Improved Load Balancing in Content Delivery Networks

Mrs. Akshara Wagh[1], Dr. Sugandha Nandedkar[2], Dr. R.M Autee[3]
[1]*M. Tech. Student; Department of Computer Science and Engineering DIEMS, Aurangabad*
[2]*Assistant. Professor Department of Computer Science and Engineering DIEMS, Aurangabad*
[3]*Head, Department of CSE Department of Computer Science and Engineering DIEMS, Aurangabad*

***Abstract*—Content delivery networks (CDNs) distribute requests across edge servers to reduce origin load and support responsive content access. This paper describes the Improvised Content Load Balancing System (ICLBS), a local request-redistribution approach that uses neighboring server load information. It reviews static, adaptive, and learning-based scheduling strategies and examines the reported local-distribution scenarios at 0.5 s and 1 s, together with a 1 s flash-crowd scenario. The recorded ICLBS redirection percentages are 0%, 0%, and 30%, respectively. Existing comparison values are tabulated and plotted separately by metric for transparency. Because the arrival and service metrics have inconsistent rate/time labels, the results support descriptive observations but do not establish latency, throughput, or statistical superiority. Recent research from 2025 and 2026 motivates cache-aware routing, adaptive control, and explicit measurement of coordination overhead. The paper identifies the additional implementation and experimental details needed for reproducible evaluation.**

***Index Terms*—Content delivery network, load balancing, request redirection, edge caching, flash crowd, distributed scheduling.**

I. INTRODUCTION

Content delivery networks (CDNs) place content and, increasingly, computation near users through geographically distributed edge servers. A request-routing policy selects a server that can serve the requested object while accounting for available capacity and network conditions. Edge-compute integration extends the role of a CDN beyond static caching [1], while cooperative server selection can improve resilience to changes in demand [2].

A CDN edge server can cache resources such as images, style sheets, scripts, and video segments. On a cache hit, the edge serves the object locally; on a miss, the request may require an upstream fetch. Therefore, proximity alone does not determine performance: cache availability, network bandwidth, server utilization, and routing decisions also matter. Measurements of video delivery systems show that server or CDN selection influences the bandwidth available to users [3].

Traffic bursts can overload a server even when other nodes retain spare capacity. Hybrid CDN–peer-to-peer server selection [4] and joint content-placement and redirection policies [5] illustrate different ways to distribute demand. Redirection must also account for the capacity of the paths between nodes; moving work away from an overloaded server can otherwise create a network bottleneck.

Research published in 2025 and 2026 broadens this problem beyond queue occupancy. Salman et al. combine edge caching, multi-access edge computing (MEC), and Q-learning to choose among service locations [14]. Pleskanka and Pleskanka incorporate cache keys, content availability, and server state into CDN load balancing [15]. These studies motivate evaluating request allocation together with cache behavior, rather than treating all candidate servers as interchangeable.

Adaptive coordination also has a cost. In related software-defined networking (SDN) research, Jha et al. investigate asynchronous federated reinforcement learning and selective parameter sharing to reduce coordination overhead, while reporting load-imbalance limitations under some traffic conditions [16]. Applying such ideas to CDN request routing requires separate validation because network-controller balancing and object-level content delivery have different decisions and constraints.

This paper focuses on local request redistribution using neighboring load information. Its scope is to describe the ICLBS approach, present the available measurements, and discuss their interpretation alongside classical and recent research. It does not implement the learning-based systems reviewed here.

A reproducible comparison would require consistent metric definitions, equivalent workloads, repeated runs, and explicit measurement of the cost of state exchange.

II. RELATED WORK

Static request-routing policies, such as random selection and round robin, avoid collecting current server-load information. They have low decision overhead, but cannot directly respond to unequal queues or heterogeneous service capacity. Dynamic policies use server or network measurements to adapt assignments. Their effectiveness depends on both the usefulness and freshness of the collected state: Dahlin shows why selecting the apparently least-loaded server can perform poorly when load information is stale [9].

Randomized sampling offers a compromise between blind assignment and global state collection. The power-of-two-choices approach samples two servers and assigns work to the less loaded one. Mitzenmacher analyzes its benefits under a specified queueing model [10]; those theoretical results should not be transferred to a heterogeneous CDN without checking the model assumptions. Centralized and distributed redirection also differ in state requirements, overhead, and failure behavior [11].

