Performance Evaluation of Active and Passive Scheduling Algorithms in IP Tunneling Load Balancing for DNS Traffic

Salisa 'Asyarina Ramadhani[1*], Luxfy Roya Azmi[2], Aulia Novira[3], Hazimah Widyagustin[4], Rendy Kurniawan[1]

[1]Electrical Engineering Study Program, Universitas Jambi, Jambi, 36361, Indonesia

[2]Health Informatics Study Program, Universitas Mercubaktijaya, Padang, 25146, Indonesia

[3]Electronic Engineering Study Program, Department of Electrical Engineering, Politeknik Negeri Padang, Padang, 25164, Indonesia

[4]Electrical Engineering Study Program, Department of Electrical Engineering, Politeknik Negeri Padang, Padang, 25164, Indonesia

E-mail: salisa@unja.ac.id*, luxfyroyaazmi@mercubaktijaya.ac.id, aulianovi@pnp.ac.id, hazimahwidyagustin@pnp.ac.id, rendikurniawan186@gmail.com

***Abstract*— The exponential increase in global internet traffic frequently causes processing bottlenecks on Domain Name System (DNS) servers, degrading network Quality of Service (QoS). To address this, an IP Tunneling (LVS-TUN) load-balancing topology was implemented using Linux Virtual Server (IPVSADM) to evaluate 10 active and passive scheduling algorithms handling connectionless UDP traffic. Performance was assessed by simulating high-stress UDP flooding with 1,000 queries via DNSPerf across five iterations, extracting the average batch completion time and throughput, calculated from Wireshark packet traces, in Mbps. Under the tested single-client configuration, the Destination Hashing (DH) algorithm produced the lowest measured batch completion time of 1.0060 seconds and the highest calculated throughput of 1.7520 Mbps. However, this outcome is governed by the deterministic routing of the single-client test topology. Crucially, operational anomalies were identified in dynamic algorithms, specifically the Shortest Expected Delay (SED) and Never Queue (NQ) algorithms. Rather than distributing the load, rapid, connectionless UDP queries prompted IPVS to reuse active connection entries during the timeout window, thereby concentrating traffic on a single backend server.**

Keywords—Tunneling, Load Balancing, UDP, Batch Completion Time, Throughput

I. INTRODUCTION

The rapid evolution of Internet of Things (IoT) architectures and the ubiquitous adoption of cloud computing services have fundamentally transformed the global digital landscape, triggering an exponential and unprecedented surge in global data demand [1], [2], [3], [4]. Within this expansive infrastructure, millions of interconnected devices constantly require instantaneous network resolutions to function correctly. At the very heart of this communication infrastructure lies the Domain Name System (DNS), which serves as the Internet's primary gateway, translating human-readable domain names into machine-readable IP addresses. Because almost every online transaction or machine-to-machine (M2M) communication begins with a DNS lookup, the high volume of concurrent queries directed to DNS servers often creates a severe processing bottleneck. When a single server is forced to handle this massive, unfiltered influx of requests, it inevitably results in resource exhaustion, abnormally high network latency, and a drastic decline in the overall Quality of Service (QoS) experienced by end-users [5], [6].

To mitigate these bottlenecks and ensure high availability and seamless network scalability, implementing load balancing has become the standard architectural imperative for distributing traffic evenly across a server cluster [7], [8]. In the context of Transport Layer (Layer 4) distribution, the Linux Virtual Server (LVS), integrated into the Linux kernel, is widely recognized for its robust packet-forwarding capabilities [9]. Among the various network topologies supported by LVS, IP Tunneling (LVS-TUN) offers comprehensive technical advantages over alternatives such as Network Address Translation (NAT) or Direct Routing (DR) [10]. Specifically, LVS-TUN enables the encapsulation of private IP packets within public IP headers, uniquely empowering the load balancer to manage server cluster members that are geographically distributed across different Local or Wide Area Networks (LAN/WAN) [6], [11]. Ultimately, the routing efficiency of this tunneling topology is largely determined by the mathematical scheduling algorithm used to determine which server node should process each incoming query [12], [13].

