Advanced International Journal for Research (AIJFR)
E-ISSN: 3048-7641 ● Website: www.aijfr.com ● Email: editor@aijfr.com

A Hybrid Cache Replacement Strategy for Named Data Networking Using Online, Approximate, and Interval based Policy Iteration with PIT-Based Eviction

Sushil Kumar Bagi [1], Prof. (Dr.) Neeraj Kumar[2]

[1]Research Scholar, Department of Engineering & Technology, Suresh Gyan Vihar University, Jaipur, Rajasthan, India
[2] Professor, Department of Engineering & Technology, Suresh Gyan Vihar University, Jaipur, Rajasthan, India

Abstract

Named Data Network is a new architecture for future network. It is based on the importance of data sharing concept without using location based IP-Address. Caching is the technique to improve the performance of NDN. However the cache replacement in this technology is the decision making process. When memory become full then cache replacement techniques are used to take the decision in order to find which data item should be evicted from memory so that new arrival of data can be inserted. The purpose of this paper to enhance the NDN performance by develop the new cache replacement technique based on an online approximate policy iteration framework which enabled lightweight decision making named Policy Iteration Cache Replacement (PI-CREPL) which is hybrid approach of policy iteration and pending interest table based cache replacement technique i.e. PIT-REPL. Where PIT-REPL is cache replacement strategy that is based on pending interest table. Unlike classical Policy Iteration, the proposed approach (PI-CREPL) performs online and interval-based policy updates without requiring convergence of the value function, enabling adaptive cache replacement under dynamic traffic conditions. This PI-CREPL is used to make the optimal decision for eviction based on two concepts, one is the dynamic programming of MDP and second one is popularity of data Pending Interest Table for eviction. Simulation results show that PI-CREPL outperforms with, Priority FIFO, LRU and PIT-REPL methods in terms of cache hit ratio and latency across both tree and linear topologies.

Keywords: NDN, Cache Replacement, Interest Packet, MDP, Data Packet, Policy Iteration

1. Introduction

Due to large number of data on internet it is needed to handle with a new way of network architecture which provides the data sharing in speedy manner that proposed architecture is NDN. The current network is based on the concept of TCP/IP that is lacking in terms of services like security, reliability and quality services[1]. The NDN principles are based on the concept of caching where store the data to the nearest of consumers so that for the same request in future, no need to establish the re-connection with producers. In the TCP/IP network every time for the same data we have to connect to the same

machine that is location oriented. In NDN, the intermediate routers maintain the three data structures (i) Pending Interest Table (PIT), to maintain the pending request (ii) Forwarding Information Base (FIB), is use to forward the request to next node or producer and (iii) Content Store (CS) is the memory which hold the data that are more in popular, to full fill the requested from this memory. Content Store is also known as cache memory. Since content store is limited memory that's why it is need to use such strategy that take the intelligent decision for eviction of data item from memory when the CS become full. There are so many traditional cache replacement strategies are available like FIFO,LFU,LRU and RR etc. out of which LRU performs high but the main lacking of this technique is that LRU performance degrade when the requested data is not the same with previously requested data[2].Also the main disadvantage of traditional techniques that they are static in nature and they are not tackle with dynamic network condition. So we require the cache replacement strategy which is not in static nature that the new strategy should be support on dynamic changes of data in network. In the research paper [3], Authors told ,Policy iteration is classical dynamic programming of the markov decision process (MDP). The Markov decision process is a similar kind of process that incorporates actions and rewards. It is a mathematical framework used for modelling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. It is a fundamental framework in Artificial Intelligence (AI) and Machine Learning (ML) used to model decision-making in environments [4]. MDP is represented by four tuples (S,A,R,P) where S is states, A is set of action, R is immediate reward after taking action A on state S.P is the probability of movement from one state to another state. Same we use the concept to represent the cache replacement strategy. The NDN network is heavily basis on network caching, where cache replacement is important activity to replace the data item from memory. Despite strong recommendation for policy iteration in replacement type MDP, there is current no research works where cache replacement in the area of named data networking that totally based on online approximation of policy iteration. Unlike classical policy iteration that is used repeated policy evaluation until convergence where in this work policy evaluation is performed using truncated Bellman updates. It will allow the caching policy to be updated incrementally when content request arrive and eviction is based on PIT table. This design significantly reduces computational overhead and makes the proposed strategy for real time deployment in NDN routers. Their result shows that the Policy iteration-inspired algorithm support to solve the problem which are polynomial in nature that is used to solve the problem of model replacement and maintenance problem.This research gap clear indicates between replacement of policy to obtain the optimal policy in MDP and practical cache replacement within the environment of named data networking under the ndnSIM simulator. Rest of this paper organized as follows introduction in section1.Discription of system overview in section 2.Related works in section 3.Proposed strategy presented in section 4.Implementation of proposed work in section 5.Experimental setup in section 6.Result and discussion in section 7.Conclusion of study in section 8 and at last references and bibliography mention in section 9.

