ACTA TECHNICA JAURINENSIS
Vol. XX, No. Y, pp. ZZ-ZZ, 2025
10.14513/actatechjaur.00804

Research Article

CSS-LSTM: An Intelligent Caching Strategy in NDN Clusters

Ibrahim Khalil Douis[1, *], Hafida Bouziane[1], Abdallah Chouarfia[1]

[1]Department of Computer Science, University of Science and Technology of Oran
Oran, 31000, Algeria
*e-mail: ibrahimkhalil.douis@univ-usto.dz

Submitted: 15/06/2025 *Revised: 22/07/2025* *Accepted: 28/07/2025* *Published online: 27/09/2025*

Abstract: In-network caching is a key aspect of Named Data Networking (NDN) due to its significant impact on network performance and data delivery efficiency. As content demand, particularly for multimedia, continues to grow, deciding what to cache, where to store it, and when to remove it has become increasingly complex. This complexity highlights the need for more intelligent caching strategies in NDN. To address this challenge, we propose an intelligent caching strategy for NDN clusters called CSS-LSTM. This approach introduces a new data structure within the cluster's main router, called the Content Store Station (CSS), it controls the caching and eviction of material according to its age. Furthermore, a Long Short-Term Memory (LSTM) model is trained to optimize caching decisions, determining whether specific content should be stored and identifying the most suitable location within the store station. Experimental results show that the LSTM model achieves 90% accuracy in caching decisions. The CSS-LSTM strategy was compared with other approaches, specifically LCE, Random, and NECS, across two different scenarios using the cache hit ratio metric, demonstrating superior performance.

Keywords: In-network caching; Named Data Networking; Intelligent caching; LSTM; Clustering

I. INTRODUCTION

In recent years, internet usage has grown exponentially, particularly for multimedia content. In November 2024, YouTube recorded 72.8 billion visits, Facebook had 12.7 billion visits, and TikTok reached 2.49 billion visits [1]. The overwhelming demand for such content presents significant challenges in ensuring efficient delivery.

The underlying design of the current internet architecture, which is predicated on a host-to-host communication model, has difficulties. The user experience has changed dramatically with the shift from PC-based to mobile computing. However, the existing network architecture struggles to keep up with the rapid proliferation of mobile devices, leading to an increasing demand for content delivery solutions[2].

A new internet architecture called Named Data Networking (NDN) was created to get around the drawbacks of the conventional host-to-host communication paradigm. NDN recognizes and retrieves content by its name instead than depending on the location of the data. This approach enables various scalable communication mechanisms, such as automatic caching, which enhances network performance [3].

Interest packets and data packets are the two types of packets used in NDN communication. To request specific data, a customer sends an Interest Packet into the network, mostly with the name of the data. A matching Data Packet is returned to the customer via the same route as the Interest Packet once the requested material has been located inside the network.

The Data Packet primarily contains the content's name, the actual data, and a digital signature [4].

In NDN, the router consists of three main data structures (**Fig. 1** [5]):

- Content Storage (CS): This part is in charge of data object caching within the network. It can reply to Interest Packets on behalf of the original producer if it has already saved the necessary data. In essence, the CS serves as a temporary cache for the data objects that the router receives [6].
- Pending Interest Table (PIT): All of the interests that the router has forwarded but have not yet been fulfilled are listed in this table. The data name transmitted over the Internet and the

corresponding incoming and outgoing interfaces are recorded in each PIT entry. When an Interest Packet comes, the NDN router first scans the Content Store for matching Data Packet. The router returns the Data Packet via the interface from where the interest originated if the requested data is located. If not, the FIB is used to forward the Interest Packet to the following router [7].

- Forwarding Information Base (FIB): Only the first one is sent upstream to the data producer or producers by the FIB. A name prefix-based routing protocol is used to populate it, and each prefix can be linked to many output interfaces [7].

