Global Journal of Engineering and Technology Advances
eISSN: 2582-5003
Cross Ref DOI: 10.30574/gjeta
Journal homepage: https://gjeta.com/

(RESEARCH ARTICLE)

Check for updates

ENHANCEMENT OF PIT-AWARE CONTENT CACHE REPLACEMENT WITH RTT-BASED SELECTIVE CACHING & Q-LEARNING FOR NAMED DATA NETWORKING

Sushil Kumar Bagi [1,*] and Neeraj Kumar [2]

[1] Department of Computer Science & Engineering, Suresh Gyan Vihar University, Jaipur, Rajasthan, India.
[2] Department of Engineering & Technology, Suresh Gyan Vihar University, Jaipur, Rajasthan, India.
[*] Corresponding Author

ORCID Details
Sushil Kumar Bagi: https://orcid.org/0009-0009-1735-3404
Neeraj Kumar: https://orcid.org/0000-0002-4318-7704

Global Journal of Engineering and Technology Advances, 2026, 28(03), 026–043

Publication history: Received on 20 July 2026; revised on 31 August 2026; accepted on 02 September 2026

Article DOI: https://doi.org/10.30574/gjeta.2026.28.3.0220

ABSTRACT

Walking down the garden path, Named Data Networking (NDN) is a new networking approach based on the concept of content retrieval; it uses a unique name for a content request rather than retrieving data by host location (i.e., IP address-based retrieval). For NDN, timely and accurate caching of popular content along the path between producer and consumer will be a vital condition. The main concept of NDN is that each router stores data to quickly fulfil consumer requests; this is known as caching. Many cache replacement algorithms have been introduced to improve cache utilisation when the cache becomes full. This paper proposes a pending interest table-based cache replacement, i.e., PIT-REPL, which evicts the content from memory based on least score and further cache hit ratio enhance by round trip time based caching i.e. RTT-Cache and also developed a novel hybrid cache replacement strategy based on pending interest table and Q-Learning i.e. PITQ-REPL strategy to evict the content from memory based on cumulative reward which derive from score of pending interest table and Q-value reward of Q-learning. We found that the PIT-REPL cache hit ratio is enhanced by RTT-Cache-based cache replacement, i.e., RTT-Cache-PIT-REPL, and further cache hit ratio is enhanced by a hybrid approach of PIT and Q-learning strategy, i.e.,PITQ-REPL. Finally, we simulated using the ndnSIM simulator and found that the PIT-REPL replacement policy alone yields suboptimal cache hit ratios. However, the combined strategy of RTT-Cache with PIT-REPL enhances the cache hit ratio and reduces the latency, and finally hybrid strategy of PIT-REPL with Q-Learning significantly improves the cache hit ratios of RTT-cache-PIT-REPL and decreases the Latency up to some extent. Finally, the results show that the PITQ-REPL is a superior cache replacement to both strategies, namely RTT-cache-PIT-REPL and PIT-REPL.

Keywords: Named Data Networking, Cache Placement, PIT-Aware Cache Replacement, Cache Hit Ratio, ndnSIM

1 INTRODUCTION

Named Data Networking (NDN) represents a significant shift from traditional IP-based architectures, focusing on content rather than host addresses. The main concept of NDN is centered around data, It is one of the information centric network. The application areas of this architecture are web application, multimedia streaming and IoT [1]. This paradigm shift allows for more efficient data retrieval and distribution, as it enables routers to cache content closer to the end-users, ultimately improving latency and bandwidth utilisation. During retransmission of data packets under any circumstances, like packet lost or damage packets, they will be easily delivered to the consumer with instant responds. The caching reduces the load on the producer so that they can deal with more requests instead of the same consumers[2]. Although several research papers have been published regarding cache replacement and cache placement so far in NDN but many of these algorithms are primarily based on synthetic traffic models and aggregate performance metrics. Even in the strategies developed with ndnSIM they are all based on CS-centric evaluations without considering the detail forwarding information available in the Pending Interest Table (PIT). The use of PIT gives the real consumer demand and ongoing traffic flows. The research, which is based on this concept, was underexplored. So this paper objective to develop the caching strategy that enhances the cache replacement decisions based on PIT by incorporating real-time Round Trip Time (RTT) cache placement.