2. System Overview:

NDN architecture is the advocacy of the NSF Future Internet Architecture (FIA) program. **NDN** communication established based on three nodes, three messages and Three data structures .Where are Three nodes are (1)Consumer; consumer is the node which request the data item from network (2) Producer; Producer is the node which produce requested data and provide to the other nodes and (3) router; router is the intermediate node which stores the data and handles the same request of consumer , Three messages are (i) Interest message; It is the set of string use as request message (ii)Data packet ; Data packet is actual data which are generated by producer (iii)NACK; It is the acknowledgement to

requested node when the requested data not found and the three data structures which are maintained by Routers are (a)CS; It is known as content store, It is basically temporary memory used as cache memory (b) PIT; It is known as Pending interest table. It maintain the list of pending interest which are still not satisfied and (c) FIB; It is known as forwarding information base table. It is used to forward the interest message based on the forwarding strategy. The same depicted in figure 1, below where it shown that PIT, FIB and CS are present on router itself. Figure 2, shows detail communication of NDN[5].

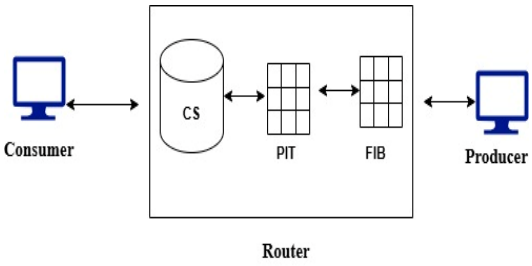

Fig. 1: System over view of NDN

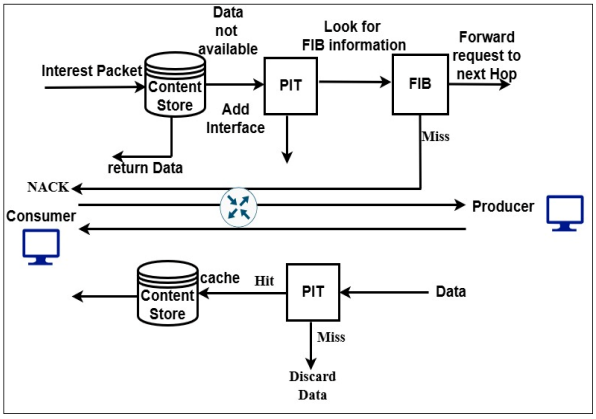

Fig. 2: Communication Procedure of NDN

3. Related Works:

In NDN network the cache replacement strategy are decision making process. There are several research has been conducted regarding cache replacement technique. In the paper [6], authors use the MDP mathematical framework for to develop the cache replacement technique. In this authors not used any specific tool for to conduct the experiment but they use the python programming to compare the MDP based cache replacement with traditional cache replacements i.e. LFU and FIFO.

In the paper [7],authors did the review of various papers and tells about the importance of cache placement and replacement technique there are many strategies discussed on cache replacement for example adaptive neuro-fuzzy inference system, Deep Learning based framework, NDN-based Universal caching etc. all authors used to developed the modern caching with machine learning but no one use the Pending Interest Table information in cache replacement. In the paper [8], authors developed the cache replacement technique that is based on content popularity by maintaining the separate data structure which maintain the popularity of content. The main disadvantages is this that it may lead to resource underutilization when popularity of content change rapidity, further more author used the separate data structure which lead to computational cost and did experiment on Icarus. In the paper [9],authors developed the cache replacement based on energy consumption and utilization, still the this strategy is not used the information directly from PIT table. In this paper [10],Authors used the Apriory algorithm and tells that this method is gain high CHR as compared to LRU. In the paper [11],Authors developed the cache replacement strategy based on FIB table where Uses FIB table information to calculate content retrieval time based on two parameters: Grade of Retrieval (GOR) and Stale Parameter (SP). Content with shorter retrieval time is assigned a higher discard priority. So based on research paper no authors used the combine approach of Policy Iteration dynamic programming and PIT table based cache replacement in NDN that take the pending interest table for eviction of data item.

