

# Decentralized Load Balancing and Adaptive Scaling using Peer-to-Peer Topologies and Predictive Machine Learning

1<sup>st</sup> Ninad Maadhavi

*Department of Computer Science and Engineering*  
*Sardar Patel Institute of Technology*  
Mumbai, India  
ninad.maadhavi22@spit.ac.in

2<sup>nd</sup> Nikhil Chaudhari

*Department of Computer Science and Engineering*  
*Sardar Patel Institute of Technology*  
Mumbai, India  
nikhil.chaudhari23@spit.ac.in

3<sup>rd</sup> Maaz Malik

*Department of Computer Science and Engineering*  
*Sardar Patel Institute of Technology*  
Mumbai, India  
maaz.malik22@spit.ac.in

4<sup>th</sup> Ruchir Khare

*Department of Computer Science and Engineering*  
*Sardar Patel Institute of Technology*  
Mumbai, India  
ruchir.khare22@spit.ac.in

**Abstract**—Horizontal scaling forms the backbone of modern cloud infrastructure, but traditional load balancing systems relying on a single master proxy introduce a critical single point of failure and performance bottleneck. This paper proposes a novel framework for Peer-to-Peer (P2P) Load Balancing coupled with Machine Learning (ML) based Adaptive Scaling to enhance service resilience and efficiency. Our P2P approach replaces the centralized proxy by distributing request routing responsibility among service instances (peers), eliminating the single point of failure. A key challenge in P2P systems is managing the excessive overhead of full connectivity; thus, we investigate how limiting peer-to-peer connection valencies affects network topology, specifically noting the tendency for low valencies to form inefficient, chain-like structures. Additionally, we integrate an ML-based CPU utilization forecasting model to predict load spikes and trigger unit spawning preemptively, mitigating crash risks associated with reactive scaling. We analyze various P2P configurations based on valency limits and their impact on load distribution, networking overhead, and latency, demonstrating a highly resilient and performance-optimized solution for high-concurrency environments.

**Index Terms**—peer-to-peer, load balancing, horizontal scaling, adaptive scaling, machine learning, decentralized system, network topology, single point of failure

## I. INTRODUCTION

Modern web services and cloud-native applications must handle massive fluctuations in traffic, often reaching millions of concurrent requests. **Horizontal scaling**, the practice of adding more server instances to distribute load, is the fundamental strategy employed to manage this demand. However, the efficacy and resilience of a horizontally scaled system are critically dependent on the mechanism used to distribute incoming requests.

Current mainstream load balancing implementations predominantly rely on a **single master proxy** (or a set of redundant proxies). While effective for basic traffic distribution,

this architecture presents two major liabilities: first, the master proxy itself becomes a performance **bottleneck** as it must process metadata for all traffic; and second, the failure of this proxy constitutes a **single point of failure** (SPOF), which can instantly crash the entire service, regardless of the availability of backend instances.

To address these limitations, this paper explores the design and implementation of a **Decentralized Load Balancing System** using a **Peer-to-Peer (P2P)** communication model. In this model, each service instance acts as a peer and a miniature load balancer, responsible for routing requests to other available peers. This logical abstraction enhances resilience by inherently removing the SPOF. Our work moves beyond established P2P concepts by focusing on two core novelties crucial for real-world adoption:

- 1) **Optimized P2P Topology and Valency:** The computational and memory overhead (session maintenance, metadata, heartbeat checks) of maintaining connections in a P2P network necessitates limiting the number of connections per peer (valency). We analyze the trade-offs of low vs. high valencies and propose strategies for constructing efficient, sparse P2P topologies that maximize load balancing performance while minimizing overhead.
- 2) **Machine Learning-based Predictive Scaling:** The rule-based threshold approach to horizontal scaling is inherently reactive and often too slow to prevent system collapse during sudden, high-volume load spikes (e.g., during a major live event). We integrate a model for CPU utilization forecasting to enable **proactive spawning** of new units, ensuring the system capacity is scaled *before* the critical threshold is reached.

The remainder of this paper is organized as follows: Section II reviews related work in decentralized load balancing and predictive scaling. Section III details the proposed P2P methodology, including the topological constraints and the load distribution algorithm. Section IV describes the ML forecasting model and the adaptive scaling policy. Section V presents the experimental results and analysis of different valency configurations. Finally, Section VI concludes the paper and outlines future research directions.

## II. RELATED WORK

The challenge of creating highly available and scalable distributed systems has led to continuous innovation across several domains, including decentralized networking, load distribution algorithms, and resource management. Our work intersects three primary areas of research: **Decentralized Service Discovery and Load Balancing**, **Network Topology Optimization**, and **Predictive Auto-Scaling**.

### A. Decentralized Service Discovery and Load Balancing

Traditional server-side load balancing [3] (e.g., LVS, HAProxy, F5) relies on a clear separation of concerns, where dedicated infrastructure handles distribution. While robust, this architecture is inherently centralized, suffering from the aforementioned SPOF issue.