The traditional approach of caching techniques such as FIFO, RR, LRU, and LFU is not so efficient in the context of Named Data Networking, as they often fail to account for adaptability in terms of content popularity and access patterns. Also, these techniques (FIFO, RR, LRU, LFU) are static and do not consider network metadata when making eviction decisions. In named data networking, three tables are used: i) PIT, ii) FIB, iii) CS with the use of the first two tables, which are maintained by NDN routers, we can make the caching strategy adaptable in the field of decision making based on real-time data as it is updated dynamically during the whole process of communication. In this paper, we propose a PIT-aware eviction strategy named PIT-REPL, which evicts the content from the content store based on a computed score which derived exclusively from actual raw data of the PIT table. The score is calculated by multiplying two information: 1) the Total number of the same requests from all face-ID, and 2) total number of different unique face-ID that generated the same interest. Based on this formula, the proposed eviction works in the content store. After this when we simulate the new PIT-REPL replacement then we got the less cache hit ratio, to improve the cache hit ratio we implemented the selective caching named as RTT-Cache which is based on round-trip time when we combine both in simulator we got improvement in the cache hit ratio and also decreased in the latency i.e. RTT-Cache with PIT-REPL achieves higher cache hit ratio at small and mid-size caches up to CS-50.

2 LITERATURE REVIEW

In the study [3], authors emphasize of integration of content size adjustment and caching policy for to the improvement of named data networking. In study [4], authors gives the overview of all modern caching strategies in named data networking such as content popularity caching ,Geo-based caching, Joint optimization technique, In network caching, content replication based caching etc. but all techniques are based on different approach rather than direct applying PIT or FIB tables real time meta data also the authors discussed about the research gap where told how the popularity of dynamic content is investigated and how to use artificial intelligence to develop the caching techniques which are simple and fast. In study [5], authors proposed the probability based caching to share the resource fairly and reduce the caching redundancy. In study [4], authors proposed pre-caching strategy based on size of content chunk in which local activity and sojourn time is used for cache placement and cache replacement. In the study[6], authors are used Graph Neural network for to achieve the higher cache hit ratio. In the study[7], authors give the comparative analysis of popularity based caching strategies. The content popularity obtained by the parameters like hit ratio, content redundancy, content diversity ratio and stretch ratio. After comparison authors found that compound popularity content caching strategy (CPCCS) has performed outstanding in comparison of others. The main idea behind the caching in this paper is based on bandwidth of link, cache capacity, content diversity, stretch ratio. In the same paper authors gives ideas of all different kinds of caching algorithm for example MAGIC algorithm is based on local information of interest like content name, Interest count and popularity count. Where the Hop based probabilistic caching (HPC) algorithm enhance the content caching mechanism in in view of reducing content redundancy for that it use the parameter to measure the number of hops between content provider and consumer. Authors also discussed about CPCCS use the caching the content based on the parameters of distance between the provider and the consumers and duration that identifies how long a transmitted content can stay at nodes. In the same paper the MPC algorithm use another table known as popularity table which contain the information of content name, popularity and threshold. In the study[8], authors proposed global cache replacement algorithm based on exergy which is a parameter that evaluate energy quality in this it help to determine whether to cache content based on probability of cache content's exergy value. It is basically uses the concept of frequencies at which interest packets arrive. In the study [9], authors proposed a dynamic approach of caching based on deep transfer learning (DTL) for vehicular network. This method uses a time varying mechanism to predict content popularity in highly dynamic environment. DTL method required large data set for training purpose. In the study [10], authors proposed novel data caching method called Cache Aging with Learning

(CAL) for information centric network in internet of things environment, it uses the artificial neural network (ANN) to enhance the data freshness, improve cache hit ratios, reduce network traffic load and decrease the data retrieval delays by intelligently managing cache data. In the study[11], focuses on developing intelligent caching strategies for 5G edge networks. The proposed architecture aims to make smarter caching decisions by analysing user behaviour, network conditions and content popularity trends. The caching methods involve a data-driven prediction system that uses machine learning algorithm to anticipate future content requirement. It take the data from "5GEdgeData" dataset, user access logs, network statistics then refine it and use in machine learning for further processing. In the study[12], authors proposed a new cache replacement policy in named data networking based on FIB table information called Discard of Fast Retrievable Content (DFRC) in which content with less retrieval time should be evicted. All above algorithms are used to obtain the caching but no one algorithm is used the PIT table directly in the process to deal with real time Meta data for cache replacement strategy. This gap motivates me to develop the cache strategy which is based on real network data during sending interest and getting response from producers.