The paper is organized as follows: Section 2 presents an overview of the recently highlighted caching approaches used in NDN. Section 3 shows our proposed caching strategy for NDN clusters based on the CSS-LSTM model. This strategy involves dividing the network into multiple clusters and designating specific routers as caching nodes (store stations). A new data structure called Content Store Station (CSS) is implemented within the main router of each cluster.

The main router leverages an LSTM model to decide which router will store the incoming Data Packets (content).

In section 4, we evaluate the CSS-LSTM strategy by comparing it to the default NDN caching strategy LCE and Random strategy, as well as NECS, which employs network clustering and the cache hit ratio.

We analyze the performance of our proposed approach and highlight its advantages across two different scenarios. Finally, the paper is summarized in Section 5.

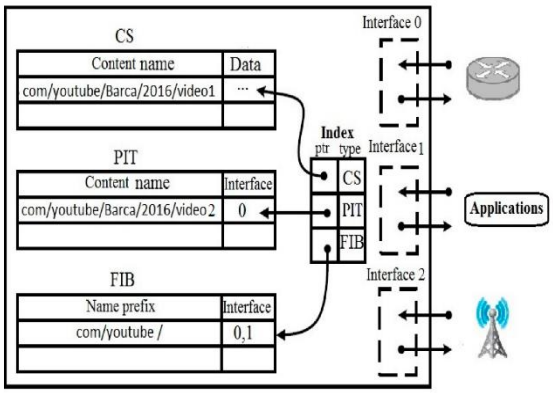

***Figure 1**.NDN router components*

II. RELATED WORK

Leave Copy Everywhere (LCE) [8] is the default caching policy in Named Data Networking (NDN). In accordance with this strategy, each router along the distribution path stores a copy of the material that the client asks (a data packet). The goal of this strategy is to speed up subsequent requests for the same material and raise the cache hit ratio. However, because it ignores the environment for storage limitations and the content qualities, such its popularity, LCE is regarded as a fundamental technique.

Least Recently Used (LRU) [9] is a cache replacement policy that controls the removal of content. While clearing out the least recently used content to make room for new data, it makes sure that recently visited content stays cached.

A more advanced cache replacement policy in NDN, known as Discard of Fast Retrievable Content (DFRC) [10], evaluates content retrieval time using information from the Forwarding Information Base table. This policy prioritizes the eviction of content that can be retrieved quickly, optimizing cache management based on retrieval performance.

Random Strategy [11] is designed to cache a single copy of content along the delivery path by randomly selecting one router. Its objective is to improve LCE's inefficient redundancy; however, it overlooks other important factors, such as the content's popularity and whether the chosen router is appropriate for caching this content.

The NECS [12] caching strategy aims to divide the network into multiple clusters, with each cluster designating a cluster head and two replacements. Using the Least Recently Used (LRU) policy as its cache replacement mechanism, the cluster head is in charge of caching incoming content in its Content Store to handle subsequent requests. In the event of the cluster head's failure, the replacements' job is to assume its duties.

The Efficient Content Caching and Eviction Priorities (CCEP) [13] strategy aims to identify the most suitable router for caching content along the transmission path, considering the content's popularity.

Once the content is cached on the selected router, eviction priorities are defined to manage potential replacements, factoring in content popularity and other key criteria. If the selected router lacks sufficient space, content with a high eviction priority will be removed to accommodate new data. This strategy modifies the standard structure of both Data Packet and Interest Packet by introducing additional fields. "Router with Maximum Caching Priority" and "Maximum Caching Priority for Content" are two new fields added to the Interest Packet. The name of the assigned router for caching is included in the Data Packet. By enabling content caching at a particular router along the download path, these improvements make sure that the router is ideally

situated in relation to the user and its capacity to manage subsequent requests.

An efficient caching policy based on a distributed content store [14] operates in a decentralized manner. It coordinates a node's caching system with those neighboring nodes. The objective is to store more popular content objects closer to the node. By maximizing the number of distinct popular content items saved locally and making sure that these objects are not duplicatedly cached on other nodes, each node helps to improve the overall caching effectiveness within its neighborhood.

