2025 International Conference on Computing, Networking and Communications (ICNC) | 979-8-3315-2096-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICNC64010.2025.10993768

Optimizing CDN Architectures: Multi-Metric Algorithmic Breakthroughs for Edge and Distributed Performance

Md Nurul Absur*, Sourya Saha*, Sifat Nawrin Nova†, Kazi Fahim Ahmad Nasif‡, Md Rahat Ul Nasib§
*Department of Computer Science, City University of New York, New York, USA
†Department of Computer Science, Chalmers University of Technology, Gothenburg, Sweden
‡College of Computing and Software Engineering, Kennesaw State University, Georgia, USA
§Samsung Austin Research Center, Austin, Texas, USA
Emails: mabsur@gradcenter.cuny.edu, ssah42@gradcenter.cuny.edu, esifatn@chalmers.se, knasif@students.kennesaw.edu, nasib131@gmail.com

***Abstract*—A Content Delivery Network (CDN) is a powerful system of distributed caching servers that aims to accelerate content delivery, like high-definition video, IoT applications, and ultra-low-latency services, efficiently and with fast velocity. This has become of paramount importance in the post-pandemic era. Challenges arise when exponential content volume growth and scalability across different geographic locations are required. This paper investigates data-driven evaluations of CDN algorithms in dynamic server selection for latency reduction, bandwidth throttling for efficient resource management, real-time Round Trip Time analysis for adaptive routing, and programmatic network delay simulation to emulate various conditions. Key performance metrics, such as round-trip time (RTT) and CPU usage, are carefully analyzed to evaluate scalability and algorithmic efficiency through two experimental setups: a constrained edge-like local system and a scalable FABRIC testbed. The statistical validation of RTT trends, alongside CPU utilization, is presented in the results. The optimization process reveals significant trade-offs between scalability and resource consumption, providing actionable insights for effectively deploying and enhancing CDN algorithms in edge and distributed computing environments.**

***Index Terms*—Content Delivery Network, Video Streaming, FABRIC Testbed, Statistical Performance Analysis, Multi-metric Analysis.**

I. Introduction

The importance of video streaming has grown significantly in the digital age, becoming a dominant force in internet traffic and user engagement. Recent statistics indicate that video accounts for over 82% of the traffic on the internet [1]. The shift to remote work, virtual learning, and digital entertainment has not only increased this demand. Still, it has also created an urgent necessity to address the challenges of delivering high-quality video content reliably and at scale. This necessitates innovative network management solutions to accommodate the growing and diverse user demands.

The unprecedented data volume and complexity growth presents significant challenges for existing Content Delivery Networks (CDNs). Traditional CDN architectures, optimized for static and moderately dynamic content, are increasingly strained by the demands of real-time, ultra-high-definition video and other bandwidth-intensive applications. Achieving scalability across geographically dispersed servers and efficiently handling dynamic content remain critical bottlenecks [2]. To ensure an optimal Quality of Experience (QoE), modern CDNs must address several urgent issues: managing the surging content volume, ensuring seamless scalability across diverse locations, and delivering ultra-low latency under varying network conditions [3]. Additionally, the shift toward localized and personalized content and the growing prevalence of interactive and immersive experiences necessitates innovative CDN design and management approaches [4]. *Thus, a scalable and reliable CDN configuration scheme is needed to consider advanced dynamic server selection, bandwidth throttling, delay modifications in the presence of essential performance metrics, and advanced statistical multi-metric decision outcomes.*

Our contribution addresses the current limitations of Content Delivery Networks (CDNs) by introducing dynamic algorithms and a comprehensive multi-metric analysis framework. These advancements aim to improve video streaming performance by reducing latency, enhancing adaptive streaming capabilities, and optimizing resource utilization across distributed systems. Our work focuses on key challenges in CDN performance management, ensuring better scalability, load balancing, and service quality. The key innovations of our approach include:

- The development of dynamic multi-algorithm frameworks aims to optimize the performance of Content Delivery Networks (CDNs).
- Implement programmatic modifications to network delays in order to simulate edge and distributed environments for scalability testing.
- A multi-metric statistical analysis aims to overcome the limitations of performance management that rely on a single metric.
- The dataset, acquired from real-life simulations, is shared for open use to foster research and development.