3 SYSTEM OVERVIEW

The main architecture of Name data network is centered around the concept of content-centric networking, where data is retrieved based on its name rather than its location. NDN is basically has seven components.

- Interest Packet
- Data Packet
- Consumer
- Producer
- Pending Interest Table (PIT)
- Forwarding Information Base (FIB) CS (Content store)

An interest packet is a request made by a set of strings in which specific data content is mentioned. A data packet is made of the name of the content and the actual data payload with the content signature, which is going to be delivered. A consumer is a node that generates the request for content. A producer is a node that produces the content payload. PIT is a table that maintains pending requests of consumers. FIB is a table that provides information on forwarding the request to the next hop. CS is a memory where data is stored or cached for fulfilling the request locally without forwarding the request to the original data producer [13]. The content store capacity will be measure in terms of the number of entries of packets; its default size is 65536. As in fig. 1 shows the basic architecture of NDN, where the consumers send their request for data to the producer with the help of routers. In this phenomenon, only Interest packet is used by the consumer to request a particular data contents from the network, once the network found the appropriate producer, it will send the requested data packet to the particular consumer. This shift in architecture allows for more efficient data retrieval and reduces the reliance on traditional IP-based routing methods, ultimately improving overall network performance and user experience. Each router has a small memory that is known as the content store, and also each router maintains the two tables PIT and FIB [14].

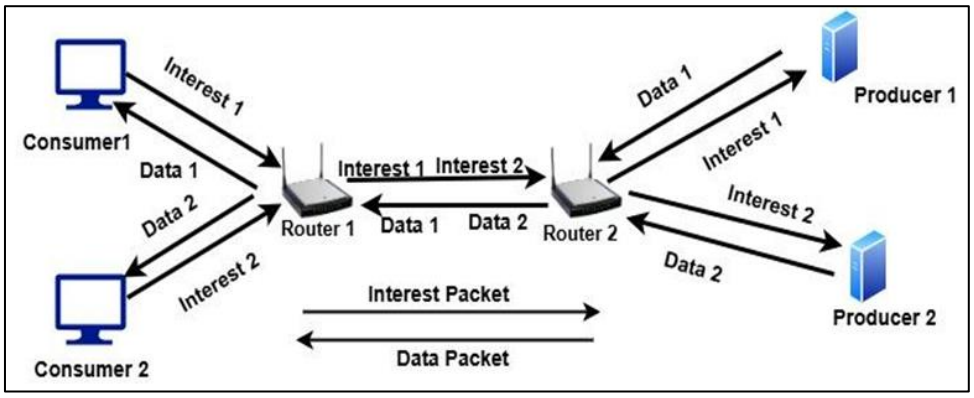

Figure 1: Basic NDN Architecture

The CS caches the incoming data content from the producer so that it can provide data to the consumer directly without request forwarding to the producer. PIT is the table maintains by the router which are responsible for maintaining the metadata of unfulfilled requests of interest packet, as well as the number of faces from which requests of interest packet are generated. These components all work together to manage data requests, cache content locally for faster access, and ensure that interests are forwarded efficiently throughout the network. This architecture not only enhances the speed of content delivery but also enables more dynamic and scalable network designs that can adapt to varying user demands and traffic patterns.

4 WORKING OF NDN

The given below Fig. 2, shows the working of NDN in which when a consumer wants to access a specific piece of content, it sends an interest packet containing the name of the desired data into the network. The router, which has CS (content Store) checks whether the content which was requested by the consumer is available in the cache. If it is found, the router retrieves the content from the CS and sends it back to the consumer, significantly reducing latency. If the content is not available in the cache, the interest packet is forwarded through the FIB to locate the producer that holds the requested data. Before forwarding to the next Hop, the router writes the pending interest in the PIT (Pending Interest Table), allowing it to track the interests that are still unresolved. This mechanism ensures that once the data is retrieved from the producer, all waiting consumers can receive the content simultaneously, further optimizing network efficiency and resource utilization. After maintaining the PIT, the router forwards the interest based on the FIB to the next appropriate hop, ensuring that the request reaches its destination promptly. This process not only enhances data retrieval speed but also minimizes unnecessary traffic across the network, contributing to a more streamlined and responsive communication system[15].