4. Proposed Work: In this section, both the Pending table-based cache replacement and the online approximate Policy iteration-based cache replacement algorithms are explained. **4.1 PIT-REPL:** The pending interest table has several parameters; we have taken only a two of them to calculate the popularity of requested data. The interest packets and the total different faces that have requested the same interest to count the total popularity. The popularity score for interest i is represented as equation one.

$$\text{Score}(i) = R_i \times F_i \quad (1)$$

Where R_i=Total number of requests received for the interest i. F_i=Total number of distinct faces that requested interest i. For each interest i, the popularity score will be calculated as per table number one.

Table 1: Popularity table

Interest Name	Total Requests	Unique Faces	PIT-Score=Requests * Faces
/root/seq=23	10	2	20
/root/seq=26	5	8	40
/root/seq=27	12	4	48

Eviction Decision: Evict the data item Di from memory whose corresponding interest i has lowest score, same is denoted with equation two. The lowest score means that the data item is less in demand.

$$D_{evict}=D_{i^*}, \quad i^*=\arg\min_{i \in C} \text{Score}(i) \quad (2)$$

If multiple interests has same minimum score then choose the one for eviction based on LRU.
As per table no. 1 the lowest PIT-Score is 20 that mean the request /root/seq=23 have less in demand so evict the corresponding data item from memory.

4.2 Policy Iteration:
In online approximate inspired based policy iteration, policy updates are performed by online while eviction decision is taken by minimum PIT score.
Policy iteration is another dynamic programming algorithm used to compute the optimal policy. It alternate between two steps.
(a) Policy Evaluation: For a given policy $\prod$, the value function $V^*(S')$ is the computed using the Bellman Expectation equation as mention in equation three.

$$V^{\prod}(s) = E[R(s,\prod(s))+\gamma V^{\prod}(T(s,\prod(s)))] \quad (3)$$

where s is the current state. $\prod(s)$ is the action selected by fixed policy $\prod$ in state s.
$R(s,\prod(s)$ is the immediate reward obtain after taking action $\prod(s)$ in state s.
$V^{\prod}(T(s,\prod(s)))]$ is value of the next state when following the same policy $\prod$.
γ it is discount factor which value lies between 0 and 1 and tells the importance of the future reward.

(b) Policy Improvement: Once the value function for the current policy is calculated, the policy is updated to improve it by selecting the action that maximizes the expected return from each state. It is denoted as per equation four.

$$PI^{\pi}(s)=\text{argmax}_{a \in A} Q^{\pi}(s,a) \quad (4)$$

where a belongs to A, mean select the action a from the action space and $Q^{\pi}(s,a)$ is the action value function under the current policy ∏.
This process repeats until the policy converges meaning it no longer changes between iterations. Now in our case every call has one iteration since there are only two actions that's why we chosen one iteration to identify the optimal policy but if there are more actions and large state space then we can modify accordingly.
Example: Let's try to understand with very simple example with single iteration as below
Let Discount Factor=0.9
Actions: (i) KEEP (ii) EVICT
KEEP means retain cache entry
EVICT means remove data item from memory
in policy iteration each action has own transition behaviour.
States represent the cache status of content item:
S_0:content is popular
S_1:contnent is less popular
tables 2 and 3 represents the transition of states.
Table 2

Action :KEEP	
Current State	**Next State**
S_0	S_1
S_1	S_1

Table 3

Action :EVICT	
Current State	**Next State**
S_0	S_0
S_1	S_0

REWARD for both actions are fixed like action keep has 5 and Evict action has 1

Step 1: Policy Initialization (π_1) i.e. always KEEP

π_1=KEEP for all S
transitions $S_0 \rightarrow S_1$ and $S1 \rightarrow S1$

Step 2: Policy Evaluation for the policy π_1 : the policy evaluation is done with value function evaluation:
The value function is updated iteratively using the Bellman expectation equation. Here, we showed only one iteration where we found the value of the policy by using equation number three. First compute the value function for policy one and then compute the value function for policy two.