Edge Caching Based on Collaborative Filtering for Heterogeneous ICN-IoT Applications approach [15] employs k-means clustering to partition the network [16], with the Silhouette coefficient method determining the optimal number of clusters. As a result, G clusters (CLs) are based on content history data, meaning clusters are organized according to content type. The edge nodes caches are divided into two categories: the first cache stores popular content within the cluster, while the second cache uses collaborative filtering to proactively store content, ensuring availability for future requests.

Mobile Edge Caching Using Deep Learning (DeepMEC) [17] introduces a learning-based caching strategy aimed at improving cache hit probability, reducing backhaul usage, and minimizing video content access delays. It uses a two-step prediction algorithm that anticipates the number of requests for each category after first classifying video content into four groups. This technique improves accuracy and lowers computational costs by removing the top 20% of video material. A caching strategy is also presented, which uses request count data and expected class labels to drive caching decisions at the Master Node. As a result, the base station will have a carefully selected list of content to store.

A GNN-based Proactive Caching Strategy in NDN Networks [18] describes a graph neural network-based algorithm, known as the GNN-GM algorithm, for optimizing cache placement. The method initially predicts the ratings viewers will give to videos they haven't seen yet using a graph neural network (GNN) model. The 'caching gain' related to storing is then regarded as the entire anticipated rating for a video. The algorithm aims to maximize this caching gain by proactively selecting which videos to cache.

For cache replacement, the algorithm ranks videos based on their cache gain. With higher-ranked videos replacing those with lower cache gain.

Graph Neural Network-Based Deep Reinforcement Learning for Intelligent Caching on SDN-based ICN [19] is a statistical model designed to analyze and generate users' content preferences. It represents the network into embedding vectors using a Graph Neural Network (GNN) and utilizes these vectors as a dataset (input data) to train an intelligent agent through the Double Deep Q-Network (DDQN) reinforcement-learning algorithm. This agent interacts with nodes and adjusts its actions based on the results of its interactions.

III. PROPOSED CACHING STRATEGY

In our strategy, the network is partitioned into multiple clusters, each containing a Store Station. Each Store Station has three routers: the main router and two replacement routers. To manage caching operations within the cluster, the Store Station utilizes a new data structure called Content Store Station (CSS), which includes the content name, size, age, and placement details.

Table 1 presents a list of acronyms used in the caching strategy. When it comes to controlling the cache and eviction procedures, CSS is essential. A Long Short-Term Memory (LSTM) model is used to make caching decisions, deciding which router in the store station will cache particular content. Additionally, an eviction algorithm is also employed to remove content and free up space for new data.

Both CSS and the LSTM model are hosted on the main router. For our approach, we utilize the New Efficient Caching Strategy Based on Clustering in NDN (NECS) [12], [20], which employs the improved K-medoids cluster algorithm [21] to divide the network into clusters. The number of clusters, K, is determined using the Silhouette coefficient. Furthermore, the TOPSIS method [22] is applied to select three routers: the main router, replacement1, and replacement2, as illustrated in **Fig. 2**.

Table 1. *CSS-LSTM Acronyms*

Acronym	***Meaning***
CS	Content Store of NDN router
CSS	Content Store Station
Req(c_i)	Number of requests of content c_i
Weight(c_i)	Weight of c_i based on request frequency
Tot.Req	Total number of requests of cached contents on the station
Size (c_i)	Size of content ci in GB
Tot.Sizes	Total size of cached contents on the station in GB
Size.Pen (c_i)	Size Penalty of content c_i

1. LSTM Model Training

In this section, we will discuss the caching decision process within the Store Station and how an LSTM model will be used to achieve this objective. The model will be trained using a dataset from