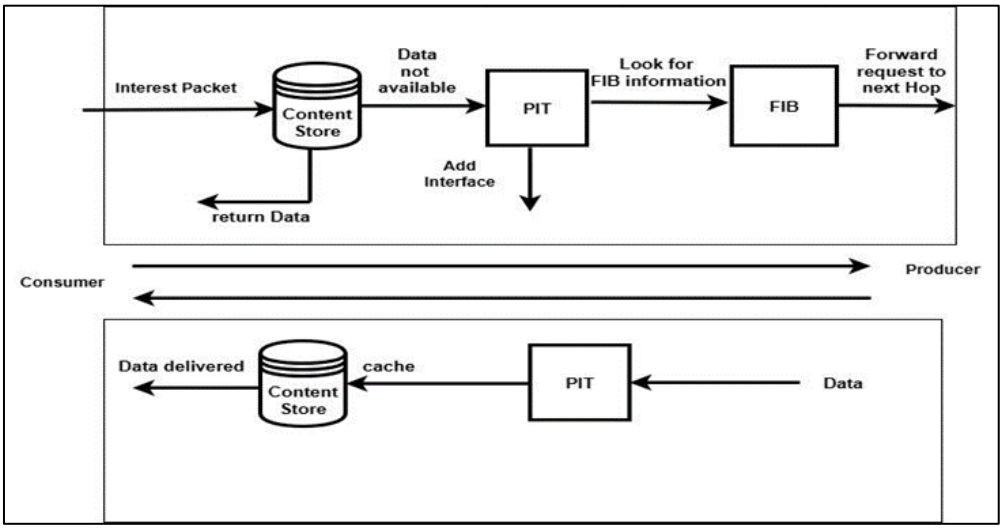

Figure 2: Caching Procedure [15]

4.1 Structure and Role of PIT (Pending Interest Table)

In NDN each router maintain pending interest table that keep the information's of interest those who are still not satisfied yet. PIT mainly includes forwarding information, unique identifiers and metrics which concern network performance. The PIT records details about each interest packet, which include the source and destination identifiers. It maintains a mapping of interests to their corresponding data packets which is used for data retrieval. PIT contains PIT entries which are used for forwarding. PIT table contain several PIT entries related to pending interest and Each PIT entries maintain two records named as In-record and Out record. As below Fig. 3, shows the hierarchy of PIT class. The detail fields of in-record and out-record are given in the table 1 and table 2 respectively.

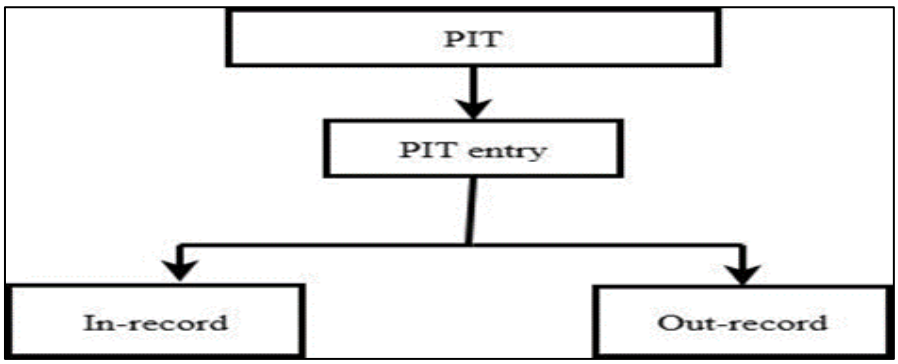

Figure 3: PIT components

4.2 In record

The in record tells the details about downstream Face. An In record inserted or updated by incoming interest pipe line. Table 1 gives the idea of fields related to in-record.

Table 1: In-record Information

a reference to the face	the Nonce in the last Interest packet from this face	the timestamp on which the last Interest packet from this face arrives	the last Interest packet
FaceId =258	144abe2f a79780ad	5861 (ms)	/root/seq=40

4.3 Out record

An out record inserted or updated by outgoing interest pipeline. An out record is deleted by incoming Data pipe line when a pending Interest is satisfied by a Data from that face. The out record entry expires when Interest Lifetime has ended after the last Interest packet is sent. Table 2 gives the idea of fields related to out-record. Where Table 3, gives all PIT information related fields with the combination or In-record and Out-record. Where table 4, gives the information that whenever the same request come their interest life time will set to 4000 milliseconds. No request should be beyond that life time.