Early research into **peer-to-peer load balancing** focused primarily on Distributed Hash Tables (DHTs) like Chord [4] and Pastry [5] for decentralized key-value storage and routing. These systems utilize structured P2P overlays that ensure logarithmic route search times. While effective for data lookup, applying these structures directly to stateless HTTP request distribution requires complex protocol mapping.

More recent work has explored client-side load balancing, where the client is responsible for selecting the server. This model, often used in microservices architectures with tools like Netflix's Eureka or Consul, shifts the burden but still requires a centralized service registry for discovery. Our research goes further by making the **service instance itself** responsible for *re-routing* an overloaded request to another peer, creating a truly decentralized, self-healing mesh topology. This concept is closely related to **gossip protocols** and epidemic-style service management [6], where state information (load, availability) is propagated through local peer-to-peer exchanges rather than a central authority.

### B. Network Topology Optimization and Valency Constraints

The performance of any P2P system is fundamentally tied to its underlying **overlay network topology**. A fully-meshed network (where every node connects to every other node) offers maximum resilience and minimal request latency but incurs  $O(N^2)$  connection overheads, rendering it impractical for large systems.

Research into **sparse graph construction** seeks to find topologies that balance resilience (connectivity) and efficiency (low valency). Concepts like the **Small-World Network** model [7], which features high clustering (local connections) and

short path lengths (few long-range connections), are highly relevant. Your key observation regarding the tendency of low valencies ( $v = 3$  in your example) to form elongated, polymer-like chains instead of robust, root-like structures is a critical concern, as such a topology significantly increases the average path length for a request, leading to higher latency and reduced load balancing efficiency. This directly motivates our investigation into optimal valency configuration strategies, distinct from the generic  $k$ -regular graph models.

### C. Predictive Auto-Scaling

Traditional scaling policies are **reactive**, based on static thresholds (e.g., "spawn a new unit if CPU usage  $\geq$  80% for 5 minutes"). While simple, the time lag between the threshold being crossed and the new unit becoming ready often leads to cascading failures during flash crowds.

Modern solutions rely on **predictive auto-scaling** using time-series forecasting models, typically employing techniques like ARIMA, Prophet, or various Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks [8]. These models learn historical load patterns to forecast future resource demand. Our contribution lies in integrating this forecasting capability directly into the decentralized P2P framework, allowing peers to collectively or individually signal the need for new resources based on a global, ML-informed prediction, thereby transforming the system from a reactive to a **proactive and adaptive** infrastructure.

## III. METHODOLOGY

This section details the design of the decentralized networking layer and the algorithmic principles governing peer interactions. We analyze two distinct topological configurations to determine the optimal balance between connectivity and computational overhead.

### A. Network Topology Models

The efficacy of a peer-to-peer load balancing system is strictly bounded by its overlay topology. We examine the transition from a theoretical fully connected state to a practically constrained crystalline structure.

1) *Fully Connected Mesh Configuration*: In the initial baseline model, we consider a **Fully Connected Configuration**, often referred to as a Full Mesh. In this setup, every service instance (node) maintains an active TCP/IP connection with every other node in the cluster.

As illustrated in Fig. 1, this topology offers distinct theoretical advantages:

- **Hop Minimization**: The number of hops required to route a request from any node to any other node is strictly 1. This ensures minimal latency in request forwarding.
- **Global State Awareness**: Every peer has immediate access to the load status of the entire cluster, theoretically allowing for perfect load distribution decisions.

However, this configuration presents critical scalability disadvantages:

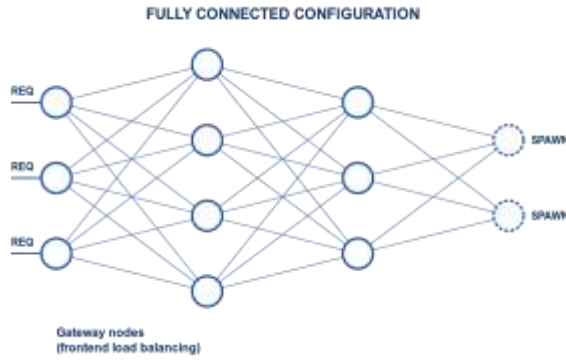


Fig. 1. Fully Connected Configuration: Gateway nodes (Front-end) distribute requests to a mesh where every node is aware of all other peers.

**Connection Overhead:** Every active connection entails a memory and CPU overhead for maintaining session state, keep-alive heartbeats, and metadata synchronization. In a cluster of size  $N$ , the number of connections grows as  $O(N^2)$ .

**Diminishing Returns:** As  $N$  increases, the resource cost of maintaining connections eventually outweighs the processing capacity of the node itself, leading to saturation before useful work can be performed. Consequently, it becomes imperative to limit the number of active neighbors per node.

2) *Sparse Crystalline Topology:* To mitigate the overhead of the fully connected mesh, we propose a **Crystalline Topology** based on a constrained growth model. Here, we introduce the concept of **Valency ( $V$ )**, defined as the maximum number of active connections a single peer maintains.

