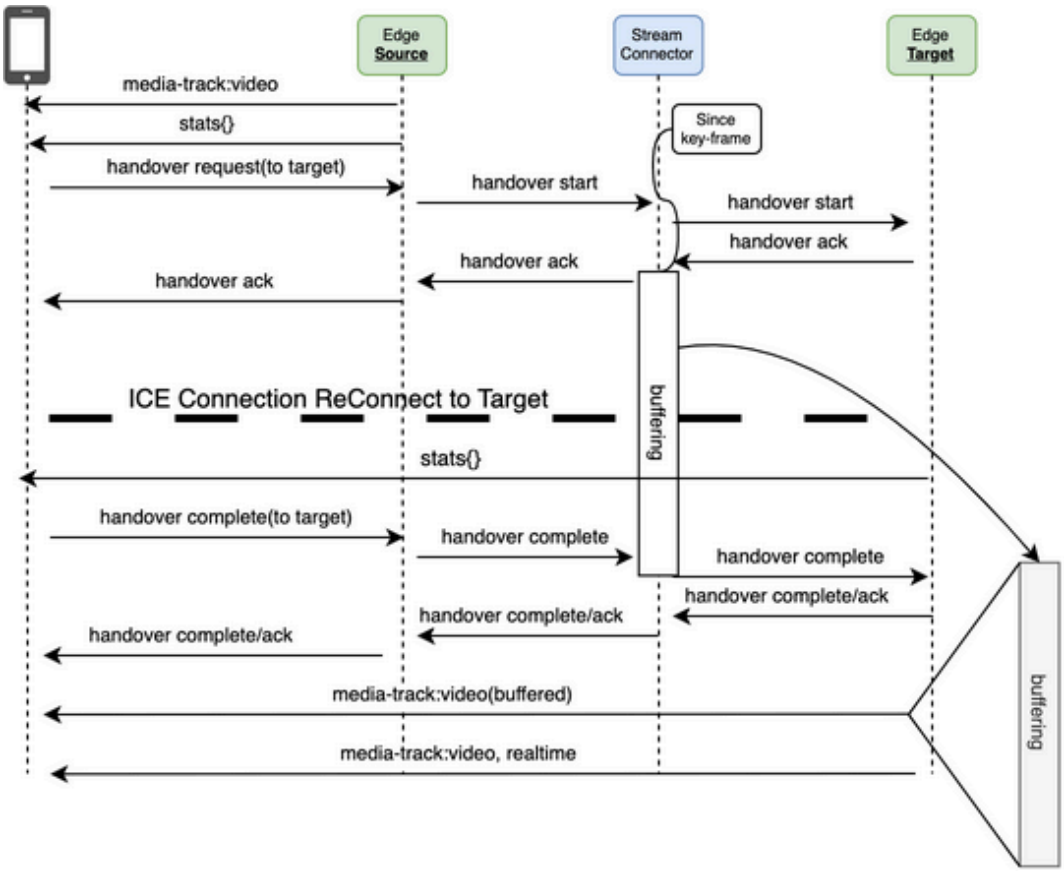


**Figure 18.** Handover Sequence

 doi:10.20944/preprints202510.1226.v1



## 11. AI Disclosure

The author used a large language model ChatGPT, OpenAI to assist in proofreading, grammar checking, and improving the clarity of expressions. The author reviewed and is responsible for the final content.

**Author Contributions:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing-Original Draft, Writing-Review and Editing, Visualization, Supervision, Project administration: D.Sugisawa.

**Funding:** This research received no external funding.

**Conflicts of Interest:** The author declares no conflicts of interest.

## References

1. SHASHIKANT ILAGER. A Decentralized and Self-Adaptive Approach for Monitoring Volatile Edge Environments. http://arxiv.org/abs/2405.07806v1
2. Mingjin Zhang. ENTS: An Edge-native Task Scheduling System for Collaborative Edge Computing. http://arxiv.org/abs/2210.07842v1
3. Yuang Chen. DMSA: A Decentralized Microservice Architecture for Edge Networks. https://arxiv.org/pdf/2501.00883v1
4. Chmieliauskas. Evaluation of Uplink Video Streaming QoE in 4G and 5G Cellular Networks Using Real-World Measurements. https://doi.org/10.1109/ACCESS.2025.3554340
5. Oliver Michel. Scalable Video Conferencing Using SDN Principles. https://arxiv.org/pdf/2503.11649
6. Ryle Zhou. Decentralized WebRTC P2P Network Using Kademlia. https://arxiv.org/abs/2206.07685
7. Dominik Scheinert. Enel:Context-Aware Dynamic Scaling of Distributed Dataflow Jobs using Graph Propagation. https://arxiv.org/pdf/2108.12211
8. Shaher Daoud and Yanzhen Qu. A COMPREHENSIVE STUDY OF DSCP MARKINGS' IMPACT ON VOIP QOS IN HFC NETWORKS. https://aircconline.com/ijcnc/V11N5/11519cnc01.pdf
9. AWS Chime Media Pipelines. https://docs.aws.amazon.com/chime-sdk/latest/dg/media-pipelines.html
10. jitsi. Jitsi Conference Focus. https://github.com/jitsi/jicofo
11. US20210218681A1. Flash crowd management in real-time streaming. https://patents.google.com/patent/US20210218681A1/en
12. Tao GU. A Hierarchical Semantic Overlay for P2P Search. https://arxiv.org/pdf/2003.05001
13. Babaoglu. Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations. https://doi.org/10.1007/b136551