Queue-adjustment policies redistribute queued work, whereas rate-adjustment policies act on arriving requests. A hybrid policy can influence both queue occupancy and incoming load. Zeng and Veeravalli analyze queue-and-rate-adjustment policies for distributed networks [12]. The choice among these models depends on whether queued requests can be moved, the cost of redirection, and the frequency of state updates.

CDN-specific research emphasizes objectives beyond queue length. Liu et al. jointly consider content placement, congestion avoidance, and request redirection [5]. Yang et al. incorporate viewer engagement into live-stream scheduling [6], and Tran et al. use a multi-armed bandit formulation for server selection in an SDN-based CDN [7]. Measurements of YouTube server selection further show that delivery decisions involve multiple factors beyond round-trip time [8].

In 2025, Salman et al. proposed a hybrid CDN architecture combining edge caches, MEC offloading, and Q-learning-based routing [14]. Its agent selects a cache, MEC server, or origin according to network state. This makes it relevant to adaptive service-node selection, but its combined caching and computation decisions differ from ICLBS neighbor-load redistribution. A fair comparison would need to align cache configuration, workloads, and service objectives.

In 2026, Pleskanka and Pleskanka studied distributed processing of resource-intensive CDN requests using resource-popularity analytics and a composite cache key [15]. Their routing method considers whether content is already available at an edge server as well as server state. They report improved caching effectiveness and page-load performance in simulation. For the present work, the main implication is to add content availability to future routing experiments; their simulation outcomes do not validate the ICLBS measurements.

Also in 2026, Jha et al. presented asynchronous federated reinforcement learning for SDN load balancing, using hierarchical coordination and selective parameter sharing [16]. They report lower latency and communication overhead relative to their baselines, alongside weaker load balance in some traffic regimes. This is adjacent evidence for the importance of coordination cost and update staleness, not a direct CDN benchmark.

Taken together, the literature motivates evaluating load, cache locality, and coordination overhead jointly. The present study retains a simpler local redistribution scope. Its contribution is limited to the stated design and available observations; the current measurements cannot establish a new state-of-the-art result. Table I summarizes representative studies and their relevance.

Table I. Summary of related research

Author	Paper Title	Description
[1] Kyryk et al. (2023)	Adaptive Edge Compute Module in CDN Networks	Edge-compute module for CDN service adaptation; relevant to extending delivery nodes beyond static storage.

Author	Paper Title	Description
[2] Nishiyama et al. (2012)	A Cooperative User-System Approach for Optimizing Performance in Content Distribution/Delivery Networks	Cooperative user/system server selection to address demand fluctuations and overload.
[3] Adhikari et al. (2015)	Measurement Study of Netflix, Hulu, and a Tale of Three CDNs	Measurements of commercial video CDNs show why delivery-path selection matters.
[4] Saengarunwong and Sanguankotchakorn (2018)	A Two-Step Server Selection in Hybrid CDN-P2P Mesh-based for Video-on-Demand Streaming	Two-step server selection in hybrid CDN–P2P video delivery; considers network and peer proximity.
[5] Liu et al. (2018)	Congestion Avoidance and Load Balancing in Content Placement and Request Redirection for Mobile CDN	Joint content placement and request redirection to balance load while limiting congestion.
[6] Yang et al. (2019)	Viewer-Oriented CDN Scheduling on Crowdsourced Live Video Stream	Viewer engagement informs live-stream placement and source selection.
[7] Tran et al. (2019)	MABRESE: A New Server Selection Method for Smart SDN-Based CDN Architecture	Multi-armed bandit server selection within an SDN-based CDN architecture.
[8] Torres et al. (2011)	Dissecting Video Server Selection Strategies in the YouTube CDN	Measurement study of YouTube routing shows multiple influences on server selection.
[14] Salman et al. (2025)	Hybrid CDN Architecture Integrating Edge Caching, MEC Offloading, and Q-Learning-Based Adaptive Routing	Joint service-location decisions using caches, MEC, and Q-learning; broader scope than neighbor-load redistribution.
[15] Pleskanka and Pleskanka (2026)	Research of Methods for Distributed Processing of Resource-Intensive Requests in Content Delivery Networks	Cache-key and server-state-aware delivery; motivates cache-aware routing experiments.
[16] Jha et al. (2026)	Asynchronous Federated Reinforcement Learning for Scalable Load Balancing in Software-Defined Networks	Hierarchical asynchronous learning with selective sharing; adjacent SDN evidence, not a direct CDN benchmark.