Table 2: Out-record Information

a reference to the face	the Nonce in the last Interest packet to this face	the timestamp on which the last Interest packet to this face is sent	Nacked Id: indicates the last outgoing Interest has been Nacked; this eld also records the Nack reason

Table 3: Complete PIT Table contains both In-records & Out-records data for each individual Entry related to pending interest

Interest	In-records	Out-records
/root/seq=20	Face 257 Nonce:d37c5f1c Expiry: 19.1s Arrival: 18.7s	Face 259 Nonce: 4d579b23 Expiry: 19.5s
	Face 258 Nonce:fc0d73d7 Expiry: 19.5s Arrival: 18.9s	
/root/seq=21	Face:258 Nonce=e4c27092 Expiry=20.0s Arrival=19.8s	Face 259 Nonce=3bc468c3 Expiry=20.0s
/root/seq=22	Face 260 Nonce=87f5ffd7 Expiry=21.0s Arrival=20.9s	Face262 Nonce:40c6a7ab Expiry=21.2s
	Face 261 Nonce=9436d7a2 Expiry=21.2s, Arrival=21.0s	

Table 4: PIT table with its interest life time

Interest	Nonce	FaceID	Interest Life time
/data1/seq=22	61b00eff	257	4000 milliseconds
/data2/seq=10	4b1ef258	257,258	4000 milliseconds
/data3/seq=47	17a88f02	257	4000 milliseconds
/data1/seq=1	5e7b806c	257, 258	4000 milliseconds
/data3/seq=22	680fba56	257, 259	4000 milliseconds

The details of these PIT fields are explained below

4.3.1 Interest Name

These fields represent the all unique unsatisfied requests generated by consumer that are still awaiting corresponding data packets. Face-ID: The face fields are basically come with both records in-record and out-record (i) in-record Face and (ii) Out-record Face where in-record Face tells about the interfaces from which the interest packet was received and Out-record Face that tells about the interfaces through which interest packet forwarded for to get the data packet. The main point is that same interest packet can be generated with the multiple Face-ID.

4.3.2 Nonce

Nonce is a random number assigned to an interest packet. Its main purpose is to distinguish duplicate interests with the same name in the network. if the same interest with the same nonce then PIT considered it's a duplicate. If the same interest name arrives with a different nonce the pit treats it as a new interest even the same name of interest.

4.3.3 Timer

Each PIT entry contains one timer, the expiry timer. This timer is used by forwarding the pipe lines. It execute when the pit entry expires. When the in-record expiry time has expired the corresponding interest request has been deleted from PIT entry. The Time stamp told about in-record interest that at what time particular interest came from this face. Where the expiry time is the timestamp on which the last Interest packet to this face is sent.

4.3.4 NackedId

The NackedId field indicates that the last outgoing Interest packet has received a NACK (Negative Acknowledgement) from the upstream node. [16]

5 PROPOSED WORK

To implement the PIT based cache replacement we required number of files like pit-in-record.cpp, pit-in-record.hpp, newpolicy.cpp and newpolicy.hpp files in ndnSIM. Calculate the total score of each stored data item of content store for to take the decision of eviction. PIT-score=Total number of same request from the different Face-ID * Total number of unique FaceID requested for the same interest.The mathematical representation of above we can write let the interest packet I, pending in the PIT. Let F={f1,f2,f3…….fk} be the set of unique ID's that request the same interest I. Let n(fi) be the number of pending requests arriving from the face fi, for the same interest I. then PIT score is

$$\mathbf{PITScore}(I) = \left(\sum_{f_i \in F} n(f_i)\right) \cdot |F| \qquad (1)$$

Where $\left(\sum_{fi\in F} n(fi)\right)$ first term denote total number of request including the duplicate. And the second term $|F|$ is number of unique faces requesting the same interest. Note: We can derive numerous of formula to take the decision with the help of Pending Interest Table. Here total request count for that interest regardless of Face ID* Total number of unique consumer requested. If we multiply both we actually computes popularity from both dimension. PIT-score gives the information of popularity of Data item as per table 5.