By limiting connections ( $V \ll N$ ), nodes form a sparse graph. However, the geometric structure of this graph is critical. We define a **Growth Parameter ( $G$ )** representing the set of initial gateway nodes.

a) *The Polymer Chain Phenomenon:* Our simulation results indicate a topological risk when the Valency limit is low relative to the growth rate ( $V \approx G$ ). Under these conditions, the network fails to branch outward effectively. Instead of forming a dense, "root-like" structure that facilitates rapid load diffusion, the graph exhibits a **"Polymer-like" behavior**.

In this polymer configuration:

- The network grows in long, linear chains.
- **High Network Diameter:** The average path length (hops) between nodes increases significantly.
- **Inefficient Balancing:** A request entering at one end of the "chain" may require multiple hops to reach an idle node at the other end, introducing latency that negates the benefits of load balancing.

To enforce a "Small-World" property and prevent chains, we implement a specific **Crystalline Growth Protocol** (Algorithm 1). This protocol forces new nodes to triangulate immediately upon instantiation.

By explicitly attempting to form  $V - 2$  additional connections, the new node  $C$  immediately reduces the local network diameter. Crucially, by reserving exactly **one slot**, we ensure that  $C$  remains capable of accepting a future child node,

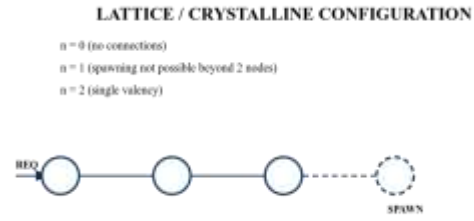


Fig. 2. Crystalline Topology Simulation: Visualizing the impact of growth parameters on network structure. Note the formation of chains when valency is constrained.

#### Algorithm 1 Crystalline Topology Growth Protocol

**Require:**  $Nodes_{initial}$  (Gateways),  $V$  (Max Valency)

```

1:  $Cluster \leftarrow Nodes_{initial}$ 
2: while System is Scaling (New Node Required) do
3:   Step 1: Parent Selection
4:    $P \leftarrow$  Select random node from  $Cluster$  where  $Degree(P) < V$ 
5:   Step 2: Spawning
6:    $C \leftarrow$  Spawn new Child Node
7:   Add Edge ( $P, C$ ) {Consumes 1 slot on  $P$  and  $C$ }
8:   Step 3: Triangulation (Reserving Slot)
9:    $Target \leftarrow V - 2$  {Total  $V - 1$  (Parent) - 1 (Reserved for future child)}
10:  for all  $n \in Cluster$  do
11:    if  $n \neq P$  and  $Degree(n) < V$  and  $Target > 0$  then
12:      Add Edge ( $C, n$ )
13:       $Target \leftarrow Target - 1$ 
14:    end if
15:  end for
16:   $Cluster \leftarrow Cluster \cup \{C\}$ 
17: end while

```

guaranteeing that the graph never reaches a "dead end" where growth is physically impossible.

#### IV. SYSTEM ARCHITECTURE

The proposed solution implements a layered architecture, decoupling the low-level networking complexities from the high-level load balancing logic. As illustrated in Fig. 3, the system is organized into four primary layers: the Application Layer (Decision Making), Protocol Layer, Messaging Layer, and the underlying Communication Layer.

##### A. Communication Layer

The foundation of our decentralized mesh is built upon the **WebRTC (Web Real-Time Communication)** standard, which facilitates low-latency, peer-to-peer data exchange.

1) *Active Signaling and ICE Framework:* Since peer nodes often reside behind Network Address Translators (NATs) and firewalls, direct connection establishment is non-trivial. We employ the **Interactive Connectivity Establishment (ICE)** framework.

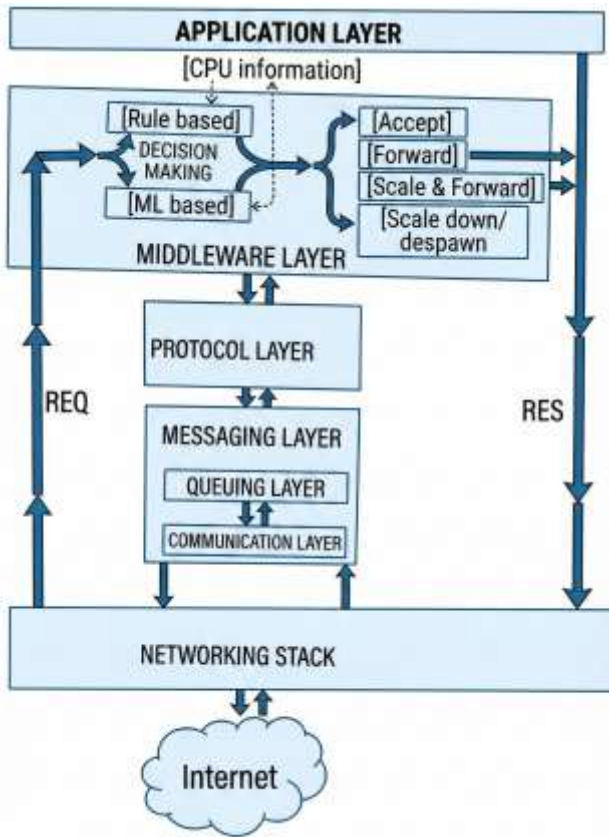


Fig. 3. Layered System Architecture: From the bottom-up Networking Stack to the top-level ML and Rule-based Decision Engine.