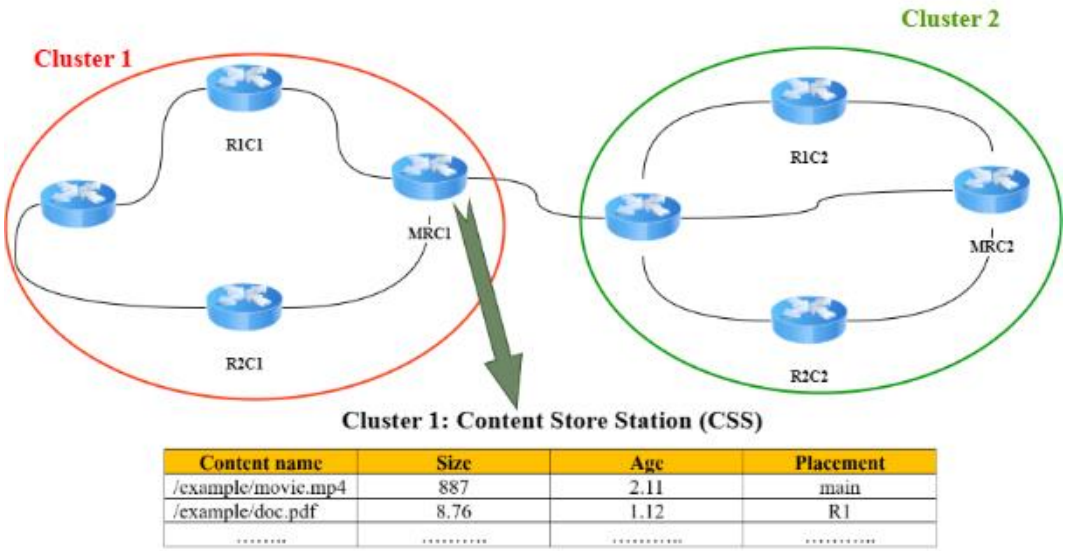

Figure 2. *NDN Network clusters*

Kaggle [23], which includes features obtained after the preprocessing, such as:

- Content name: a unique name for the requested content
- Content size: the size of the content in Bytes,
- Delay: time delay between requesting an item and receiving it.
- Time slot (*t*): requests arrive sequentially in discrete time slots, where ∀tϵ[1,T], the slot number is calculated based on the timestamp, in this case, the slot size is 15 minute, which means the range of slots is [1, 96].
- Number of requests: the total number of times the content has been requested.
- Content placement: the caching status of the content. If the placement is Miss, it means the content is not cached; otherwise, it is classified as 'Hit.

In the preprocessing step, requests for the same content are aggregated to compute the total number of requests as a feature. For each aggregated content item, we determine the following:

- Mean Latency: The average latency for all requests related to the content.
- Last Request Time: The timestamp of the most recent request, converted into a time slot.
- Placement: Remains the same placement as in the original requests.

This transformation ensures that the dataset captures both temporal and performance-related insights, leading to more effective caching decisions.

Recurrent Neural Networks (RNNs): are made especially to handle sequential data by preserving information from earlier inputs in a concealed state. An input layer, a hidden layer, and an output layer make up their fundamental architecture. Recurrent connections, as opposed to feedforward neural networks, allow information to be stored and cycled within the network.[24]

Table 2. *Training Hyperparameters of LSTM model*

Hyperparameter	***Value***
Embedding Dimension	50
LSTM Units (Layer 1)	128
LSTM Units (Layer 2)	64
Dropout Rate	0.2
Batch size	128
Epochs	30
Optimizer	Adam
Loss Function	Sparse Categorical Crossentropy
Activation Function	Softmax

Hochreiter and Schmidhuber [25] introduced Long Short Term Memory (LSTM) networks to address the vanishing gradient problem commonly found in basic standard RNNs. The gating mechanisms of LSTMs, which regulate the information flow throughout the network, are their primary innovation. Furthermore, LSTMs improve their ability to handle complex sequential data by learning relationships between requests over time steps.[26]

To train the model for caching decisions, we utilize the previously mentioned dataset, which contains over 5,000,000 examples. This dataset is divided into 80% for training and 20% for testing. **Table 2** presents the architecture of the LSTM model used.