III. PROPOSED METHODOLOGY

A. Architecture and comparison policies

Figure 1 shows the supplied architecture: a request queue and master server precede allocation to cluster-local queues. The proposed local redistribution step uses load information from neighboring nodes. This combines an entry-point dispatcher with a distributed

balancing stage; it should not be described as a fully decentralized architecture. The figure also lists heap-space allocation and proximity sensing, for which implementation details are not available in the present record.

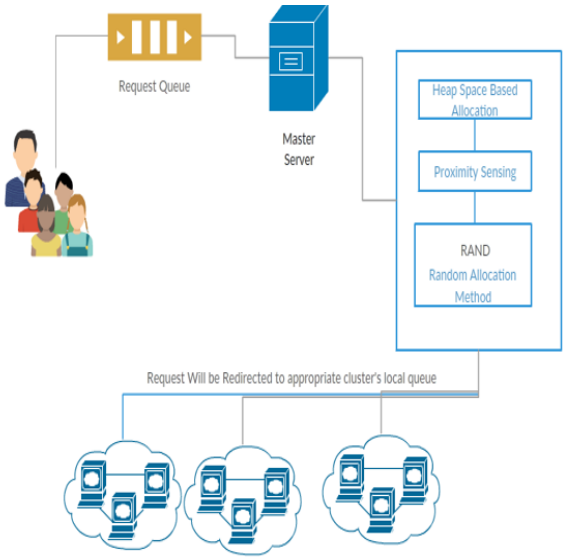

Figure 1. Supplied ICLBS architecture

Random selection (RAND) assigns each incoming request to an eligible server with equal probability. Round robin (RR) cycles through eligible servers without consulting their current queues. These are static routing baselines; they should be distinguished from task-bundle sizing in a parallel-computing framework.

Least-loaded selection (LL) sends a request to the server with the lowest measured load. Stale measurements can cause several dispatchers to select the same apparently underloaded node [9]. Response-time-based selection instead uses a measured time metric, whose sampling window and network contribution must be specified.

Two random choices (2RC), also described in the original draft as 2RCBBA, samples two servers and assigns the request to the less-loaded candidate [10]. The supplied charts additionally contain a policy labeled CLB. Its exact expansion and implementation are not documented, so CLB is retained as a source label rather than assigned an inferred meaning.

The ICLBS description proposes periodic exchange of load information between neighboring nodes. On receiving a request, a server either serves it locally or redirects it to an eligible neighbor. Local exchange limits the set of nodes involved in each update, but the total overhead still depends on the number of neighbors, update frequency, message size, and request rate. Redirection also requires loop prevention and a fallback when no usable neighbor is available.

B. Task-bundle sizing background

Manual task bundling uses a fixed number of tasks per bundle. In JPPF, bundle-size selection is distinct from CDN request routing [13]. These background policies are not identified as separate measured baselines in the supplied charts.

Proportional task bundling uses previous node performance to allocate work [13]. Let $\bar{t}_i > 0$ be the measured mean execution time per task at node i, $p > 0$ the proportionality factor, n the number of eligible nodes, and B the available task budget. Faster nodes receive larger weights. An idealized normalized allocation is:

$w_i = (1/\bar{t}_i)^p$; $W = \Sigma_{j=1}^{n} w_j$; $b_i^* = B\, w_i/W$.

Here b_i^* is a real-valued target, not an executable integer allocation. An implementation must specify rounding, residual-task allocation, minimum and maximum bundle sizes, and initialization for nodes without timing samples.

The performance-cache size controls how much recent execution history contributes to a node's estimate. The proportionality factor controls how strongly differences in mean task time affect the target bundle sizes. Their configured values and the JPPF version are not reported in the supplied experiment.

C. Local redistribution rule and queue model

The original pseudocode does not define its probability weights and uses a single binary random choice for multiple queued requests. It therefore cannot establish the implemented weighted-selection behavior. The following mathematical rule makes the intended load-difference idea explicit as a design specification; it must be checked against the implementation before being treated as the algorithm used for the reported results.