Although an extensive body of recent literature has examined the optimization of various load-balancing algorithms [13], a specific research gap persists regarding the operational behavior of LVS schedulers when handling DNS traffic. The majority of foundational load-balancing performance evaluations naturally lean toward stateful, connection-oriented protocols, particularly Transmission Control Protocol (TCP) traffic utilized in web environments [14], [15], [16]. In TCP-based systems, a three-way handshake establishes a persistent connection, allowing the load balancer to accurately track the active load on each server. Conversely, DNS infrastructure primarily operates over the User Datagram Protocol (UDP)—a fundamentally connectionless protocol that fires datagrams without establishing a dedicated session.

Consequently, there is a critical need to evaluate how the connectionless nature of UDP—specifically, how IPVS handles UDP timeouts and connection entries—affects real-time mathematical computations in load balancer scheduling algorithms [17]. This impact becomes potentially disruptive when applied to dynamic (active) algorithms, such as Shortest Expected Delay (SED) or Never Queue (NQ) [18], [19]. Because dynamic algorithms calculate server weight based on the number of "active connections," applying them to connectionless UDP traffic creates a systemic mismatch. This architectural dissonance can trigger severe operational anomalies in which the load balancer's routing mechanisms override the scheduling logic, causing data traffic to

accumulate disastrously on a single server (computational load imbalance) [20], [21].

Addressing this specific gap, this article systematically evaluates the performance of 10 distinct scheduling algorithm variations (encompassing both active and passive approaches) using the IPVSADM module in an IP Tunneling topology designed exclusively for DNS UDP traffic. The measured QoS parameters strictly focus on system batch completion time and average network throughput during high-stress simulations involving targeted query flooding. The primary objective of this study is twofold: to empirically evaluate the comparative performance of these algorithms under single-client flooding conditions and, more importantly, to demonstrate, isolate, and rigorously analyze the operational anomalies that arise in dynamic LVS algorithms when forced to handle UDP traffic.

II. Materials and Methods

This is an applied experimental study in computing and networking conducted at the Institute for Information and Communication Technology Development (LPTIK) at Universitas Andalas. A quantitative methodology was utilized to evaluate the load balancer's performance by measuring specific QoS parameters during DNS traffic flooding.

A. System Architecture and Experimental Setup

The system architecture was designed using the Linux Virtual Server via IP Tunneling (LVS-TUN) topology. To ensure reproducibility, the network environment was fully virtualized using VMware on a physical server with an Intel Xeon E3-1220 v3 processor (4 Cores) and 16 GB of RAM. All virtual nodes in this architecture ran Debian 10 Buster with the Linux kernel 4.19.0-16-amd64.

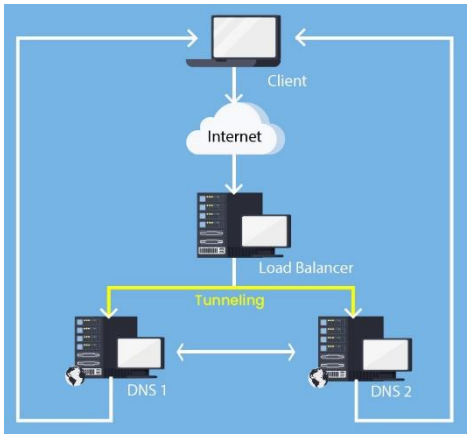


Fig. 1. Network Topology Design

The architecture is functionally divided into three primary components. The first component is the client node, which operates as a Virtual Private Server (VPS) and serves as the sole source of DNS queries using the DNSPerf utility. This client interacts directly with the second component, the load balancer (director). Configured with the IPVSADM module within the Linux kernel, the director serves as the primary gateway, receiving client requests and routing them to the backend cluster. Finally, the third component is the DNS server cluster (real servers), which consists of two synchronized VPS nodes operating the BIND9 utility. These nodes are configured as a DNS Master at 10.164.0.24 and a DNS Slave at 10.164.0.25. A visual representation of this complete network architecture and tunneling topology is illustrated in Fig. 1.