- **Signaling Phase:** Before a direct data channel opens, peers exchange session descriptions (SDP) via a lightweight signaling server (e.g., WebSocket). This is the only centralized component, used solely for the initial handshake, not for data routing.
- **ICE Servers:** Peers query STUN (Session Traversal Utilities for NAT) servers to discover their public IP addresses and ports. In restrictive network environments, TURN (Traversal Using Relays around NAT) servers act as fallback relays.

2) **NAT Traversal and "Hole Punching":** A critical requirement for a true peer-to-peer nature is the ability to bypass firewalls without manual port forwarding. Our system utilizes **UDP Hole Punching**. By simultaneously sending packets to each other via the candidates discovered by the ICE process, the routers at both ends "punch a hole" in their NAT tables, allowing a direct, persistent bi-directional stream to form.

3) **Peer Discovery via DHT:** To remove dependence on static IP lists, we integrate a **Distributed Hash Table (DHT)** (based on the Kademlia protocol). When a new node spawns, it registers its Node ID and ICE candidates in the DHT. Other peers utilize the DHT to lookup active neighbors, ensuring the network remains resilient even if the signaling server experiences temporary downtime.

## B. Protocol Layer

Once the communication channel is established, data exchange is governed by a strict application-level protocol designed for minimal overhead and high throughput.

1) **Protocol Level Addressing and Message Format:** Data packets within the mesh follow a standardized schema. To ensure global uniqueness and temporal ordering without a central clock, we utilize **ULID (Universally Unique Lexicographically Sortable Identifier)** for message IDs. The message structure is defined as follows:

TABLE I  
PROTOCOL MESSAGE STRUCTURE

| Field     | Type          | Description                          |
|-----------|---------------|--------------------------------------|
| id        | String (ULID) | 128-bit sortable unique identifier.  |
| timestamp | Int64         | Unix epoch time of creation.         |
| sender    | String (UUID) | The Node ID of the origin peer.      |
| receiver  | String (UUID) | The Node ID of the target peer.      |
| content   | Object/JSON   | The payload (Load stats or Request). |

The Javascript implementation of the message constructor, as used in our simulation, ensures that every packet carries its provenance and temporal context:

```
class Message {
  constructor(message) {
    this.message = message;
    this.message.id = ulid();
    this.message.timestamp = Date.now();
    this.sender = message.sender;
    this.receiver = message.receiver;
    this.content = message.content;
  }
}
```

2) **Heartbeat Protocol:** To maintain an up-to-date view of the neighborhood topology, nodes execute a Heartbeat Protocol.

- 1) **Emission:** Every  $\Delta t$  seconds, a node broadcasts a HEARTBEAT message containing its current CPU usage and Queue Length to all connected peers.
- 2) **Failure Detection:** If a peer fails to receive a heartbeat from neighbor  $N_i$  for a duration of  $2 \times \Delta t$ ,  $N_i$  is marked as "Suspected Dead."
- 3) **Pruning:** Upon confirming the timeout, the connection is severed, and the node triggers a DHT lookup to find a replacement neighbor, maintaining the target valency  $V$ .

3) **Website Name Resolution (DNS):** Before a client can interact with the P2P mesh, it must locate an entry point. This utilizes the traditional Client-Server **Domain Name System (DNS)**. The domain is resolved to the public IPs of the **Gateway Nodes**. It is crucial to note that DNS is exclusively a client-side operation; once the request hits a Gateway, all subsequent routing is resolved internally via the DHT and UUIDs.

## V. MIDDLEWARE LAYER

Sitting between the raw protocol handling and the application execution is the Middleware Layer. This layer serves as the central brain of the node, coordinating both immediate stability checks and long-term predictive scaling. It comprises two distinct decision engines: Rule-Based and ML-Based.

### A. Rule-Based Decision Engine

The Rule-Based engine handles immediate, critical decision-making to prevent system collapse. It operates on deterministic logic with  $O(1)$  complexity.

- **Static Thresholds:** The middleware executes instantaneous checks on every incoming packet. For example, if  $CPU > 90\%$ , the node immediately triggers **Load Shedding**, rejecting non-critical requests regardless of any ML prediction.
- **Cooldown Heuristics:** To prevent "flapping" (rapid oscillation between scaling up and down), the middleware enforces a "Cooldown Period" (e.g., 300 seconds) after any scaling event.
- **Sanity Checks:** This acts as a fail-safe override. If the ML model predicts a load of 0% but the actual CPU is at 100%, the Rule-Based layer forces a spawn event.

### B. ML-Based Adaptive Scaling

While the rule-based engine manages stability, the ML-based component drives efficiency by predicting traffic patterns before they occur.