For node i, let N_i contain eligible neighbors with usable state. Define $d_{ij} = \max(0, L_i - L_j)$ and $D_i = \Sigma_{j \in N_i} d_{ij}$, where L denotes a consistently defined load metric. If $D_i = 0$, retain the request locally. Otherwise, select one neighbor j with probability $P_{ij} = d_{ij}/D_i$. Draw a new selection for each request. Eligibility, state expiration, redirection limits, and failed-transfer handling are

implementation requirements, not validated features of the current experiment.

Let $q_i[k]$ be the queue length at the beginning of interval k, $A_i[k]$ the requests admitted during the interval after routing, and $C_i[k]$ the service capacity during that interval, all measured in requests. Under a slotted model in which service acts on the initial queue and admitted arrivals join at the end, the nonnegative queue update is:

$$q_i[k+1] = \max\{0, q_i[k] - C_i[k]\} + A_i[k]. \quad (1)$$

Equation (1) states its event-ordering convention explicitly. If λ_i and μ_i are arrival and service rates in requests/s over an interval Δt, then $A_i \approx \lambda_i \Delta t$ and $C_i \approx \mu_i \Delta t$. Redirection changes the admitted arrival count at both the sending and receiving nodes. A queue can drain when effective demand stays below capacity, but a short observation window alone does not establish long-run stability.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. Available measurements and metric definitions

The supplied results cover a local-distribution setting labeled 0.5 s, a local-distribution setting labeled 1 s, and a flash-crowd setting labeled 1 s. The role of these labels—sampling interval, workload duration, or another configuration parameter—is not specified. Tables II–IV reproduce the numerical data embedded in the original charts, rounded to three decimal places; no additional experiments were performed for this revision.

The source labels its first two metrics “average arrival rate” and “average service rate” but gives their unit as seconds. Rates should be expressed in requests/s, whereas times or intervals may be expressed in seconds. The tables therefore use “reported arrival metric” and “reported service metric” until the implementation is checked. Lower service time and higher service rate imply different performance interpretations, so neither direction is assumed here. Redirection is retained as the reported percentage.

End-to-end response time, if measured in a future evaluation, should include routing and transfer time, queue waiting, service time, and return transfer. These components must not be conflated with arrival rate or service capacity. The current document provides no consistent end-to-end latency measurements.

B. Local distribution: 0.5 s setting

Table II. Reported comparison values for the 0.5 s local-distribution setting

Policy	Reported arrival metric	Reported service metric	Redirection (%)
RAND	145.575	412.606	0.000
RR	138.827	377.649	0.000
LL	5.990	50.442	37.812
2RC	3.608	33.124	0.000
CLB	3.675	22.361	45.024
ICLBS	3.653	13.760	0.000

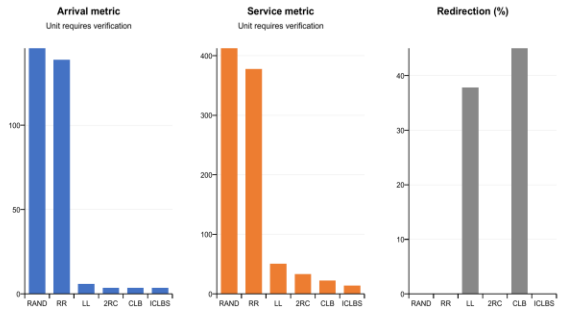

Figure 2. Comparison values from the original chart for the 0.5 s local-distribution setting; metric units require verification

C. Local distribution: 1 s setting

Table III. Reported comparison values for the 1 s local-distribution setting

Policy	Reported arrival metric	Reported service metric	Redirection (%)
LL	5.546	47.531	32.541
2RC	4.858	49.046	0.000
CLB	4.063	31.447	32.194
ICLBS	2.117	5.860	0.000

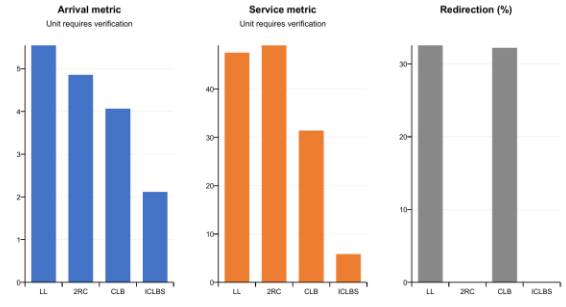

Figure 3. Comparison values from the original chart for the 1 s local-distribution setting; metric units require verification