B. Variations of Scheduling Algorithms

The IPVSADM module on the director node was configured to sequentially execute 10 distinct scheduling algorithms, which are broadly classified into two operational categories based on their load-calculation mechanisms:

1. **Passive (Static) Algorithms:** These consist of scheduling algorithms that do not track active connections, including Round Robin (RR), Weighted Round Robin (WRR), Destination Hashing (DH), and Source Hashing (SH).
2. **Active (Dynamic) Algorithms:** Consist of algorithms that take server status into account in real time, including Least Connection (LC), Weighted Least Connection (WLC), Locality-Based Least Connection (LBLC), Locality-Based Least Connection with Replication (LBLCR), Shortest Expected Delay (SED), and Never Queue (NQ).

C. Test Scenarios and Parameters

Performance testing was conducted by simulating high-stress client traffic directed at the load balancer's virtual IP (VIP) address. The execution logic for the performance evaluation is detailed in **Procedure 1** to ensure full experimental reproducibility.

Procedure 1: Performance Evaluation of LVS Scheduling Algorithms

Input: Set of 10 Scheduling Algorithms (***A***), Workload of 1,000 DNS queries (***Q***), Iterations ***N*** = 5

Output: Average Batch completion time, Average Throughput (in Mbps)

1. Initialize LVS-TUN network topology (Client, Director, DNS Master, DNS Slave).
2. Execute preliminary TCP connection to validate IPVSADM routing distribution.
3. **For each** scheduling algorithm ***a*** **in** ***A***:
4. Apply algorithm ***A*** configuration to the IPVSADM director.
5. **For** trial ***i*** = 1 **to** ***N***:
6. Start **Wireshark** packet capture on the network interface.
7. Execute **DNSPerf** to transmit ***Q*** via UDP to the director's VIP.
8. Stop capture upon complete resolution of ***Q***.
9. Extract **Batch completion time** and *Throughput* metrics from Wireshark.
10. **End For**
11. Calculate the average *Batch completion time* and *Throughput* across ***N*** trials.
12. **End For**
13. Compile and compare final QoS metrics for all algorithms ***A***

During testing, the primary evaluated Quality of Service (QoS) parameters were batch completion time and

throughput. Batch completion time (T_b), recorded in seconds, is defined as the total time required to transmit and completely resolve the 1,000 queries generated by DNSPerf. Network throughput was calculated from packet-level traces captured by Wireshark directly at the network interface. The throughput (***TP***), expressed in Megabits per second (Mbps), was determined using Equation (1):

$$TP = \frac{Total\ bits\ transferred}{T_b \times 10^6} \quad (1)$$

where the total bits transferred encompasses all transmitted DNS query and response datagrams within the captured test interval.

III. Results and Discussions

A. System Performance Without a Load Balancer (Baseline)

The testing phase commenced by establishing a baseline for DNS traffic performance in a local network environment without a load balancer. To rigorously evaluate the system's absolute processing capacity, a flooding simulation was executed using a concentrated burst of 1,000 simultaneous DNS queries. As detailed in Table 1, the system yielded an average batch completion time of 1.0106 seconds and an average throughput of 1.7472 Mbps across five separate testing iterations. The variance across these five tests was exceptionally minimal, indicating a stable but fundamentally limited processing capability within the single-server architecture.

TABLE I. Test Results Without a Load Balancer

Testing Iteration	Batch completion time (s)	Throughput (Mbps)
Testing 1	1.010	1.746
Testing 2	1.012	1.748
Testing 3	1.010	1.748
Testing 4	1.011	1.747
Testing 5	1.010	1.747
Average	1.0106	1.7472

In the baseline scenario without a load balancer, DNS traffic followed a direct path, with client queries explicitly directed to the primary DNS server without intermediate load distribution. Under the flooding load of 1,000 queries, this single receiving server experienced resource exhaustion, leading to packet queueing and the observed baseline latency. Unlike the distributed configurations, this scenario lacked an intermediary balancing layer, exposing the single node to the complete processing burden of concurrent resolutions.