Table 5: Popularity table

Interest Name	Total Requests	Unique Faces	PIT-Score=Requests * Faces
/root/seq=25	10	1	10
/root/seq=26	5	7	35
/root/seq=27	12	3	36

We can decide the content popularity for caching in three different ways based on score related information

- High request count, few faces means that might not be worth for caching.
- Moderate requests, many faces mean strong data to cache inside the content store.
- High requests, many faces mean highest priority to keep in the cache.

So in short we can say that total requests are equivalent to intensity of demands; unique faces equivalent to spread of demand and score is equivalent to overall popularity across network. So now based on the above score we can create the replacement policy i.e. when the new data come and cache is full then evict the data from cache which has lowest score. If more than one data which have same score then choose the first one for the eviction. Below fig. 4 and fig. 5 are the flow charts related to PIT based replacement selective cache of RTT respectively. The second approach is based on Round trip time between the events of Interest packet generation and reception of corresponding data. This RTT is used to calculate the threshold value. RTT threshold is use as a decision boundary used to take the decision on newly arrived data item from producer should cache or not. It means cache the data into the content store only when value of RTT is less than of RTT threshold of all data item previously received so far, because large RTT means data was likely fetch from distant node, more congested path, this caching strategy known as RTT-Cache as per fig. 5. The mathematical expressions are written as given in equation 2, 3 and 4. Let Di represent the i^{th} data packet received by the router, and let RTT_i corresponding round trip time measured between the interest transmission and Data reception. The average round trip time of all previously received data packets is defined as equation three and the caching decision variable C_i defined as equation four. The threshold value is used to determine the caching of data item in the content store which is represented with below equation two.

$RTT_{th}=\mu+\sigma$ (2)

where μ = mean RTT of previously received data packets

σ = standard deviation of RTT value

1. Mean RTT

$\mu=1/n\Sigma rtt_i$

2. Standard deviation

$\sigma=\sqrt{1/n\sum(rtt_i-\mu)^2}$

Threshold:
$RTT_{th}=\mu+\sigma$

Then cache only if rtt_ms< RTT_{th}

$$\overline{RTT}_{i-1}=\frac{1}{i-1}\sum_{k=1}^{i-1}RTT_k \quad (3)$$

$$C_i=\begin{cases}1, & RTT_i < RTT_{th}\\ 0, & \text{otherwise}\end{cases} \quad (4)$$

Where in equation four C_i =1 represents that Data packet permitted to store into content store, while C_i =0 means not to catch the data packet.