$V\ \pi_1(s_1)=5+0.9\times V\pi_1 S_1=50$
$\Rightarrow V\pi_1 S_1=50$
So now we can compute $v\pi_1(s_0)=5+0.9\times 50=50$
After computation the value function for π1 it shown in table 4.
Table 4: value function for policy π1

State	Value
S0	50
S1	50

Step3 :Policy Evaluation for the policy π2
Now we can check with other action i.e. EVICT
π_2=EVICT for all S
For transitions
$S_0 \rightarrow S_0$
$S_1 \rightarrow S_0$
Policy Evaluation (π_2)
so first compute the value for $v\pi_2(s_0)$
$v\pi_2(s_0)=1+0.9\times v\pi_2 s_0 \Rightarrow v\pi_2 s_0 = 10$ then calculate the $v\pi_2(s_1)$ as per transition
$v\pi_2(s_1)=1+.9x10=10$
Now the value function for π_2 is shown in table 5.
Table 5: value function for policy π_2

State	Value
S_0	10
S_1	10

Step 4: Policy comparison: Find the optimal policy by using equation number four with the help of table 4 and 5. It is represented in table 6.

Table 6: comparison of value function for both policy π_1 and π_2

State	V π_1	V π_2
S0	50	10
S1	50	10

$V\pi_1(S) > V\pi_2(S)\ \forall S$
Step 5: Policy Improvement Decision: The optimal Policy=KEEP as per above comparison, like this online approximate policy iteration works. It execute after some n number of eviction as interval basis to reduce computational overhead.

5. Implementation of Proposed PIT-REPL and PI-CREPL algorithms:

The design of online approximation policy iteration is implemented by using interval-based policy evaluation and policy improvement periodically after observing a batch of cached data items in the content store, rather than after every single event. This way, we can reduce the computational overhead, making this strategy lightweight.

To implement both algorithms, we required to obtain the information of all unique faces and the total number of requests from the pit-in-record.cpp file with the use of some data structures like map, vector and set. That information will be accessed by both custom policies like pit.cpp i.e. pending interest table based cache replacement and pi.cpp i.e. Policy iteration based cache replacement. In ndnSIM we have developed the three separate methods apart from the five policy methods, so all methods look like (i) Policy evaluation, (ii) Policy improvement, (iii) Reward, (iv) evictEntries, (v) doAfterInsert, (vi)doBeforeUse, (vii)doAfterRefresh (viii) doBeforeErase [12]

The main function in the pending interest table-based cache replacement is evictentries, and the important function policy iteration is policy evaluation, policy improvement, and evict entries.
The pseudo code for the above three important functions is written below:

(i)Policy evaluation: In policy evaluation is the process where we can evaluate the each state here state is action and eviction.

1.
   Input:
   a. Policy π(s)
   b. Discount factor γ
   c. State space S
   d. Value function V(s)
2. Output:
   a. Updated value function V(s)
3. for each state s ∈ S do
   a. a ← π(s) // action selected by current policy
   b. if a = KEEP then
   c. s' ← NextStateKeep(s)
   d. else
   e. s' ← NextStateEvict(s)
   f. end if
   g. V(s) ← Reward(s, a, s') + γ · V(s')
4. end for
5.
   (ii) Policy Improvement:

   Input:
   a. Value function V(s)
   b. Discount factor γ
   c. State space S
   d. Current policy π(s)

6. Output:
   a. Updated policy π(s)
   b. Policy stability flag
7. policyStable ← true

8. for each state s ∈ S do
    a. s_keep ← NextStateKeep(s)
    b. s_evict ← NextStateEvict(s)
    c. Q_keep ← Reward(s, KEEP, s_keep) + γ · V(s_keep)
    d. Q_evict ← Reward(s, EVICT, s_evict) + γ · V(s_evict)
    e. if Q_keep ≥ Q_evict then
    f. bestAction ← KEEP
    g. else
    h. bestAction ← EVICT
    i. end if
    j. if π(s) ≠ bestAction then
    k. π(s) ← bestAction
    l. policyStable ← false
    m. end if
    n. V(s) ← max(Q_keep, Q_evict)
9. end for
10. return policyStable

(iii)Evict Entries

victimName ← argmin_name Stats[name].score // find the content with minimum score
victimEntry ← entry ∈ Q where entry.name = victimName // locate the victim in cache queue
emit beforeEvict(victimEntry) // notify the cs about eviction
remove victimEntry from Q //remove from local queue

6. Experimental Setup:

The tree and linear topologies with the following simulation parameters are used to simulate the strategies.