D. Flash-crowd scenario: 1 s setting

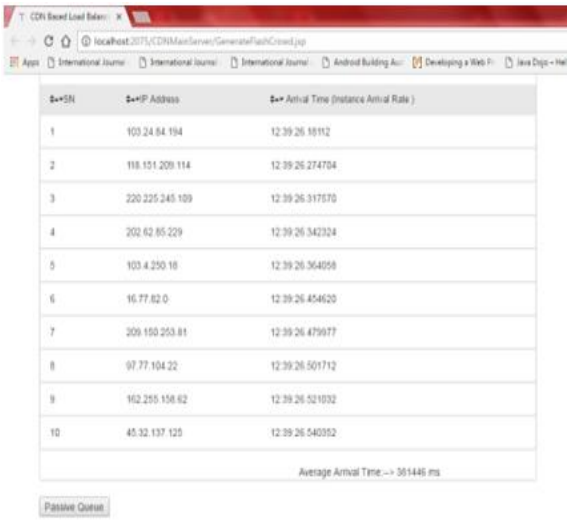

Figure 4. Supplied flash-crowd generation screenshot

Table IV. Reported comparison values for the 1 s flash-crowd setting

Policy	Reported arrival metric	Reported service metric	Redirection (%)
LL	184.965	28.978	31.019
2RC	190.885	471.419	0.000
CLB	182.392	26.872	37.914
ICLBS	127.148	10.420	30.000

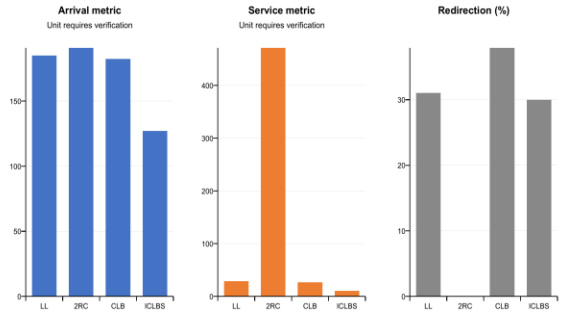

Figure 5. Comparison values from the original chart for the 1 s flash-crowd setting; metric units require verification

Across the supplied scenarios, the ICLBS redirection value increases from 0% in the two local-distribution settings to 30% during the flash-crowd setting. In the flash-crowd chart, LL and CLB have reported redirection values of 31.019% and 37.914%, respectively, while 2RC has 0%. These are descriptive comparisons only: a lower redirection percentage does not establish better service, since it may reflect local capacity, policy behavior, or unreported failures. The arrival and service values cannot resolve that question until their units and measurement procedures are confirmed.

E. Reproducibility and limitations

The record does not specify hardware, server count, topology, link conditions, software versions, cache state, content sizes, workload distribution, baseline configurations, update interval, warm-up period, number of repetitions, or uncertainty estimates. Raw logs are also unavailable. Consequently, statistical significance, queue stability, execution-time savings, and general superiority of static or adaptive algorithms cannot be established. A complete evaluation should use equivalent offered workloads, record completion and failure counts, and report latency distributions, throughput, cache-hit ratio, queue occupancy, and control-message overhead with repeated trials.

V. CONCLUSION

This paper describes local load redistribution for CDNs and presents the available ICLBS observations alongside classical and recent research. The supplied data record no redirection in the local-distribution settings and 30% redirection in the flash-crowd setting. They do not justify the original claim that static algorithms are generally more efficient than adaptive methods. Ambiguous metric units and missing experimental details limit interpretation. Future work should verify the implemented decision rule, define the measurements consistently, and evaluate cache-aware neighbor selection and state-update costs under repeatable workloads. The 2025–2026 literature provides relevant directions [14]–[16], but its reported improvements must be evaluated independently in this system.

REFERENCES

[1] M. Kyryk, N. Pleskanka and M. Pleskanka, "Adaptive Edge Compute Module in CDN Networks," 2023 IEEE 5th International Conference on Advanced Information and Communication Technologies (AICT), Lviv, Ukraine, 2023, pp. 41-43, doi: 10.1109/AICT61584.2023.10452684.