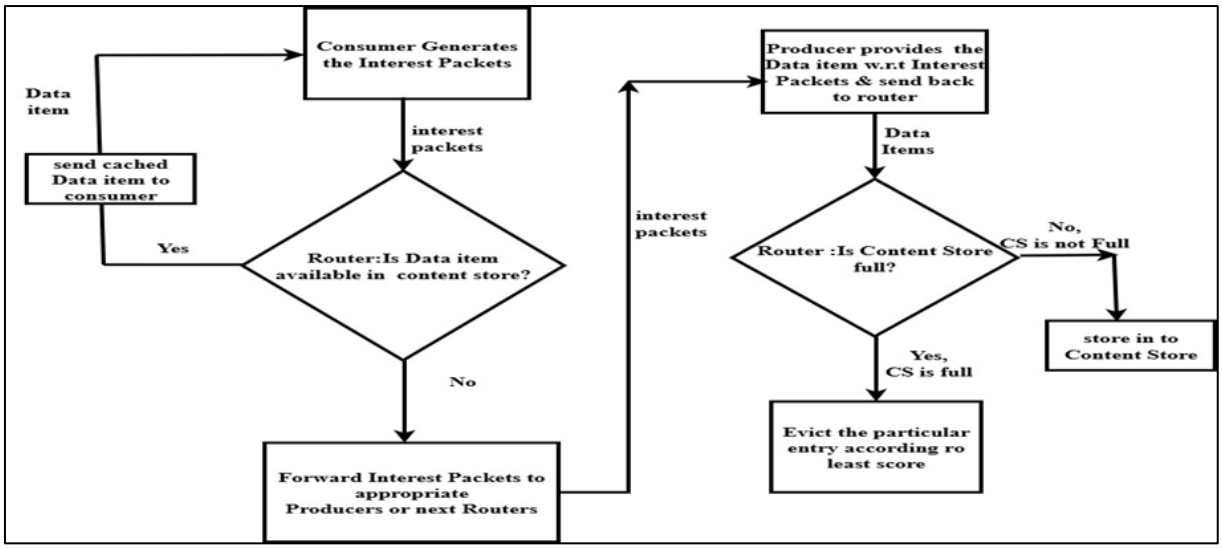

Figure 4: Flow chart of PIT-REPL cache replacement strategy

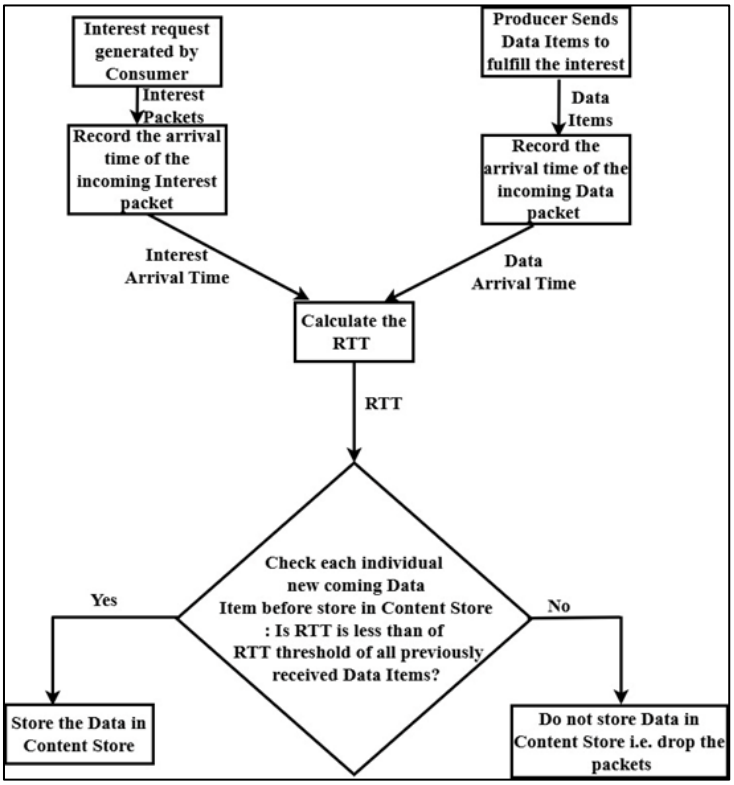

Figure 5: Flow chart of Selective Cache based on RTT

6 IMPLEMENTATION OF Q-LEARNING IN CACHE REPLACEMENT

The Q-learning is also known as model free learning. It is one of the techniques of reinforcement learning that works by learning on action-value function and it is developed by Watkins. This algorithm works by Q-table which maintains the Q-value of each state action pair. The three main element of Q-learning are state, action and reward. After each action the Q table will update according to Bellman equation[17]. When the researcher wants to apply the Q-learning in ndnSIM, they should to know about the cache replacement important functions to make own policy. In ndnSIM nfd::cs::Policy is the base class for all cache replacement policy implementations. The Policy class is also used to maintain the capacity limit of cache memory and this can be set via Policy::setLmit method. There are four public methods which are used to know what data packet added to cache memory, When data packet to remove from cache memory, What action is use to take after same data found in the cache memory, in order to maintain the capacity limit. These methods are given below.

- Policy::afterInsert is called after a new data item is inserted in to cache memory. So this happen when the cache is not full and wants to insert new data item, also when cache is full and want to insert the new data then also called that's why this function is beneficial after cache miss occurred. During each call of this method always check the capacity size and their limit.
- (ii) Policy::afterRefresh is called after existing data item in content store is again come from producer. It is use when cache hit occuered
- .(iii)Policy::beforeErase is called before of any data item is erased from cache memory.
- (iv)Policy::beforeUse is called when data item in cache memory is found by the searching. So we can write the function related to cache hit inside this method. apart from the above four methods, pure virtual function evictEntries must be define by researcher for to handle the capacity limit of cache memory. The evictEntries is used to decide which data item should be evicted from cache memory in order to maintain the capacity limit.The evictEntries method internally called beforeErase method. In this Q-Learning based strategy named as PITQ-REPL , the eviction is based on the score finally derived from both Q-learning and Pending Interest Table as per equation 5. for example

Score= QEvict + PIT-Score (5)