Tree topology shown in fig. 3, and linear topology shown in fig. 4. Table 7 represents the simulation parameters for the practical implementation of the above strategies. In tree topology, a single namespace is used to emphasize interest aggregation, whereas in linear topology, multiple namespaces are used to create more diversity and heterogeneous traffic, making cache eviction decisions harder.

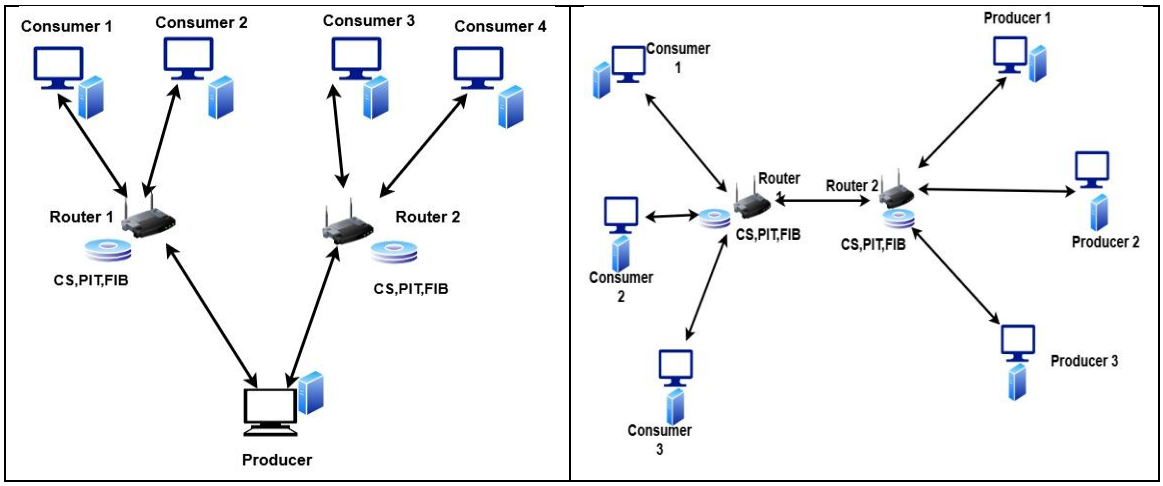

Fig.3: Tree Toplogy	Fig. 4: Linear Topology

Table 7 : Simulator Parameters:

Parameter	**Value**
Request Rate	100 request per sec.
Number of contents	500
q-Plateau Parameter	0.8
s- Skew Parameter	1.2
CS size	10/20/30/40/50/60/70/80 /90/100 (entries)
Cache Strategy	Priority FIFO,LRU,PIT-REPL, PI-CREPL
Forwarding Strategy	best-route
Simulation Time	100 s
Topology	Tree Topology & Linear Topology
Number of topology nodes	7 (Tree Topology)
	8 (Linear Topology

7. Result and Discussion:

The following tables 8 and 9 show the results generated by the ndnSIM simulator to evaluate the performance of cache replacement algorithms, i.e., Priority FIFO, LRU, PIT-REPL, and PI-CREPL, in terms of CHR (%) and Latency(micro seconds).

Table 8: The average CHR and the average Latency of cache replacement algorithms under Tree Topology

Tree Topology									
SNo.	CS-Size	Priority FIFO	LRU (Average CHR)	PIT-REPL (Average CHR)	PI-CREPL (Average CHR)	Average Latency Priority FIFO	Average Latency LRU	Average Latency PIT-REPL	Average Latency PI-CREPL
1	10	26.04	30.56	21.43	36.20	5233.951 us	5086.867 us	5405.212 us	4821.341 us
2	20	39.09	45.50	40.04	54.26	4795.263 us	4582.589 us	4767.896 us	229.942 us
3	30	46.82	53.85	49.69	62.21	4530.04 us	4302.721 us	4437.762 us	3978.411 us
4	40	52.58	59.15	55.92	67.14	4340.828 us	4121.497 us	4227.444 us	3824.64 us
5	50	56.86	63.29	60.51	70.76	4201.584 us	3986.994 us	4078.099 us	3711.273 us
6	60	60.21	66.57	63.85	73.28	4087.214 us	3876.588 us	3961.456 us	3632.952 us

7	70	63.43	69.41	67.16	75.44	3989.462 us	3790.565 us	3863.031 us	3564.096 us
8	80	65.85	71.63	69.65	77.24	3908.019 us	3720.07 us	3779.437 us	3507.898 us
9	90	67.87	73.36	71.58	78.65	3840.642 us	3660.85 us	3716.279 us	3462.147 us
10	100	69.66	74.92	73.19	79.97	3780.271 us	3608.779 us	3663.854 us	3420.55 us