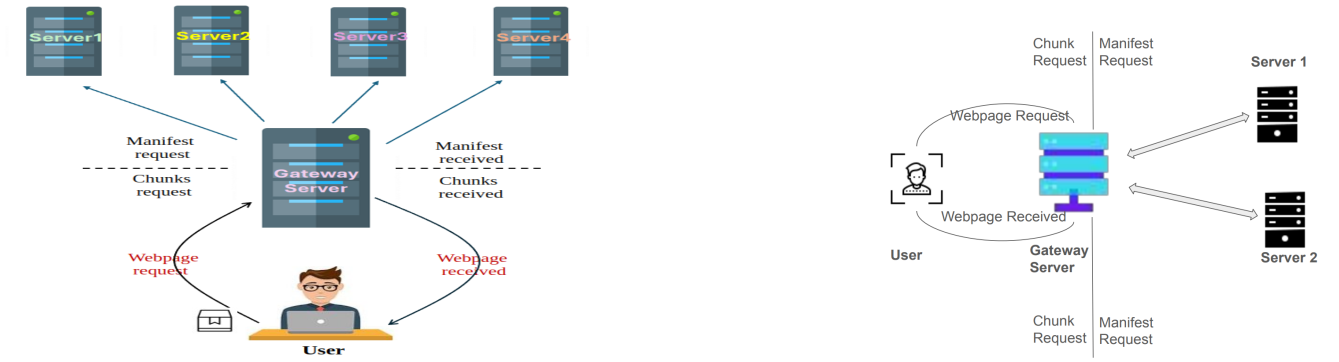
The rest of the paper is organized as follows. Section II discusses the related work in this area. Section III introduces our solution approach with different configurations, and Section IV discusses performance evaluation. Finally, Section V concludes the paper addressing future works.

II. Related Works

Content Delivery Network (CDN) optimization is an active area of research that has garnered significant attention in recent years. The applications of CDN are pretty diverse, and the variability of CDN management has made this use case ubiquitous in terms of system optimization, network control, resource efficiency, and so on. Software-Defined Networking (SDN) is gaining substantial attention to address scalability and cost challenges in Content Delivery Networks (CDNs). Yang et al. [5] have proposed a Software-Defined Content Distributed Network (SDCDN) solution. Another alternative model is implemented by combining Peer-to-Peer (P2P) and CDN together to achieve optimal content experience [6]. Despite this, the flexibility and global reach offered by CDNs continue to make them the preferred choice for video streaming and real-time multimedia services [7].

To address the shortcomings of CDNs, Zhou et al. use a load balancing algorithm based on playback volume and the remaining resources of the server to reduce latency and enhance Quality of Experience (QoE) [8]. Vaton et al. measure web browsing performance using different CDN to decrease loading times by up to 400 percent [9]. In [10], a new anycast distribution method is proposed that incorporates spatial locality of the popularity of the content and addresses large-scale scalability. Recent progress has also centered

Authorized licensed use limited to: CUNY- Graduate Center. Downloaded on January 26,2026 at 21:22:12 UTC from IEEE Xplore. Restrictions apply.



(a) Topology Overview of Testbed CDN Architecture

(b) Local Edge CDN Architecture

Fig. 1: CDN Setup Across Different Configurations

on refining content delivery networks (CDNs) by implementing P2P-CDN architecture [11] and integrating multi-source streaming to provide an exceptional viewing experience [12].