2. Cache Management

In the proposed strategy, all traffic within the cluster is routed through the main router, which processes both Interest Packets (content requests) and Data Packets (content delivery). The main router is responsible for making caching decisions—determining whether to cache the content and, if so, selecting the optimal placement using the LSTM model. If caching is not required, the router simply forwards the Data Packet to its intended destination. **Fig. 3** illustrates the caching management framework.

This process is outlined in Algorithm 1, which explains the steps taken to manage the caching effectively. When the model decides to cache the content, it first calculates the content's age using Equation 3. If the selected router has sufficient space, the content is cached immediately. However, if space is limited, the cache manager removes expired content or content with an age smaller than that of the new content to make room. Once the content is cached, its information is added to the Content Store Station (CSS). If any content is deleted, the cache manager also removes the corresponding information from the CSS.

Each time new content is added to the CSS, the age of all stored contents is updated according to Algorithm 2. This ensures an efficient and up-to-date cache.

If sufficient space cannot be allocated for the new content, it is forwarded to its destination without being cached.

$$\text{Weight}(c_i) = \begin{cases} \dfrac{Req_{c_i}}{Tot.Req}, & if\ c_i\ cached \\ \dfrac{Req_{c_i}}{Req_{c_i} + Tot.Req}, & else \end{cases} \quad (1)$$

$$Size.Pen(c_i) = \frac{Size\ (c_i)}{Tot.Sizes} \times \propto \quad (2)$$

$$Age(c_i) = Weight(c_i) - Size.Pen(c_i) \quad (3)$$

Where Age(c_i) < 0, Then Age(c_i) = 0.

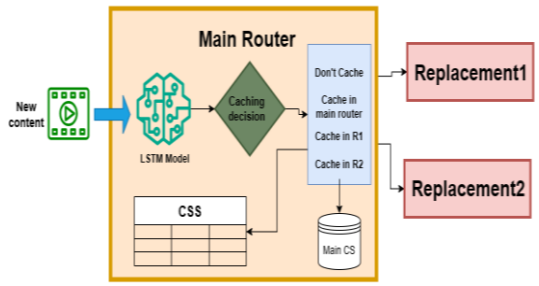

***Figure 3**. Caching management schema*

The penalty coefficient (α) is represented as follows:

- If the storage used is less than 25% of the total station size, α equals 0.
- If the storage used is between 25% and 50% of the total station size, α is 0.75.
- If the storage used exceeds 50% of the total station size, α is equal to 1.

Algorithm 1 LSTM-based Caching Algorithm

Require: Input content c
Ensure: Efficient caching decision for c

```
decision ← LSTMModel(c)
if decision = "Don't Cache" then
    Forward(c)
else
    age ← calculate age of c using Equation (3)
    if HasSpace(ChosenRouter) then
        Cache(c, ChosenRouter)
        Add c information to CSS
    else
        Initialize sizes_to_delete ← 0, to_delete_contents ← ∅
        Initialize i ← 1
        free_space ← FreeSpace(ChosenRouter)
        while i ≤ |CSS| and sizes_to_delete + free_space < size(c) do
            Let item ← CSS[i]
            if item.placement = ChosenRouter and item.age < age then
                sizes_to_delete ← sizes_to_delete + item.size
                to_delete_contents ← to_delete_contents ∪ {item}
            end if
            i ← i + 1
        end while
        if free_space + sizes_to_delete ≥ size(c) then
            Delete(to_delete_contents)
            Cache(c, ChosenRouter)
            Add c information to CSS
        else
            Forward(c)
        end if
    end if
    UpdateCSS()
end if
```

Algorithm 2 UpdateCSS Algorithm

Ensure: Updated Content Store Station (CSS)

```
for each item ∈ CSS do
    item.age ← ComputeAge(item)        ▷ Using Equation (3)
end for
```