Table 9: The average CHR and the average Latency of cache replacement algorithms under Linear Topology

Linear Topology									
SNo.	CS-Size	Priority FIFO	LRU (Average CHR)	PIT-REPL (Average CHR)	PI-CREPL (Average CHR)	Average Latency Priority FIFO	Average Latency LRU	Average Latency PIT-REPL	Average Latency PI-CREPL
1	10	2.73	2.97	0	2.06	58361.451 us	58232.811 us	60440.784 us	58897.606 us
2	20	7.35	9.12	5.52	14.87	55117.748 us	54117.493 us	56486.965 us	49876.094 us
3	30	11.24	14.61	11.58	25.37	52609.957 us	50779.448 us	52598.824 us	44393.749 us
4	40	14.55	19.52	16.02	28.47	50595.16 us	48110.516 us	49787.065 us	42691.31 us
5	50	17.37	23.36	19.97	31.85	48899.33 us	46036.236 us	47564.782 us	40989.056 us
6	60	19.80	26.77	22.92	35.13	47487.941 us	44298.498 us	45863.762 us	39503.128 us
7	70	22.32	29.79	26.34	37.76	46168.477 us	42870.352 us	44249.011 us	38332.328 us
8	80	24.56	32.33	28.74	39.93	44994.213 us	41673.687 us	43044.743 us	37383.76 us
9	90	26.57	34.55	30.89	41.86	44046.395 us	40608.325 us	41995.473 us	36543.878 us
10	100	28.30	36.58	33.32	43.72	43149.845 us	39705.171 us	40959.823 us	35765.408 us

(a)Cache Hit Ratio and Latency analysis: As shown in the table, across both topologies, PI-CREPL

consistently outperforms Priority FIFO, LRU, and PIT-REPL in terms of cache hit ratio and latency. In tree topology, PIT-REPL cache hit ratio lies above priority FIFO and lower of LRU from CS 20 to 100. At CS-10, the PIT-REPL is below Priority FIFO because Priority FIFO is static; it does not require any dynamic state information, whereas PIT-REPL requires PIT based dynamic counting, minimum data content and request history to make a decision. Under a very small content store the all this information is not appropriately available, which leading low cache hit rate. The same thing happened in linear topology; for example, at cs-10 and 20, the PIT-REPL cache hit rate is lower than Priority FIFO. After CS-30 to 100, the PIT-REPL hit rate is higher than Priority FIFO. In both Tree and Linear Topology, the PI-CREPL achieve high cache hit ratio in comparison to all strategies like LRU, Priority FIFO, and PIT-REPL. Given the linear topology design, which offers diversity and low temporal locality, we can say that the proposed method PI-REPL remains effective under unfavourable conditions. Same depicted in a linear graph, like figs 5 and 7, showing the average cache hit ratio for tree and linear topologies, respectively, with respect to the content size for all the above-discussed caching strategies where PI-CREPL has higher cache hit ratio in comparison to Priority FIFO, LRU, PIT-REPL, and PI-CREPL. Where Fig 6 and 8 shown average latency in tree topology and linear topology, respectively, and these figures show PI-CREPL has the lowest latency among all caching strategies.

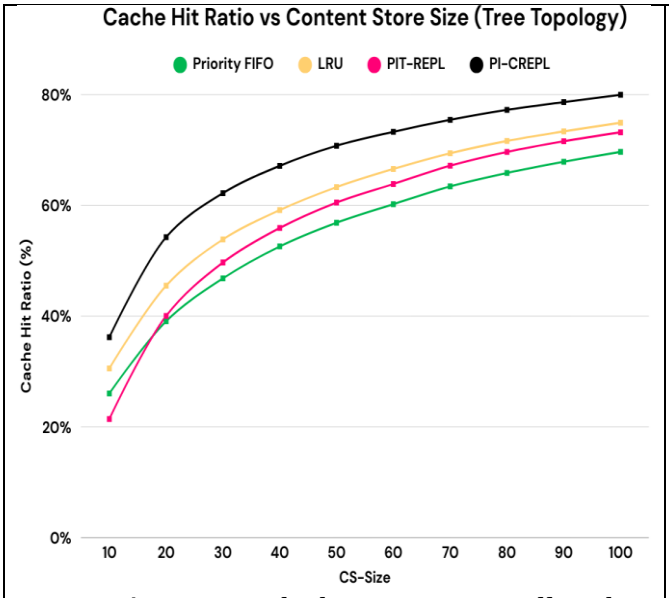

Fig. 5: Average cache hit ratio among all cache strategies i.e. Priority FIFO,LRU,PIT-REPL and PI-REPL in Tree Topology