Working on a CDN in a testbed environment aims to replicate a scalable, twin-like large network system and measure network variability. This approach is becoming popular in the video content and streaming research community. In [13], the authors introduced a CDN emulator to carry out discrete event simulation and considered resource consumption a performance metric. The FABRIC testbed is an innovative and rapidly emerging platform that comprises a wide range of use cases and significant computational capabilities [14]. This infrastructure offers diverse computational resources, and researchers are leveraging its power to address many complex network management challenges through various applications [15] [16]. Some works use machine learning and statistical methods to make CDNs more ubiquitous, better performing, scalable, and applicable to the industry [17], [18].

III. System Model & Problem Formulation

To tackle CDN architecture, we formulated problem statements and sought optimal solutions by implementing two types of configurations. The first configuration consisted of a large-scale distributed CDN server setup, which emulated high-end computation and scalability within the FABRIC testbed. The second configuration featured an edge device with minimal system requirements, representing a small CDN setup that can still be utilized in various applications to address cost constraints.

A. Testbed Setup For Scalable Implementation

The architecture of our testbed Content Delivery Network (CDN) implementation, as illustrated in Figure of 1a, underscores the pivotal role of distributed servers—Server1, Server2, Server3, and Server4—in facilitating the communication process. Central to this architecture is the Gateway Server, which orchestrates interactions between the user and the distributed servers. When a user requests a webpage, the initial query is directed to the Gateway Server, which issues manifest requests to the various distributed servers. The manifest, containing essential metadata regarding the content chunks stored across these servers, enables the Gateway Server to determine the appropriate servers for subsequent chunk requests efficiently. These content chunks, representing optimized segments of the webpage, are then retrieved and delivered to the user by the designated servers. This streamlined process maximizes throughput and minimizes latency, exemplifying the scalability of our CDN model, which is designed to emulate real-world, large-scale server distribution across diverse geographical locations.

B. Local Edge-like System Setup

In Figure 1b, a CDN architecture is deployed on local devices to emulate edge systems, enabling performance evaluation under dynamic, real-time scenarios. Upon a user's webpage request, the Gateway Server initiates manifest requests to multiple servers (e.g., Server 1 and Server 2), which respond with the necessary metadata. The Gateway Server then fetches specific content chunks, ensuring efficient delivery while minimizing latency and enhancing user experience. To optimize performance further, the system diligently monitors round-trip time (RTT) values, dynamically selecting servers with lower RTTs to maintain high reliability and throughput, and effectively simulates real-world CDN operations.

C. Advance CDN Setup Algortihms

Four advanced algorithms are used to make the overall CDN setup more dynamic, adaptable to throttled segmentation, and to address network delay issues. These algorithms measure CDN performance using various metrics and establish which setup is better suited for scalability.

Algorithm 1 demonstrates an implementation for streaming video using the DASH protocol while handling HTTP redirects. It initializes a DASH player, intercepts HTTP requests to process media segments dynamically, constructs redirect URLs, and updates the requests accordingly, all while logging essential details.

Algorithm 2 processes an MPD file from a specified URL, retrieves metadata such as the video ID and segment count, and downloads the media segments while implementing throttling to mimic real-world network conditions. The utility function handles file writing correctly while adhering to throttling restrictions.

Algorithm 3 pings a list of servers, measures the round-trip time (RTT) for each server, and selects the one with the lowest RTT for directing clients. If no valid RTT is found, a random server will be chosen as a fallback option.

Algorithm 4 adjusts network delay using Linux Traffic Control by introducing a random delay between 200 and 800 milliseconds in a continuous loop. It verifies these changes and pauses for a duration determined by a Poisson distribution.