1) *Exploratory Data Analysis (EDA):* To train our forecasting models, we utilized a dataset comprising system performance metrics sampled at fixed 10-second intervals.

a) *Dataset Features:* The dataset consists of the following key features, with no missing values recorded:

TABLE II  
DATASET FEATURE SET

| Feature          | Type    | Description                  |
|------------------|---------|------------------------------|
| cpu_util_percent | Float64 | Processor usage (0-100%).    |
| mem_util_percent | Float64 | RAM usage (0-100%).          |
| net_in           | Float64 | Incoming network throughput. |
| net_out          | Float64 | Outgoing network throughput. |
| disk_io_percent  | Float64 | Disk read/write intensity.   |

b) *Trend and Seasonality:* Initial visualization of the `cpu_util_percent` (Fig. 4) reveals clear periodic patterns corresponding to user activity cycles.

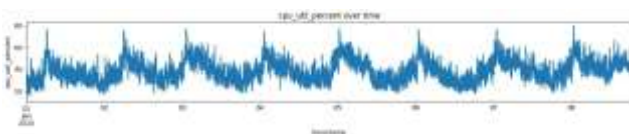


Fig. 4. CPU Utilization over Time: Raw time-series data showing distinct cyclic load patterns.

To isolate the underlying signal, we performed a seasonal decomposition (Fig. 5). The analysis confirms a strong **Seasonal** component oscillating at regular intervals, combined with a fluctuating **Trend**, validating the need for time-aware forecasting models.

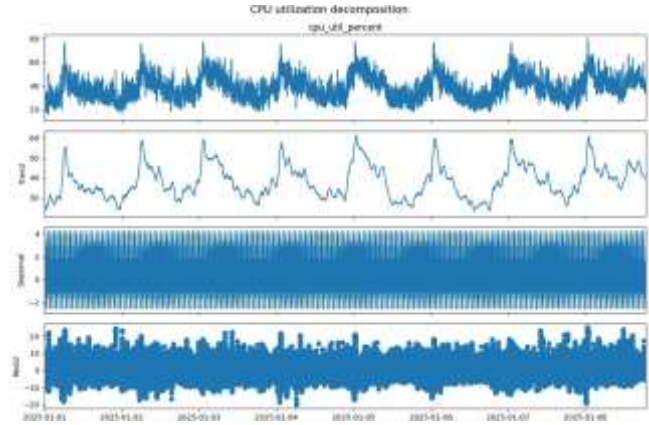


Fig. 5. Time Series Decomposition: Separating the signal into Trend, Seasonality, and Residual noise.

2) *Performance Evaluation:* We evaluated two distinct forecasting approaches to determine the optimal model for our proactive scaler: a statistical SARIMA model and a Machine Learning-based XGBoost model.

a) *SARIMA Model Performance:* The Seasonal AutoRegressive Integrated Moving Average (SARIMA) model was selected to capture the strong seasonality observed in the EDA. As shown in Fig. 6, while the model successfully captures the general wave pattern, it struggles with high-frequency noise. The forecast tends to average out extreme behaviors, making it slow to react to sudden spikes.

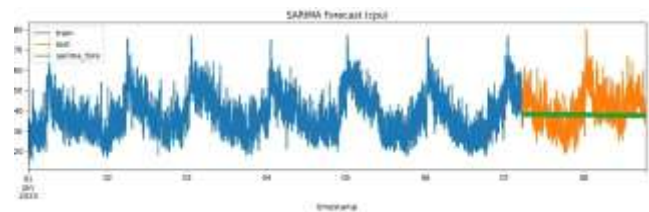


Fig. 6. SARIMA Forecast: The model captures the general baseline but lacks precision in peak prediction.

b) *XGBoost Model Performance:* To improve accuracy, we implemented a Time-Agnostic XGBoost regressor using a sliding window approach. By treating previous time steps as distinct features, the model achieved superior performance.

As evidenced in Fig. 7, the XGBoost model tracks the actual CPU spikes with remarkable precision, achieving a **Mean Absolute Error (MAE) of just 0.549**. This low error rate ensures that the auto-scaler triggers exactly when needed, minimizing both latency and resource waste.

c) *Live System Implementation:* We integrated the XGBoost model into the live P2P mesh. Fig. 8 displays the

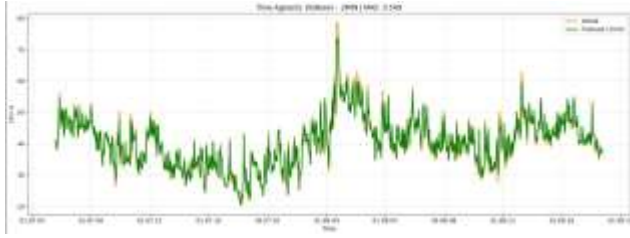


Fig. 7. XGBoost Forecast Performance: The model demonstrates near-perfect tracking of actual load (MAE: 0.549), capturing sharp spikes effectively.

real-time monitoring dashboard, where the "Predicted CPU" (Red Dotted Line) leads the "Actual CPU" (Blue Line). This predictive lead time allows the system to spawn new peers proactively.