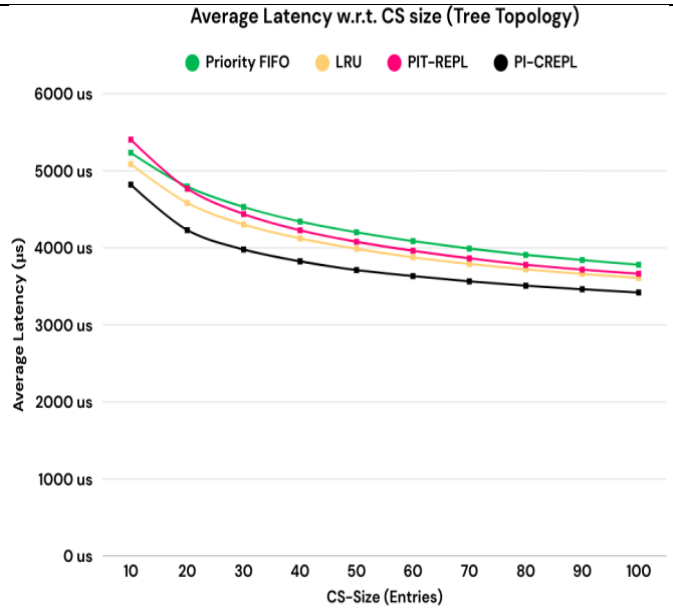

Fig. 6: Average Latency among all cache strategies i.e. Priority FIFO,LRU,PIT-REPL and PI-REPL in Tree Topology

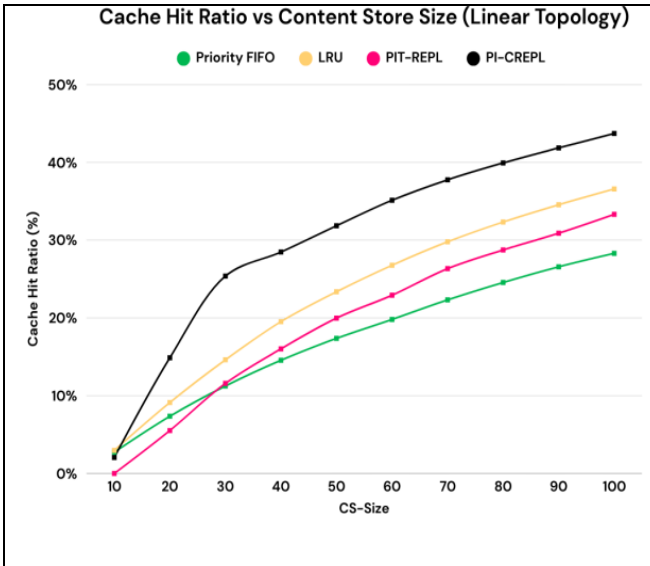

Fig. 7: Average cache hit ratio among all cache strategies i.e. Priority FIFO,LRU,PIT-REPL and PI-REPL in Linear Topology

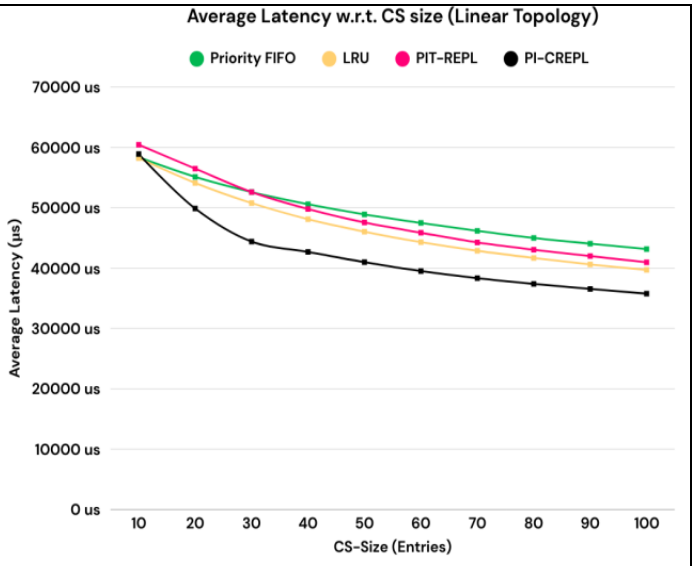

Fig. 8: Average Latency among all cache strategies i.e. Priority FIFO,LRU,PIT-REPL and PI-REPL in Linear Topology