B. Performance Under Load Balancing: TCP Validation vs. UDP Anomalies

Before running the UDP simulations, an initial TCP validation was conducted solely to verify that both backend real servers were reachable and capable of processing connections through the tunneling topology. Fig. 2 illustrates this connectivity check using the NQ algorithm as a representative sample, verifying that both backend nodes correctly register active TCP connection states under the director's routing rules.

```
root@salisa-LB:~# sudo ipvsadm -l
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.164.0.60:domain nq
  -> 10.164.0.24:domain           Tunnel  1      1          0
  -> 10.164.0.25:domain           Tunnel  1      2          0
root@salisa-LB:~# telnet 10.164.0.60 53
Trying 10.164.0.60...
^C
root@salisa-LB:~# sudo ipvsadm -l
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.164.0.60:domain nq
  -> 10.164.0.24:domain           Tunnel  1      2          0
  -> 10.164.0.25:domain           Tunnel  1      2          0
```

Fig. 2. Load Balancing with the TCP Protocol on NQ

In stark contrast, profound operational anomalies emerged when the system was subjected to the actual DNS UDP traffic, particularly when dynamic algorithms such as SED and NQ were applied. Rather than distributing the load evenly, ipvsadm monitoring tools—specifically, the ActiveConn and InActConn counters—revealed that UDP queries were piling up on a single server node (IP: 10.164.0.25). This accumulation of connections on a single server is shown in Fig. 3.

```
  -> 10.164.0.24:53               Tunnel  1      0          0
  -> 10.164.0.25:53               Tunnel  1      0          0
root@salisa-LB:~# sudo ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
UDP  10.164.0.60:53 nq
  -> 10.164.0.24:53               Tunnel  1      0          0
  -> 10.164.0.25:53               Tunnel  1      0          6
root@salisa-LB:~# sudo ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
UDP  10.164.0.60:53 nq
  -> 10.164.0.24:53               Tunnel  1      0          0
  -> 10.164.0.25:53               Tunnel  1      0          8
root@salisa-LB:~# sudo ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
UDP  10.164.0.60:53 nq
  -> 10.164.0.24:53               Tunnel  1      0          0
  -> 10.164.0.25:53               Tunnel  1      0          10
```

Fig. 3. Load Balancing with the UDP Protocol on NQ

This phenomenon is not driven by standard "cache memory," but rather by the inherent mechanisms IPVS utilizes to handle connectionless protocols. When the initial UDP datagram arrives, IPVS allocates a connection entry and sets a specific UDP timeout window. Because our simulation involved a rapid, concentrated flood of 1,000 queries originating from a single client IP within this brief timeout period, the scheduler effectively bypassed the load-recalculation logic (such as SED's active-delay metrics or NQ's idle-server checks). Instead, to minimize computational overhead, the director continuously forwarded subsequent datagrams to the existing active connection entry associated with the DNS Slave.

While this aggressive connection-entry reuse defeats the primary purpose of load balancing by overloading a single server, it paradoxically results in faster individual batch completion times. The system incurs zero computational overhead when recalculating server weights or resolving new destination addresses for incoming datagrams, which helps explain why the SED algorithm recorded the fastest batch completion time among the dynamic algorithms despite the severe load imbalance.

C. Evaluation of Active Scheduling Algorithms

The implementation of IPVSADM utilizing active (dynamic) algorithms generally improved the system's QoS performance compared to the baseline. Among the active

algorithm variations, the Shortest Expected Delay (SED) algorithm recorded the lowest duration, achieving an average batch completion time of 1.0068 seconds and an average throughput of 1.7508 Mbps, followed by the Weighted Least Connection (WLC) algorithm with a batch completion time of 1.0082 seconds and a throughput of 1.7500 Mbps. A comprehensive comparison of batch completion times for all tested active algorithms is shown in Fig. 4.

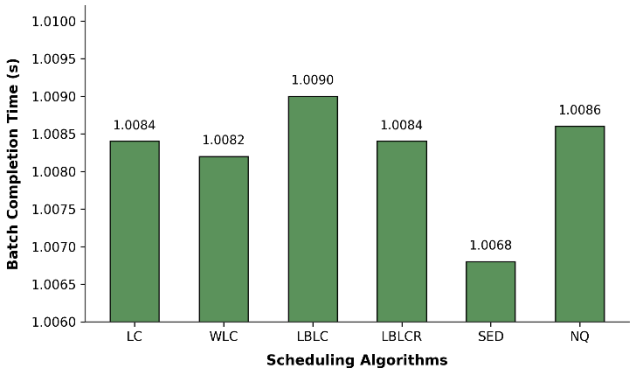


Fig. 4. Batch Completion Time Comparison for Active Scheduling Algorithms Comparison for Active Scheduling Algorithms