D. Dataset Collection For Multimetric Correlation Analysis

Relying on a single metric is insufficient for making scalability decisions in *CDN*. To support this, we gathered extensive data using various real-life mimicking configurations: *four servers, eight servers, and twelve servers*. We collect a vast number of *RTT & CPU consumption* data to find a pattern and make trade-offs between cost management and scalability decisions. This dataset can be found at `https://github.com/Tomxx7/FABRIC_RTT_CPU.git`

Authorized licensed use limited to: CUNY- Graduate Center. Downloaded on January 26,2026 at 21:22:12 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: DASH Video Streaming with HTTP Redirects

```
Data: videoElement, initialMpdUrl
Result: Optimized media segment handling
Function DASHStreaming(videoElement, initialMpdUrl):
    $player \leftarrow$ InitializePlayer(videoElement, initialMpdUrl)
    OverrideXMLHttpRequest()
    $player$.on("FRAGMENT_LOADING", ProcessSegment)
Function ProcessSegment(event):
    if IsMediaSegment(event.request) then
        $fileName \leftarrow$ ExtractFileName(event.request.url)
        $redirectUrl \leftarrow$ ConstructRedirectUrl(fileName)
        UpdateRequest(event.request, redirectUrl)
        LogRequest(fileName)
    end
Function InitializePlayer(video, url):
    return new Player(video, url)
Function OverrideXMLHttpRequest():
    Intercept HTTP requests to log and modify as needed
Function IsMediaSegment(request):
    return request.url.endswith(".mp4")
Function ExtractFileName(url):
    return url.split("/")[-1]
Function ConstructRedirectUrl(fileName):
    return http://cdn.example.com/ + fileName
Function UpdateRequest(request, url):
    request.url $\leftarrow$ url
Function LogRequest(fileName):
    Output log information about the request
```

Algorithm 2: Download and Parse MPD with Throttled Segment Download

```
Data: php_url (PHP script URL)
Result: Total media segments downloaded
Parse MPD from php_url; extract video ID, segment count
Download initial segment
for each segment in MPD do
    Construct URL, download with throttle
end
return Number of segments
Function save_and_throttle_downloadresponse,
  filename:
    Write response to file with throttle
end
```

Algorithm 3: Select Server Based on Ping RTT

```
Result: Redirect to server with the lowest RTT
Initialize min RTT to $\infty$, selected server to null
Function getPingRTT(serverIP):
    Ping server, extract RTT
    return RTT or high value on failure
end
foreach server in list do
    $RTT \leftarrow$ getPingRTT(Extract IP(server URL))
    if $RTT < min\ RTT$ then
        min RTT $\leftarrow$ RTT; update selected server
    end
end
if no valid RTT then
    Select random server
end
Redirect client to selected server with video filename
```

Algorithm 4: Dynamic Network Delay Modification with Linux Traffic Control

```
Result: Continuously updated network delay
while True do
    Generate random delay (200-800ms)
    Remove existing delay; apply new delay
    Verify and print success of removal/addition
    Set and print sleep time (Poisson distribution)
    Sleep for the calculated duration
end
```

Algorithm 5 pings a list of servers multiple times (num_pings) to measure **Round Trip Time (RTT)**, modifies the RTT with a Poisson-distributed random value, and logs the results (including a timestamp) to a CSV file (ping_results1000.csv). It captures RTT for each server in real-time, handles ping failures gracefully, and delays each round by one second.

Algorithm 5: Ping Servers and Record RTT with Poisson Modification

```
Data: servers, num_pings, output_file
Result: CSV file with timestamps and RTT for each server
Function ping_server(ip_address):
    Execute ping command and extract RTT
    Add Poisson random value (mean, $\lambda = 200$ms)
    return Modified RTT or None
Function record_results(servers, output_file):
    Write headers (servers) to output_file
    for $i \leftarrow 1$ to num_pings do
        Initialize row with timestamp
        foreach server in servers do
            Append RTT or N/A to row
        end
        Write row to output_file; Wait 1 second
    end
record_results(servers, output_file)
```

Algorithm 6 fetches **CPU utilization** data from Prometheus by querying total and active CPU times, calculates the utilization percentage, and adds a Poisson-distributed random value to simulate variability. The data and timestamp are logged into a CSV file every 2 seconds for 1500 iterations. It handles errors gracefully and ensures missing values are marked as "N/A" in the output. Prometheus is an open-source, research-driven monitoring system optimized for time-series data collection and analysis in distributed and cloud-native environments [19]. It provides a robust pull-based metric scraping mechanism. It leverages PromQL, a powerful query language, to enable advanced metric correlations and real-time insights, supporting scalability and precision in dynamic systems.