8. Conclusion:

This manuscript presents a comparative evaluation of traditional cache replacement strategies (Priority FIFO and LRU) and adaptive cache replacement strategies (PIT-REPL and PI-CREPL) in Named Data Networking under tree and linear topologies, varying the content store sizes. The Results show that adaptive cache replacement strategies significantly outperform traditional approaches; with PI-CREPL achieving the highest cache hit ratio and lowest latency amongst the others in both topologies. Where the cache hit ratio of PIT-REPL is higher than priority FIFO despite experiencing a cold-start effect at very small content store size due to sparse PIT information. PI-CREPL depends on a model-based Markov Decision Process which requires predefined state transition and reward to take the decision which may not actual inclusion of highly dynamic network traffic. The results show that adaptive policy-iteration-based cache replacement improved Named Data Networking performance. In future work, this study will be extended by integrating a model-free reinforcement learning technique to develop an adaptive cache replacement strategy using both PIT-FIB tables.

Refrences and Bibliography:

1. E. T. D. Silva, J. M. H. D. Macedo, and A. L. D. Costa, “NDN Content Store and Caching Policies: Performance Evaluation,” Computers, vol. 11, no. 3, p. 37, Mar. 2022, doi: 10.3390/computers11030037.
2. L. Saino, I. Psaras, and G. Pavlou, “Icarus: a Caching Simulator for Information Centric Networking (ICN),” in Proceedings of the Seventh International Conference on Simulation Tools and Techniques, Lisbon, Portugal: ICST, 2014. doi: 10.4108/icst.simutools.2014.254630.
3. A. Fern, S. Yoon, and R. Givan, “Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes,” jair, vol. 25, pp. 75–118, Jan. 2006, doi: 10.1613/jair.1700.

4. F. Bizzarri, C. Mocenni, and S. Tiezzi, “A Markov Decision Process with Awareness and Present Bias in Decision-Making,” Mathematics, vol. 11, no. 11, p. 2588, June 2023, doi: 10.3390/math11112588.
5. S. Mastorakis, A. Afanasyev, and L. Zhang, “On the Evolution of ndnSIM: an Open-Source Simulator for NDN Experimentation,” SIGCOMM Comput. Commun. Rev., vol. 47, no. 3, pp. 19–33, Sept. 2017, doi: 10.1145/3138808.3138812.
6. S. K. Bagi and N. Kumar, “Markov decision process-based cache replacement for efficient content retrieval in named data networking,” in Artificial Intelligence and Sustainable Innovation, 1st ed., London: CRC Press, 2025, pp. 179–184. doi: 10.1201/9781003654049-27.
7. R. Alubady, M. Salman, and A. S. Mohamed, “A review of modern caching strategies in named data network: overview, classification, and research directions,” Telecommun Syst, vol. 84, no. 4, pp. 581–626, Dec. 2023, doi: 10.1007/s11235-023-01015-3.
8. T. Fukushima, M. Iio, K. Hirata, and M. Yaoamoto, “Popularity-Based Content Cache Management for in-Network Caching,” in 2019 International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia: IEEE, Jan. 2019, pp. 411–413. doi: 10.1109/ICOIN.2019.8718180.
9. “46. An In-Network Caching Scheme Based on Energy Efficiency for Content-Centric Network(2018-IEEE)[YING AN1AND XILUO2”.
10. C. A. Kerrache, G. Rathee, Y. Guellouma, H. Gasmi, and B. Ziani, “Towards an Efficient and Secure Cache Management Using Apriori-Based Interests Prediction in Named Data Networking,” in 2023 11th International Conference on Intelligent Systems and Embedded Design (ISED), Dehradun, India: IEEE, Dec. 2023, pp. 1–6. doi: 10.1109/ISED59382.2023.10444543.
11. 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, vol. 86, no. 3, pp. 585–596, July 2024, doi: 10.1007/s11235-024-01140-7.
12. Alexander Afanasyev and Junxiao Shi, “NFD Developer’s Guide,” Named Data Networking Project, Aug. 2021. [Online]. Available: https://named-data.net/publications/techreports/