IV. Results And Discussion

After training the LSTM model using the previously discussed dataset and parameters, we developed a model capable of making caching decisions within the NDN cluster. During the testing phase, which utilized 20% of the dataset, we achieved an accuracy of approximately 90%. To simulate the proposed caching strategy, we utilized OpenAI Gym [27], a toolkit for AI development. The cache sizes for the store station were configured as follows: main router cache size = 200 GB, replacement cache 1 size = 180 GB, and replacement cache 2 size = 150 GB. The LSTM model is deployed in the main router to make caching decisions, while the Content Store Station (CSS) data structure is also located in the main router to manage the caching and eviction process.

The proposed strategy is evaluated against other caching strategies, namely LCE, Random and NECS. To compare their performance, we used the cache hit ratio [28] as a metric, which effectively

measures caching efficiency. Equation 4 provides the formula for calculating the cache hit ratio.

$$Cache\ Hit\ Ratio = \frac{\sum_{n=1}^{N} hit_i}{\sum_{n=1}^{N}(hit_i + miss_i)}$$

Fig. 4 and **5** illustrate the results of four caching strategies: CSS-LSTM, NECS, Random, and LCE. These figures compare the performance of each strategy across two different scenarios. While a big number of desired contents are taken into consideration in the second scenario, only a small number are involved in the first.

The results indicate that both CSS-LSTM and NECS achieve higher cache hit ratios compared to Randon and LCE, particularly as the number of requested contents increases. However, Random and LCE initially perform well when the number of requested contents is low.

As content requests increase, NECS displays a notable improvement in the cache hit ratio. The network clustering and caching management, which is managed by the cluster head using the Least Recently Used (LRU) eviction strategy, are responsible for this improvement.

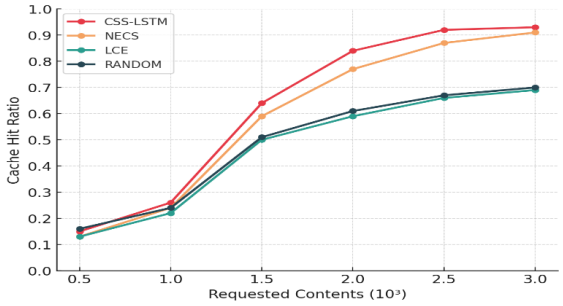

Figure 4. *Cache Hit Ratio vs Requested 10^3 contents*

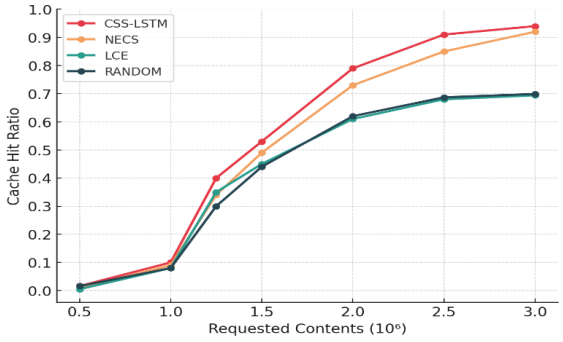

Figure 5. *Cache Hit Ratio vs Requested 10^6 contents*

In the presented graphs, CSS-LSTM demonstrates the highest cache hit ratio, regardless of the number of requested contents. Similar to NECS, CSS-LSTM benefits from network clustering; however, its key advantage lies in intelligent caching management. It makes strategic decisions about which content to cache and where to store it. Moreover, it employs a storage mechanism controlled by the CSS data structure, ensuring that popular content is retained longer than less popular content.

V. Conclusion

We offer an intelligent caching technique for an NDN cluster in this paper. This tactic optimizes caching choices by utilizing an LSTM model built into the cluster's primary router. We also present a CSS data structure for age-based content eviction management.

Experimental results demonstrate that the LSTM model achieves a high accuracy of approximately 90%, allowing the caching station to handle the caching process both efficiently and intelligently.