The SED algorithm operates in a highly responsive manner by calculating the ratio of active connections to server computing capacity in real time, thereby theoretically ensuring that traffic is routed to the server with the lowest expected latency. Furthermore, fundamental network principles dictate that at a constant workload, data throughput is inversely proportional to the batch completion time. As demonstrated by the SED algorithm, achieving the fastest batch completion time (1.0068 s) inherently yielded the highest network throughput (1.7508 Mbps). Conversely, the LBLC algorithm recorded the slowest batch completion time, 1.0090 seconds, resulting in the lowest throughput, 1.7470 Mbps. These throughput variations are illustrated in Fig. 5.

However, as previously analyzed in Section 3.2, the exceptionally fast batch completion time of the SED (and NQ) algorithm under UDP traffic is a byproduct of the connectionless UDP timeout mechanism rather than a perfectly balanced load distribution. Because IPVS continuously forwards datagrams to the existing active connection entry on the DNS Slave (IP: 10.164.0.25) to avoid additional routing computation, the SED algorithm consistently ranks at the top in raw speed, despite failing to achieve a consistent equilibrium across the server cluster.

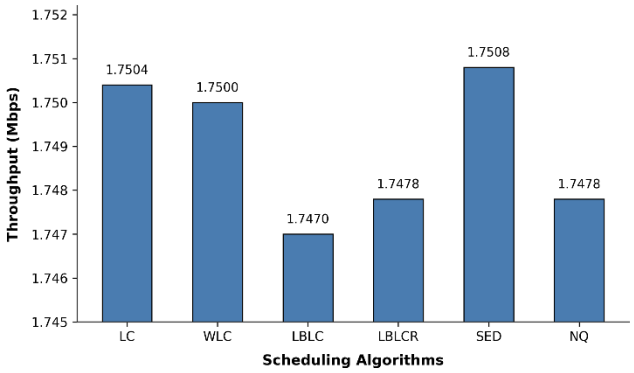


Fig. 5. Throughput Comparison in Active Scheduling Algorithms

D. Evaluation of Passive Scheduling Algorithms

Among the passive (static) scheduling algorithms, Destination Hashing (DH) yielded the lowest measured batch completion time and the highest calculated throughput under the tested configuration. As summarized in Table 2, the DH algorithm recorded an average batch completion time of 1.0060 seconds and an average throughput of 1.7520 Mbps.

TABLE II. AVERAGE MEASUREMENT DATA FOR THE PARAMETERS OF THE PASSIVE SCHEDULING ALGORITHM

Algorithms	**Batch completion time (s)**	**Throughput (Mbps)**
Round Robin (RR)	1.0086	1.7500
Weighted Round Robin (WRR)	1.0082	1.7492
Destination Hashing (DH)	1.0060	1.7520
Source Hashing (SH)	1.0068	1.7502

The deterministic routing of the DH algorithm is governed by a static hash table mapping destination server addresses. When processing incoming DNS queries, IPVSADM computes a hash key to bind packet forwarding to the designated backend server. In this experimental setup, the measured efficiency of hash-based algorithms (DH and SH) is primarily driven by the single-client test topology. Because queries originated deterministically from a single source address toward a fixed destination, the hashing function consistently directed traffic to the same backend target without recalculating routing weights or introducing scheduling overhead. Consequently, these metrics reflect the deterministic properties of static hashing under concentrated single-source workloads rather than a generalized superiority across multi-client environments.

E. Comparative Analysis of Ten Scheduling Algorithms and Baseline

A comprehensive performance comparison of all 10 evaluated LVS scheduling algorithms, alongside the baseline scenario, is presented in Table 3.