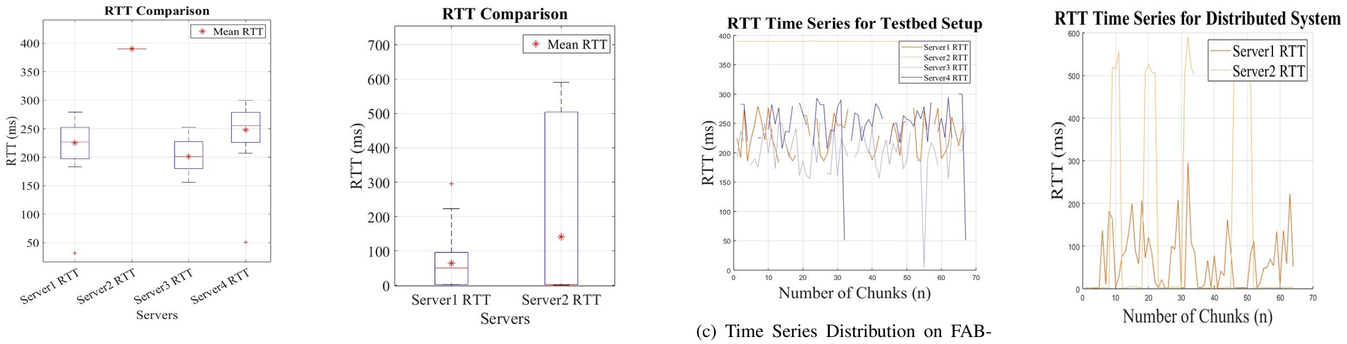
IV. Experiment Results

A. Scalability

To explore more effective scalability setups for a CDN system, we examine four distinct servers in the FABRIC testbed that simulate large-scale CDN servers, along with two servers intended for a smaller edge-like CDN configuration.

In Figure 2, we evaluate the Round-Trip Time (RTT) performance across two distinct setups. This analysis will feature two visualizations: a boxplot that illustrates the distribution and variability of RTT values and a time series distribution that highlights factors such as network congestion, hardware limitations, and geographical distances between nodes. These visualizations will provide valuable insights into the latency performance of each server, aiding in the optimization of server selection for improved network efficiency and reduced latency.

Authorized licensed use limited to: CUNY- Graduate Center. Downloaded on January 26,2026 at 21:22:12 UTC from IEEE Xplore. Restrictions apply.



(a) RTT Distribution on FABRIC

(b) RTT Distribution on Edge

(c) Time Series Distribution on FABRIC

(d) Time Series Distribution on Edge

Fig. 2: Performance Analysis Based On RTT Across Setups

Algorithm 6: CPU Utilization Logging with Prometheus

```
Data: PROMETHEUS_URL, TOTAL_CPU_QUERY,
      ACTIVE_CPU_QUERY
Result: CSV file of CPU utilization percentages
Function fetch_cpu_utilization():
    Query Prometheus for total and active CPU times
    Compute utilization: $\frac{\text{Active\_CPU}}{\text{Total\_CPU}} \times 100$
    Add Poisson noise ($\lambda = 30$)
    return utilization per instance
Function write_to_csv(writer, data, timestamp):
    Initialize header if first write
    Write timestamp and utilization (or N/A for missing
      values)
Main Procedure:
Open CSV file
for $i \leftarrow 1$ to 1500 do
    Record current_time and fetch utilizations
    Write data to CSV
    Sleep for 2 seconds
end
```