A comparative analysis of the CSS-LSTM strategy against three other strategies, NECS, Random, and LCE, across two different scenarios reveals that CSS-LSTM exhibits higher caching efficiency than Random and LCE, the default caching strategy. Moreover, CSS-LSTM outperforms NECS in two important aspects: first, in making intelligent caching decisions, and second, in effectively managing cached content using CSS.

The CSS-LSTM is designed to adapt to high demand for content in Named Data Networking NDN, particularly for multimedia content, which requires more advanced and intelligent management for efficient caching.

AUTHOR CONTRIBUTIONS

I.K. Douis: Conceptualization, Experiments, Theoretical analysis, Writing.

H. Bouziane: Writing, Review, and editing.

A. Chouarfia: Supervision, Review, and editing.

Disclosure Statement

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ORCID

I.K. Douis https://orcid.org/0009-0009-7417-025X

A. Chouarfia https://orcid.org/0000-0003-3331-720X

REFERENCES

[1] T. Bianchi, "Global top websites by monthly visits 2024," Statista. Accessed: Feb. 12, 2025. [Online]. https://www.statista.com/statistics/1201880/most-visited-websites-worldwide/

[2] M. I. Victor- Ikoh and L. G. Kabari, "Internet Architecture: Current Limitations Leading Towards Future Internet Architecture," IJCSMC 10 (5) (2021) pp. 102–112. https://doi.org/10.47760/ijcsmc.2021.v10i05.011

[3] L. Zhang et al., "Named Data Networking (NDN) Project." Oct. 31, 2010. Accessed: Feb. 15, 2025. [Online]. Available: https://named-data.net/wp-content/uploads/TR001ndn-proj.pdf

[4] A. Tariq, R. A. Rehman, and B.-S. Kim, "Forwarding Strategies in NDN-Based Wireless Networks: A Survey," IEEE Commun. Surv. Tutorials 22 (1) (2020) pp. 68–95. https://doi.org/10.1109/COMST.2019.2935795

[5] A. Majed, X. Wang, and B. Yi, "Name Lookup in Named Data Networking: A Review," Information 10 (3) (2019) p. 85. https://doi.org/10.3390/info10030085

[6] D. Doan Van and Q. Ai, "In-network caching in information-centric networks for different applications: A survey," Cogent Engineering 10 (1) (2023) p. 2210000. https://doi.org/10.1080/23311916.2023.2210000

[7] L. Zhang et al., "Named Data Networking," ACM SIGCOMM Computer Communication Review 44 (3) (2014). https://doi.org/10.1145/2656877.2656887

[8] D. Saxena, V. Raychoudhury, N. Suri, C. Becker, and J. Cao, "Named Data Networking: A survey," Computer Science Review 19 (2016) pp. 15–55. https://doi.org/10.1016/j.cosrev.2016.01.001

[9] Y. Li, M. Yu, and R. Li, "A Cache Replacement Strategy Based on Hierarchical Popularity in NDN," in 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN), Prague, Czech Republic: IEEE, Jul. 2018, pp. 159–161. https://doi.org/10.1109/ICUFN.2018.8436597

[10] M. Hosseinzadeh, N. Moghim, S. Taheri, and N. Gholami, "A new cache replacement policy in named data network based on FIB table information," Telecommun Syst 86 (3) (2024) pp. 585–596. https://doi.org/10.1007/s11235-024-01140-7

[11] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and Sangheon Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," in 2012 Proceedings IEEE INFOCOM Workshops, Orlando, FL, USA: IEEE, Mar. 2012, pp. 316–321. https://doi.org/10.1109/INFCOMW.2012.6193512

[12] N. E. H. Fethellah, H. Bouziane, and A. Chouarfia, "NECS-based Cache Management in the Information Centric Networking," Int. J. Interact. Mob. Technol. 15 (21) (2021) p. 172. https://doi.org/10.3991/ijim.v15i21.20011