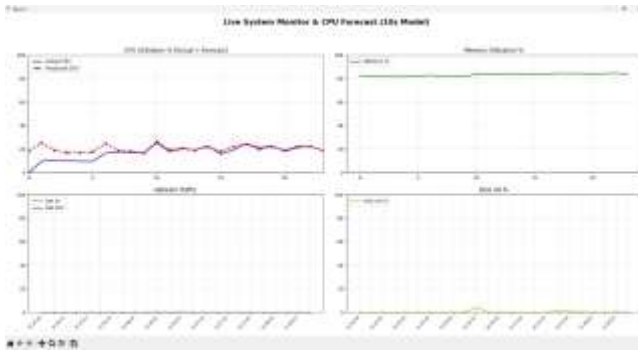


Fig. 8. Live System Monitor: The Dashboard confirms the correlation between predicted and actual load in a real-time environment.

### C. Request Lifecycle: Fates of the Request

Based on the synthesized inputs from the Rule-Based engine and the ML forecasts, every request entering a node is assigned one of four distinct "Fates." This decision logic is executed in the Application Layer (as detailed in Fig. 3).

1) *Fate 1: Accept (Local Processing)*: If the local CPU utilization  $U_{local}$  is below the safety threshold  $\tau_{safe}$  (e.g., 70%) and the predicted future load  $\hat{U}_{future}$  does not exceed capacity, the node accepts the request.

- **Action:** The request is passed to the Service Execution thread.
- **Outcome:** Zero network latency penalty; standard processing time.

2) *Fate 2: Forward (P2P Offloading)*: If  $U_{local} > \tau_{safe}$  but the cluster aggregate load is stable, the node initiates the **Smart Forwarding** routine.

- **Action:** The node queries its *Neighbor Table* to identify the peer with the lowest reported load ( $P_{min}$ ). The request is encapsulated in a new Protocol Message and transmitted via WebRTC.
- **Outcome:** The current node prevents saturation. The request incurs a minor latency penalty ( $\approx 5 - 10ms$ ) but avoids queuing delays.

3) *Fate 3: Scale & Forward (Proactive Expansion)*: This is the system's most critical failure-prevention mechanism. If  $U_{local} > \tau_{critical}$  AND the ML model predicts a sustained surge ( $\hat{U}_{future} > 90\%$ ), mere forwarding is insufficient as neighbors may soon be overwhelmed.

#### • Action:

- 1) **Signal:** The node emits a `SPAWN_INTENT` signal to the orchestration layer (e.g., Kubernetes/Docker API).
- 2) **Offload:** Crucially, the *current* request is still forwarded to a neighbor. The node does **not** attempt to process it while handling the expensive spawning operation.

- **Outcome:** Capacity is increased *before* the cluster saturates. The "Forward" action buys time for the new node to boot.

4) *Fate 4: Scale Down / Despawn*: To ensure cost-efficiency, the system monitors for under-utilization.

- **Condition:** If  $U_{local} < \tau_{idle}$  (e.g., 10%) for a duration exceeding the Cooldown Period ( $t > 300s$ ), and the node is not a designated Gateway.
- **Action:** The node sends a `LEAVE` signal to the DHT, closes active WebRTC channels, and terminates its process.
- **Outcome:** Resource reclamation and cost reduction.

## VI. SIMULATION RESULTS AND TOPOLOGICAL ANALYSIS

To empirically validate the structural behavior of the proposed P2P mesh, we conducted a discrete-event simulation using the 'GraphSimulator' engine. The objective was to observe the organic growth of the network under strict valency constraints and identify topological inefficiencies.

### A. Simulation Parameters

The simulation was initialized with the following constraints, designed to stress-test the topology:

- **Gateways (G):** 2 (Initial seed nodes).
- **Max Valency (V):** 3 (Hard limit on connections per peer).
- **Growth Rate:** 1 Node per step.
- **Seed:** 42 (For reproducibility).
- **Visual Encoding:** Purple nodes represent the initial Gateways/oldest peers; Yellow nodes represent the most recently spawned peers.

### B. Step-by-Step Evolutionary Analysis

We analyze the network evolution over 20 discrete time steps.

1) *Phase 1: Nucleation (Steps 1-3)*: In the initial phase (Fig. 9 and Fig. 10), the network exhibits ideal triangulation.

**Inference:** At this stage, the graph diameter is 1. The low valency ( $V = 3$ ) is sufficient for small clusters, as nodes have not yet saturated their connection slots.

Crystal growth (nodes: 3, edges: 2)

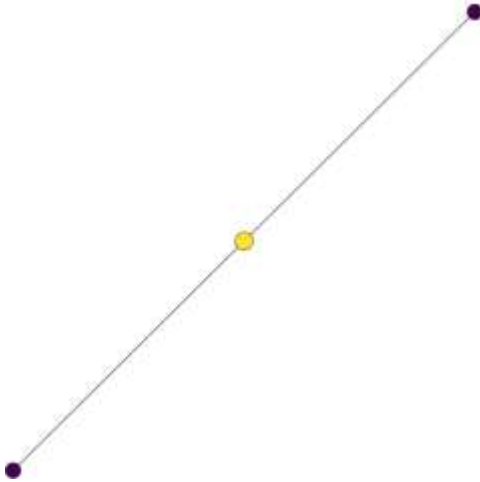


Fig. 9. Step 1 (Total Nodes: 3): The first child spawns. With  $G = 2$ , it connects to both Gateways, forming a perfect triangle. Connectivity is 100%.