TABLE III. AVERAGE MEASUREMENT DATA FOR TEN SCHEDULING ALGORITHMS

Algorithms	**Batch completion time (s)**	**Throughput (Mbps)**
Round Robin (RR)	1.0086	1.7500
Weighted Round Robin (WRR)	1.0082	1.7492
Least Connection (LC)	1.0084	1.7504
Weighted Least Connection (WLC)	1.0082	1.7500
Locality-Based Least Connection (LBLC)	1.0090	1.7470
Locality-Based Least Connection with Replication (LBLCR)	1.0084	1.7478
Destination Hashing (DH)	1.0060	1.7520
Source Hashing (SH)	1.0068	1.7502
Shortest Expected Delay (SED)	1.0068	1.7508
Never Queue (NQ)	1.0086	1.7478
Without load balancing	1.0106	1.7472

Under the tested single-client configuration, the DH algorithm produced the lowest measured batch completion time (1.0060 seconds) and the highest calculated throughput

(1.7520 Mbps). Evaluating these configurations against the baseline scenario indicates that introducing the LVS director mitigates the direct traffic concentration experienced when queries are sent unmanaged to a single receiving server. While the unmanaged baseline incurred delay due to localized node saturation, the load-balanced configurations managed packet flow through structured kernel-level forwarding mechanisms. Although the absolute differences among algorithms are on the order of several milliseconds, these empirical variations demonstrate how protocol handling and scheduling constraints govern processing performance under intensive DNS query loads.

IV. Conclusions

This study demonstrates that while IP Tunneling-based load balancing (LVS-TUN) provides a viable framework for routing DNS traffic, scheduler efficacy is largely determined by transport-level protocols and client topology. Under the evaluated single-client query-flooding configuration, the Destination Hashing (DH) algorithm produced the lowest measured batch completion time (1.0060 s) and the highest calculated throughput (1.7520 Mbps); however, this performance reflects the deterministic routing advantages of static hashing from a single source rather than a generalized algorithmic superiority. Crucially, dynamic algorithms such as Shortest Expected Delay (SED) and Never Queue (NQ) exhibited notable operational anomalies under connectionless UDP traffic, where rapid query bursts prompted IPVS to reuse active connection entries within the timeout window, paradoxically accelerating individual batch completion times at the expense of severe single-server load accumulation. Consequently, future work should prioritize distributed, multi-client testbeds to eliminate single-source hash bias and integrate automated failover mechanisms, such as Keepalived, alongside Direct Routing architectures to comprehensively assess DNS cluster resilience.

References

[1] A. Nag *et al.*, "Exploring the applications and security threats of Internet of Thing in the cloud computing paradigm: A comprehensive study on the cloud of things," *Trans. Emerg. Telecommun. Technol.*, vol. 35, no. 4, Apr. 2024, doi: 10.1002/ett.4897.

[2] A. Mughaid, I. Obeidat, L. Abualigah, S. Alzubi, M. S. Daoud, and H. Migdady, "Intelligent cybersecurity approach for data protection in cloud computing based Internet of Things," *Int. J. Inf. Secur.*, vol. 23, no. 3, pp. 2123–2137, Jun. 2024, doi: 10.1007/s10207-024-00832-0.

[3] M. M. Nair and A. K. Tyagi, "AI, IoT, blockchain, and cloud computing: The necessity of the future," in *Distributed Computing to Blockchain*, Elsevier, 2023, pp. 189–206. doi: 10.1016/B978-0-323-96146-2.00001-2.

[4] R. Desnita, W. Amelia, B. Sunaryo, R. Rahmayanti, and S. 'Asyarina Ramadani, "An IoT-Based Real-Time Monitoring System for Intradialytic Cycling in Hemodialysis Patients," *Ingénierie des systèmes d Inf.*, vol. 31, no. 1, pp. 27–35, Jan. 2026, doi: 10.18280/isi.310104.

[5] A. Heideker and C. Kamienski, "Network Queuing Assessment: A Method to Detect Bottlenecks in Service Function Chaining," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 4, pp. 4650–4661, Dec. 2022, doi: 10.1109/TNSM.2022.3188932.

[6] M. Golec, G. K. Walia, M. Kumar, F. Cuadrado, S. S. Gill, and S. Uhlig, "Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions," *ACM Comput. Surv.*, vol. 57, no. 3, pp. 1–36, Mar. 2025, doi: 10.1145/3700875.

[7] M. Singh, P. Bhardwaj, R. Bhardwaj, and S. Narayan, "Advancing Scalability and Efficiency in Distributed Network Computing Through Innovative Resource Allocation and Load Balancing Strategies," 2024, pp. 722–740. doi: 10.1007/978-3-031-67195-1_80.