Figure 2b illustrates the inherent challenges associated with edge-based or small-scale CDN setups, where Server 2 consistently demonstrates high variability and instability in round-trip time (RTT), significantly affecting performance and user experience. Even within the more controlled FABRIC testbed shown in Figure 2a, which includes four servers, Server 2 fails, resulting in an infinite RTT that indicates its inability to respond. This observation reveals a critical limitation: even a single point of failure can jeopardize overall service reliability despite the presence of multiple servers. While edge setups may present cost advantages, their lack of robustness and greater susceptibility to failures render them less suitable for real-world applications that require uninterrupted and reliable content delivery. These findings emphasize prioritizing resilience and stability in CDN architectures to ensure seamless user experiences.

Figures 2c and 2d illustrate a comparative Round Trip Time (RTT) analysis between Testbed and Distributed System configurations, highlighting their respective advantages and limitations. The Testbed setup demonstrates stable RTT across four servers, with Server 2's absence resulting in infinite RTT, highlighting vulnerabilities even in controlled environments. This setup is ideal for high-reliability applications like finance and healthcare. In contrast, the edge setup shows significant RTT variability, particularly with Server 2, which is affected by geographic and traffic fluctuations. While this variability challenges latency-sensitive applications, it offers valuable insights for stress-testing CDNs, helping identify failure points and optimization strategies. The Testbed ensures consistency, while the Distributed System provides crucial data for improving CDN scalability and resilience.

B. Multi Metric Trade-off Analysis

Relying on a single metric, such as round-trip time (RTT) or CPU utilization, can lead to misleading conclusions about system performance and scalability. For instance, while CPU utilization may indicate a stable average of around 30%, it can obscure increasing RTT, which points to rising latency as server count increases. Conversely, RTT highlights scalability issues due to coordination delays but does not specify whether computational bottlenecks or network inefficiencies cause these. Analyzing both metrics reveals that although CPU load may stabilize with more servers, communication overhead significantly hampers network performance. This comprehensive approach is vital for identifying the actual limitations of scalability and opportunities for optimization.

Figure 3a shows a clear trend of increasing latency with more servers. The four-server setup has the lowest median latency (about 220 ms) and the least variability. In contrast, the twelve-server setup displays a broader range (approximately 240–280 ms), highlighting the challenges of managing distributed coordination. This increased RTT variance indicates that maintaining consistent network performance becomes more complex as server count rises.

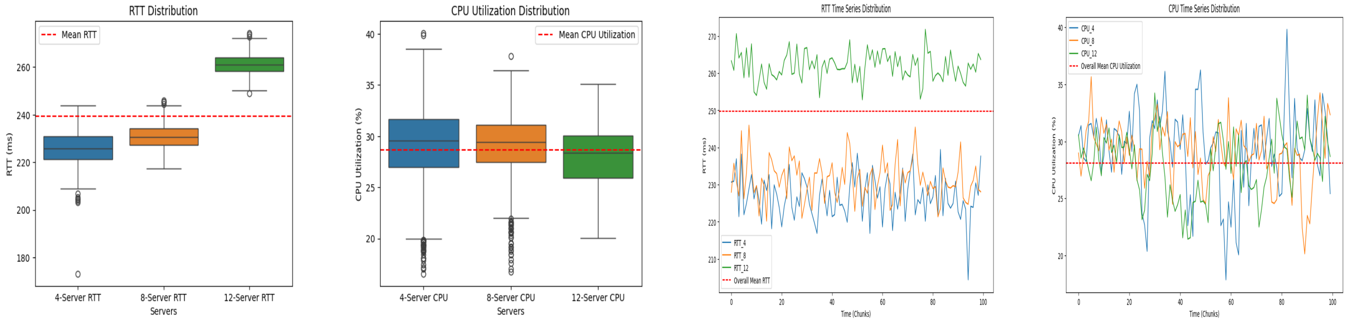
Figure 3b shows that the median utilization across all setups remains stable at approximately 30%. Notably, the 4-server configuration exhibits the most comprehensive interquartile range and shows outliers below 20%, highlighting some inefficiencies during specific intervals. In contrast, the 8-server and 12-server setups demonstrate narrower ranges, suggesting improved load balancing, though this may come at the cost of consistency under heavier loads.