Crystal growth (nodes: 6, edges: 8)

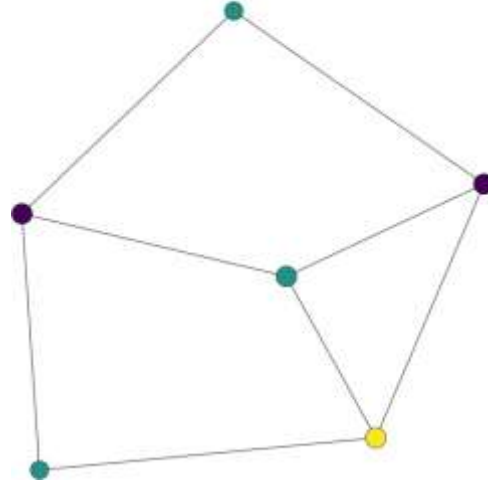


Fig. 11. Step 4 (Total Nodes: 6): The Gateway nodes (Purple) have reached  $V_{max} = 3$ . They cannot accept new connections. Growth is forced outwards.

Crystal growth (nodes: 4, edges: 4)

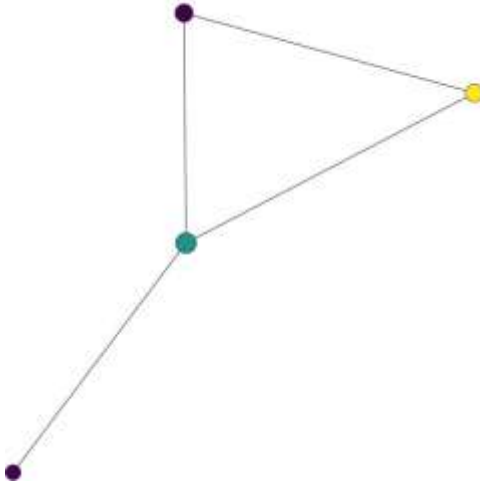


Fig. 10. Step 2 (Total Nodes: 4): The second child spawns. It attempts to maximize connectivity within the constraint  $V = 3$ . The core remains dense.

Crystal growth (nodes: 9, edges: 12)

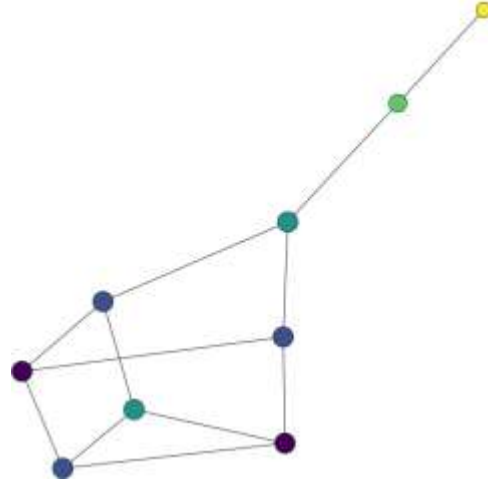


Fig. 12. Step 7 (Total Nodes: 8): The structural weakness begins to manifest. The network cannot grow "radially" (like roots) because the core is blocked. It begins to spurt directionally.

2) *Phase 2: Saturation and Elongation (Steps 4-8)*: As the node count exceeds the valency capacity of the core Gateways ( $N > G \times V$ ), the "Oldest" nodes (Purple) become fully saturated. New nodes are forced to attach to the periphery (Teal/Green nodes).

**Inference:** We observe the onset of the **Polymer Chain Phenomenon**. Because the core nodes are locked, new nodes form a single-file line extending away from the center to preserve their own connection slots.

3) *Phase 3: The Polymer Chain (Step 10+)*: By Step 10 (Fig. 13), the topology has visibly fractured into two distinct

clusters connected by a "bridge" node.

4) *Final State: Structural Failure (Step 20)*: The final simulation state (Fig. 14) confirms the hypothesis.

### C. Critical Observations

The simulation provides irrefutable evidence of the relationship between Valency ( $V$ ) and Network Diameter ( $D$ ).

- **The "Roots vs. Polymer" Analogy:** An efficient P2P network should resemble "roots," spreading wide and shallow to minimize hops. However, with low max va-

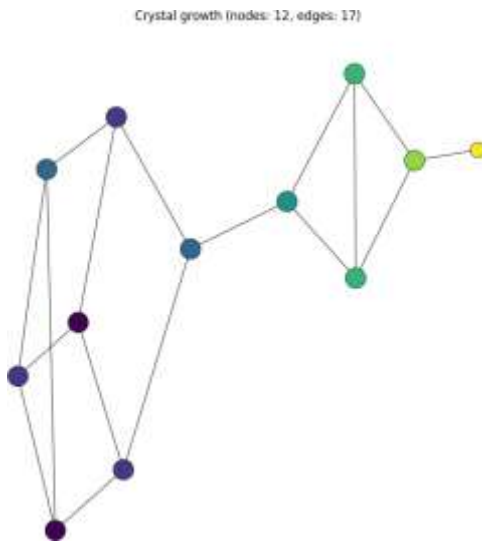


Fig. 13. Step 10 (Total Nodes: 12): Distinct clustering occurs. A request entering at the bottom Purple node requires 4+ hops to reach the Yellow tip.