[8] S. M. Mirmohseni, C. Tang, and A. Javadpour, "Optimizing network performance through cloud load balancing and virtualization in software-defined networking," *Computing*, vol. 108, no. 3, p. 45, Mar. 2026, doi: 10.1007/s00607-026-01635-y.

[9] H. Wen *et al.*, "A TX_XDP-Enhanced and Fallback-Aware Optimization Strategy for LVS-Based L4 Load Balancing," in *Proceedings of the 2025 10th International Conference on Cloud Computing and Internet of Things*, New York, NY, USA: ACM, Nov. 2025, pp. 92–98. doi: 10.1145/3785520.3785533.

[10] W. Lin, "Tunneling and VPN," in *Trends in Data Protection and Encryption Technologies*, Cham: Springer Nature Switzerland, 2023, pp. 149–154. doi: 10.1007/978-3-031-33386-6_26.

[11] A. Veichtlbauer, C. Praschl, L. Gaisberger, G. Steinmaurer, and T. I. Strasser, "Toward an Effective Community Energy Management by Using a Cluster Storage," *IEEE Access*, vol. 10, pp. 112286–112306, 2022, doi: 10.1109/ACCESS.2022.3216298.

[12] M. Kumari, M. P. Singh, and A. K. Singh, "A latency sensitive and agile IIoT architecture with optimized edge node selection and task scheduling," *Digit. Commun. Networks*, vol. 12, no. 3, pp. 482–494, Mar. 2026, doi: 10.1016/j.dcan.2025.04.012.

[13] S. Tang *et al.*, "A Survey on Scheduling Techniques in Computing and Network Convergence," *IEEE Commun. Surv. Tutorials*, vol. 26, no. 1, pp. 160–195, 2024, doi: 10.1109/COMST.2023.3329027.

[14] H. Doroud, T. Wiese, F. Erlacher, and F. Dressler, "Work Balancing vs. Load Balancing for Network IDS Parallelization," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*, IEEE, Jun. 2023, pp. 488–493. doi: 10.1109/IWCMC58020.2023.10183026.

[15] P. Priyadarshini, P. S. Kiran, and A. Kumar, "Performance Study of TCP in Vehicular Ad Hoc Networks," 2027, pp. 93–115. doi: 10.1007/978-3-032-22249-7_6.

[16] J. J. López Escobar, R. P. Díaz-Redondo, and F. Gil-Castiñeira, "Unleashing the power of decentralized serverless IoT dataflow architecture for the Cloud-to-Edge Continuum: a performance comparison," *Ann. Telecommun.*, vol. 79, no. 3–4, pp. 135–148, Apr. 2024, doi: 10.1007/s12243-023-01009-x.

[17] S. M. Mousavi Jalali and F. Rezaei, "IoT data caching and scheduling in heterogeneous mobile networks by exploiting mobility and transient data characteristics," *Peer-to-Peer Netw. Appl.*, vol. 18, no. 4, p. 167, Jul. 2025, doi: 10.1007/s12083-025-01940-x.

[18] A. Tawab Akrami, G. Attigeri, and M. C. Belavagi, "Comprehensive Review of Collaborative Data Caching in Edge Computing," *IEEE Access*, vol. 13, pp. 71408–71431, 2025, doi: 10.1109/ACCESS.2025.3563407.

[19] M. D. Tache (Ungureanu), O. Păscuțoiu, and E. Borcoci, "Optimization Algorithms in SDN: Routing, Load Balancing, and Delay Optimization," *Appl. Sci.*, vol. 14, no. 14, p. 5967, Jul. 2024, doi: 10.3390/app14145967.

[20] A. Simpson, M. Alshaali, W. Tu, and M. R. Asghar, "Quick UDP Internet Connections and Transmission Control Protocol in unsafe networks: A comparative analysis," *IET Smart Cities*, vol. 6, no. 4, pp. 351–360, Dec. 2024, doi: 10.1049/smc2.12083.

[21] D. Wu, Y. Zhang, B. Liu, X. Li, E. Alowaisheq, and H. Duan, "Exploring and Analyzing Cross Layer DoS Attack Against UDP-based Services on Linux," in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: ACM, Nov. 2025, pp. 4559–4573. doi: 10.1145/3719027.3744878.