[2] H. Nishiyama, H. Yamada, H. Yoshino and N. Kato, "A Cooperative User-System Approach for Optimizing Performance in Content Distribution/Delivery Networks," in IEEE Journal on Selected Areas in Communications, vol. 30, no. 2, pp. 476-483, February 2012, doi: 10.1109/JSAC.2012.120228.

[3] V. K. Adhikari et al., "Measurement Study of Netflix, Hulu, and a Tale of Three CDNs," in IEEE/ACM Transactions on Networking, vol. 23, no. 6, pp. 1984-1997, Dec. 2015, doi: 10.1109/TNET.2014.2354262.

[4] A. Saengarunwong and T. Sanguankotchakorn, "A Two-Step Server Selection in Hybrid CDN-P2P Mesh-based for Video-on-Demand Streaming," 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2018, pp. 499-504, doi: 10.1109/ICTC.2018.8539453.

[5] J. Liu, Q. Yang and G. Simon, "Congestion Avoidance and Load Balancing in Content Placement and Request Redirection for Mobile CDN," in IEEE/ACM Transactions on Networking, vol. 26, no. 2, pp. 851-863, April 2018, doi: 10.1109/TNET.2018.2804979.

[6] W. Yang, Y. Hu, L. Ding and Y. Tian, "Viewer-Oriented CDN Scheduling on Crowdsourced Live Video Stream," 2019 IEEE 2nd International Conference on Electronics and Communication Engineering (ICECE), Xi'an, China, 2019, pp. 112-117, doi: 10.1109/ICECE48499.2019.9058519.

[7] H. -A. Tran, S. Souihi, D. Tran and A. Mellouk, "MABRESE: A New Server Selection Method for Smart SDN-Based CDN Architecture," in IEEE Communications Letters, vol. 23, no. 6, pp. 1012-1015, June 2019, doi: 10.1109/LCOMM.2019.2907948.

[8] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo and S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," 2011 31st International Conference on Distributed Computing Systems, Minneapolis, MN, USA, 2011, pp. 248-257, doi: 10.1109/ICDCS.2011.43.

[9] M. Dahlin, "Interpreting stale load information," IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 10, pp. 1033–1047, 2000. doi: 10.1109/71.888643.

[10] M. Mitzenmacher, "The power of two choices in randomized load balancing," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp. 1094–1104, 2001. doi: 10.1109/71.963420.

[11] V. Cardellini, M. Colajanni, and P. S. Yu, "Request redirection algorithms for distributed Web systems," IEEE Transactions on Parallel and Distributed Systems, vol. 14, no. 4, pp. 355–368, 2003. doi: 10.1109/TPDS.2003.1195408.

[12] Z. Zeng and B. Veeravalli, "Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks," IEEE Transactions on Computers, vol. 55, no. 11, pp. 1410–1422, 2006. doi: 10.1109/TC.2006.180.

[13] JPPF, "ProportionalBundler," JPPF 6.1.4 API documentation. Available: https://javadoc.io/static/org.jppf/jppf-common/6.1.4/org/jppf/load/balancer/impl/ProportionalBundler.html. Accessed: Sep. 3, 2026. [Software documentation.].

[14] A. D. Salman, A. T. Zeyad, A. A. S. Al-karkhi, S. M. Raafat, and A. J. Humaidi, "Hybrid CDN architecture integrating edge caching, MEC offloading, and Q-learning-based adaptive routing," Computers, vol. 14, no. 10, Art. no. 433, Oct. 2025. doi: 10.3390/computers14100433.

[15] N. Pleskanka and M. Pleskanka, "Research of methods for distributed processing of resource-intensive requests in content delivery networks," Computer Design Systems. Theory and Practice, vol. 8, no. 1, pp. 175–188, 2026. doi: 10.23939/cds2026.01.175.

[16] A. K. Jha, A. Mahendran, I. Ghafir, and M. Hamada, "Asynchronous federated reinforcement learning for scalable load balancing in software-defined networks," Discover Computing, vol. 29, Art. no. 320, Jun. 2026. doi: 10.1007/s10791-026-10147-4.