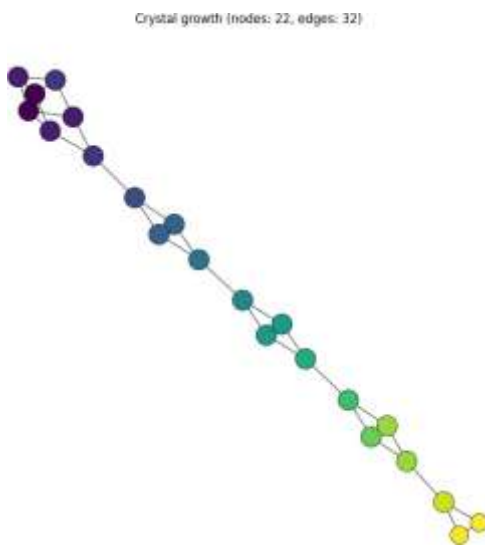


Fig. 14. Step 20 (Final State): The "Polymer Chain." With  $V = 3$ , the network has formed a long, inefficient diagonal chain instead of a mesh.

lencies ( $V \approx 3$ ), the network lacks the "width" to spread. It essentially grows like a "polymer" or a snake.

- **Latency Implications:** In the chain configuration (Fig. 14), a request entering the Gateway (Purple) must traverse  $N/2$  hops on average to find an idle node at the tail. This introduces linear latency  $O(N)$ , which is unacceptable for high-performance load balancing.
- **The Doubling Solution:** Theoretical analysis suggests that **doubling the valency** (e.g.,  $V = 6$ ) drastically shortens the chain length. It allows the core Gateways to support a "second layer" of peers, turning the linear

chain into a logarithmic tree structure  $O(\log N)$ .

## VII. CONCLUSION AND FUTURE WORK

This paper presented a novel decentralized architecture for horizontal scaling...

## REFERENCES

- [1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," *Phil. Trans. Roy. Soc. London*, vol. A247, pp. 529–551, April 1955.
- [2] J. Clerk Maxwell, *A Treatise on Electricity and Magnetism*, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
- [3] W. Zhang, "Linux Virtual Server for Scalable Network Services," in *Proc. Ottawa Linux Symp.*, 2000, pp. 212–221.
- [4] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003.
- [5] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM Int. Conf. on Distributed Systems Platforms*, 2001, pp. 329–350.
- [6] P. S. Senthil Kumar, C. S. Shajitha, and V. R. Sarma, "Gossip-based load balancing strategy for virtual machines in cloud computing," in *Proc. IEEE Int. Conf. on Computational Intelligence and Computing Research*, 2013, pp. 1–6.
- [7] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.
- [8] J. Wang, M. H. Li, B. B. Chen, and Y. Wang, "Cloud resource prediction based on LSTM-RNN," in *Proc. IEEE Int. Conf. on Cloud Computing and Big Data Analytics*, 2017, pp. 248–253.
- [9] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proc. IPTPS '02*, Springer, 2002, pp. 53–65.
- [10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. ACM SIGCOMM*, 2001, pp. 161–172.
- [11] A. Al-Nidawi and H. Yahyaoui, "A Review of Load Balancing Techniques in Cloud Computing," *Int. J. Comput. Appl.*, vol. 182, no. 11, pp. 16–22, 2018.
- [12] M. Armbrust et al., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- [13] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
- [14] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [15] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed., Hoboken, NJ: Wiley, 2015.
- [16] C. Jennings, H. Kaplan, R. Rescorla, and J. Uberti, "Resource Reservation Protocol (RSVP)," *IETF RFC 7478: Web Real-Time Communication Use Cases and Requirements*, 2015.
- [17] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session Traversal Utilities for NAT (STUN)," *IETF RFC 5389*, Oct. 2008.
- [18] T. Reddy, A. Johnston, P. Matthews, and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to STUN," *IETF RFC 5766*, Apr. 2010.
- [19] H. K. Meena, A. R. K. Rao, and E. S. Pilli, "Load Balancing in Cloud Computing: A Survey," in *Proc. Int. Conf. on Advanced Computing and Communication Systems (ICACCS)*, 2017, pp. 1–7.
- [20] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, 2016.
- [21] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.
- [22] S. S. Tyagi and P. K. Gupta, "A Comparative Study of Scheduling Algorithms in Cloud Computing," *Int. J. Eng. Sci. Technol.*, vol. 4, no. 6, pp. 2975–2983, 2012.
- [23] A. Singh and M. Hemalatha, "Cloud Computing: A Survey of Auto-Scaling," in *Proc. Int. Conf. on Green Computing and Internet of Things (ICGCIoT)*, 2015, pp. 912–915.