[13] M. Alkhazaleh, S. A. Aljunid, and N. Sabri, "An Efficacious Content Caching and Eviction Priorities (CCEP) for In-network Caching High Performance in Information-centric Networking," IAENG International Journal of Applied Mathematics 53 (2023) pp. 169–182. [Online]. https://www.iaeng.org/IJAM/issues_v53/issue_1/IJAM_53_1_22.pdf

[14] N.-T. Dinh and Y. Kim, "An Efficient Distributed Content Store-Based Caching Policy for Information-Centric Networking," Sensors 22 (4) (2022) p. 1577. https://doi.org/10.3390/s22041577

[15] D. Gupta, S. Rani, S. H. Ahmed, S. Verma, M. F. Ijaz, and J. Shafi, "Edge Caching Based on Collaborative Filtering for Heterogeneous ICN-IoT Applications," Sensors 21 (16) (2021) p. 5491. https://doi.org/10.3390/s21165491

[16] Y. Liu, Z. Ma, Z. Yan, Z. Wang, X. Liu, and J. Ma, "Privacy-preserving federated k-means for proactive caching in next generation cellular networks," Information Sciences 521 (2020) pp. 14–31. https://doi.org/10.1016/j.ins.2020.02.042

[17] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "DeepMEC: Mobile Edge Caching Using Deep Learning," IEEE Access 6 (2018) pp. 78260–78275. https://doi.org/10.1109/ACCESS.2018.2884913

[18] J. Hou, H. Lu, and A. Nayak, "A GNN-based Proactive Caching Strategy in NDN Networks," Jun. 08, 2022. https://doi.org/10.21203/rs.3.rs-1713271/v1

[19] J. Hou, T. Tao, H. Lu, and A. Nayak, "Intelligent Caching with Graph Neural Network-Based Deep Reinforcement Learning on SDN-Based ICN," Future Internet 15 (8) (2023) p. 251. https://doi.org/10.3390/fi15080251

[20] N. E. H. Fethellah, H. Bouziane, and A. Chouarfia, "New Efficient Caching Strategy based on Clustering in Named Data Networking," Int. J. Interact. Mob. Technol. 13 (12) (2019) p. 104. https://doi.org/10.3991/ijim.v13i12.11403

[21] C. Li and K. Okamura, "Cluster-based In-networking Caching for Content-Centric Networking," International Journal of Computer Science and Network Security (IJCSNS) 14 (11) (2014). http://paper.ijcsns.org/07_book/201411/20141101.pdf

[22] M. Behzadian, S. Khanmohammadi Otaghsara, M. Yazdani, and J. Ignatius, "A state-of the-art survey of TOPSIS applications," Expert Systems with Applications 39 (17) (2012) pp. 13051–13069. https://doi.org/10.1016/j.eswa.2012.05.056

[23] K. L. Bảo, "Predicting the Popularity of Online Content." 2022. Accessed: Jan. 05, 2025. [Online]. https://www.kaggle.com/datasets/bokhnhl/no-package-live-streaming

[24] C. Zhang et al., "Toward Edge-Assisted Video Content Intelligent Caching With Long Short-Term Memory Learning," IEEE Access 7 (2019) pp. 152832–152846. https://doi.org/10.1109/ACCESS.2019.2947067

[25] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Computation 9 (8) (1997) pp. 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[26] Y. Wang and V. Friderikos, "A Survey of Deep Learning for Data Caching in Edge Network," Informatics 7 (4) (2020) p. 43. https://doi.org/10.3390/informatics7040043

[27] G. Brockman et al., "OpenAI Gym," Jun. 05, 2016, arXiv: arXiv:1606.01540, https://doi.org/10.48550/arXiv.1606.01540

[28] M. A. Naeem, S. A. Nor, S. Hassan, and B.-S. Kim, "Compound Popular Content Caching Strategy in Named Data Networking," Electronics 8 (7) (2019) p. 771. https://doi.org/10.3390/electronics8070771

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution NonCommercial (*CC BY-NC 4.0*) license.