Figure 3c shows that latency rises as the server count increases. The four-server setup has the lowest average RTT at about 230 ms with minimal fluctuations. The eight-server setup averages around 240 ms, while the twelve-server configuration reaches approximately 270 ms. This trend highlights that additional servers lead to communication overhead and coordination delays, negatively impacting latency as setups scale.

Figure 3d shows that The CPU utilization time series fluctuates around a mean value of approximately 30% utilization for all setups. The 4-server setup exhibits higher variance with periodic spikes, suggesting an uneven load distribution. The 8-server and 12-server setups show more stable utilization but introduce occasional dips, possibly due to load redistribution inefficiencies. Including the mean CPU utilization as a reference highlights that while the system remains stable on average, transient conditions can lead to processing inefficiencies or bottlenecks.

The analysis reveals a complex relationship between Round Trip Time (RTT) and CPU utilization as server configurations scale. While increasing the number of servers reduces CPU variability and promotes more effective load balancing, it also leads to higher RTT due to increased inter-server coordination. For example, the 4-server setup exhibits higher CPU variability and frequent load spikes,

Authorized licensed use limited to: CUNY- Graduate Center. Downloaded on January 26,2026 at 21:22:12 UTC from IEEE Xplore. Restrictions apply.



(a) RTT Distribution Across Configurations

(b) CPU Utilization Distribution Across Configurations

(c) RTT Time Series Across Configurations

(d) CPU Utilization Time Series for Configurations

Fig. 3: Multi-Metric Analysis of RTT and CPU Utilization Across Server Setups

whereas the 8-server and 12-server configurations demonstrate more excellent stability with occasional dips from load redistribution inefficiencies. Notably, despite similar CPU utilization levels, the 12-server setup experiences an average RTT of approximately 270 ms compared to about 230 ms for the 4-server setup. These results emphasize the trade-offs between processing efficiency and network latency, highlighting the need for careful resource balancing to optimize scalability and performance in distributed systems.

V. Conclusion

This paper presents dynamic algorithms and multi-metric analysis to tackle significant challenges in Content Delivery Networks (CDNs), emphasizing scalability, load balancing, and latency optimization. The study uncovers critical trade-offs between increasing server capacity and maintaining performance by examining round-trip time (RTT) and CPU utilization across various server configurations. Results obtained from FABRIC and edge environments indicate that larger server setups enhance load stability and incur greater RTT overhead due to inter-server coordination. These findings emphasize the limitations of single-metric approaches and highlight the importance of multi-metric frameworks in capturing the complexities of system dynamics. Future research will integrate additional metrics, including bandwidth and memory utilization, to enrich the analysis and provide actionable insights into network performance. Furthermore, exploring reinforcement learning models for dynamic resource management will facilitate intelligent decision-making under varying network loads. This research establishes a foundation for scalable, efficient, and adaptive CDN architectures that effectively meet real-world demands.

References

[1] D. Kim, D.-W. Seo, and M. Choi, “Edge caching and computing of video chunks in multi-tier wireless networks,” *J. Netw. Comput. Appl.*, vol. 226, no. C, Jul. 2024.

[2] T. Plagemann, V. Goebel, A. Mauthe, L. Mathy, T. Turletti, and G. Urvoy-Keller, “From content distribution networks to content networks — issues and challenges,” *Computer Communications*, vol. 29, no. 5, pp. 551–562, 2006, networks of Excellence. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366405002094

[3] D. Priyanka and Channakrishnaraju, “A cutting-edge approach to elevate the quality of service in cloud based content delivery network,” in *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, vol. 1, 2024, pp. 1–7.

[4] Y. Ma, Y. Wu, J. Li, and J. Ge, “Apcn: A scalable architecture for balancing accountability and privacy in large-scale content-based networks,” *Information Sciences*, vol. 527, pp. 511–532, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025519300659

[5] H. Yang, H. Pan, and L. Ma, “A review on software defined content delivery network: A novel combination of cdn and sdn,” *IEEE Access*, vol. 11, pp. 43 822–43 843, 2023.

[6] S. M. Y. Seyyedi and B. Akbari, “Hybrid cdn-p2p architectures for live video streaming: Comparative study of connected and unconnected meshes,” in *2011 International Symposium on Computer Networks and Distributed Systems (CNDS)*, 2011, pp. 175–180.

[7] B. Kara and G. Simon, “Power efficient multi-cdn communication over content steering server,” in *Proceedings of the 15th ACM Multimedia Systems Conference*, ser. MMSys ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 478–484. [Online]. Available: https://doi.org/10.1145/3625468.3652196

[8] S. Zhao, F. Zhou, Z. Yin, Z. Bi, and C. Ju, “Cdn load balancing algorithm based on playback volume and server remaining resources,” in *2023 9th International Conference on Computer and Communications (ICCC)*, 2023, pp. 266–270.

[9] A. Saverimoutou, B. Mathieu, and S. Vaton, “Web view: A measurement platform for depicting web browsing performance and delivery,” *IEEE Communications Magazine*, vol. 58, no. 3, pp. 33–39, 2020.

[10] C. Kato and N. Kamiyama, “Designing server sets for anycast cdn using genetic algorithm,” in *ICC 2024 - IEEE International Conference on Communications*, 2024, pp. 3010–3015.

[11] R. Farahani, E. Çetinkaya, C. Timmerer, M. Shojafar, M. Ghanbari, and H. Hellwagner, “Alive: A latency- and cost-aware hybrid p2p-cdn framework for live video streaming,” *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1561–1580, 2024.

[12] X. Yuan, L. Pu, J. Shi, Q. Gong, and J. Xu, “Muster: Multi-source streaming for tile-based 360° videos within cloud native 5g networks,” *IEEE Transactions on Mobile Computing*, vol. 22, no. 11, pp. 6616–6632, 2023.

[13] H.-L. La, A.-T. N. Tran, M. Yoshimi, T. Nakajima, and N. Thoai, “Cdnet: A content delivery network emulator,” in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021, pp. 1–7.

[14] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K.-C. Wang, T. Lehman, and P. Ruth, “Fabric: A national-scale programmable experimental network infrastructure,” *IEEE Internet Computing*, vol. 23, no. 6, pp. 38–47, 2019.

[15] P. Ruth, I. Baldin, K. Thareja, T. Lehman, X. Yang, and E. Kissel, “Fabric network service model,” in *2022 IFIP Networking Conference (IFIP Networking)*, 2022, pp. 1–6.

[16] X. Yang, E. Kissel, A. Essiari, L. Zhang, T. Lehman, I. Monga, P. Ruth, K. Thareja, and I. Baldin, “Fabfed: Tool-based network federation for testbed of testbeds - paradigm and practice,” in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2024, pp. 01–07.

[17] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, “Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2499–2508.

[18] J. H. Parikh and P. A. Bryant, “Content delivery network management system and method,” Patent US11 659 015B2, May 23, 2023, u.S. Patent. [Online]. Available: https://patents.google.com/patent/US11659015B2/en

[19] P. Authors, “Prometheus: Open-source monitoring and alerting toolkit,” https://prometheus.io/, 2024, accessed: 2024-11-27.

Authorized licensed use limited to: CUNY- Graduate Center. Downloaded on January 26,2026 at 21:22:12 UTC from IEEE Xplore. Restrictions apply.