QEvict is the Q value of evict action which is calculated with the help of learning rule equation 5 and PIT-score is the score of popularity of content based on pending interest table. The final score is used to evict the content from content store that those content has lowest score should be evict. The updation rule of Q-learning in this case is given in equation number 6.

double newQ =oldQ + m_alpha * ((reward + pitScore)+m_gamma * maxNextQ - oldQ (6)

where newQ is the updated value of Q-table w.r.t. state and action. oldQ is the previous value of Q value that is before decision taking place in Q-table. m_alpha is the learning rate; reward is the feedback integer like for positive feedback give +5 reward for keep action (i.e. hit) and for negative feedback give -10 for evict action (i.e. miss) if we represent same above equation in actual mathematical form of Q-Learning including state and action manner then it looks like below equation 7.

$$Q_{\text{new}}(s,a) = Q_{\text{old}}(s,a) + \alpha\left[(r + \text{PITScore}(c)) + \gamma \max_{a'} Q(s',a') - Q_{\text{old}}(s,a)\right] \quad (7)$$

6.1 Example of Hybrid approach of PIT-REPL with Q-Learning Cache replacement Technique

suppose that we have two size of cache memory with three different request like 1,2,3 and the request 1,2,,3,2 come and its PIT score are 10,35,36 and 40 respectively then the Q-learning will work like below from table 6 to table 10.

Table 6: Initialization of Q-table

State of cache memory	Evict	Not to Evict i.e. keep
C(1,1) current state	0	0
C(1,2)	0	0
C(1,3)	0	0
C(2,2)	0	0
C(2,3)	0	0
C(3,3)	0	0

Request 1 come and the current state of cache memory is (1,1) and the next state is also (1,1) ,here hit occurred because 1 is already in the cache so reward is +5 and now update the Q-table according to bellman equation double newQ = oldQ + m_alpha * ((reward + pitScore) + m_gamma * maxa'NextQ(s',a') – oldQ(s,a); at current state (1,1)

maxQ((1,1)) = maximum Q value of each possible action of next state. newQ(1,1)=0+0.9(+5+10)+.85*0-0 newQ(1,1)=0.9(15)+0.85*0

newQ(1,1)=.9(15)+0
new Q(1,1)=13.5

Table 7: Updated Q-table after process the request 1

State of cache memory	Evict	Not to Evict i.e. keep
C(1,1) current state	0	13.5
C(1,2)	0	0
C(1,3)	0	0
C(2,2)	0	0
C(2,3)	0	0
C(3,3)	0	0

REQUEST

Next request is 2it's a cache miss then evict the 1 from the state (1,1) and it became (1,2) and update the table when reward is -10

- next Q=oldQ+.9(r+PITscore)+.85(0-0)
- nextQ=0+.9(-10+35)+.85(0-0)
- nextQ=.9(25)
  nexQ=22.5

now current state is (1,2) and the previous state Q value updated accordingly

Table 8: updated Q-table after process the next request 2

State of cache memory	Evict	Not to Evict i.e. keep
C(1,1)	22.5	13.5
C(1,2) current state	0	0
C(1,3)	0	0
C(2,2)	0	0
C(2,3)	0	0
C(3,3)	0	0

REQUEST

if the current state is (1,2) and 3 request come then the next state either (1,3) or (2,3)

the maximum Q value of each action at next state

MaxQ[(1,3),E].[Q[(1,3),NE]
Max(0,0)
=0
the maximum Q value of each action at next state

MaxQ[(2,3),E].[Q[(2,3),NE]
Max(0,0)
=0

this is tie so choose the next state as per FIFO lets suppose (1,3) then update the Q table and reach to next state (1,3) accordingly.

for(1,3).

nextQ((1,2),E)=0+.9(-10+36)+.85(0-0)
nextQ((1,2),E)=0+.9*16+0

nextQ((1,2),E)=14.4

and the now current state become (1,3)

Table 9: Updated Q-table after process the request 3

State of cache memory	Evict	Not to Evict i.e. keep
C(1,1)	22.5	13.5
C(1,2)	14.4	0
A(1,3) current state	0	0
C(2,2)	0	0
C(2,3)	0	0
C(3,3)	0	0

Again 2 come then the next state is the maximum Q value with each possible action the next state is depends on the maximum q value of state (1,2) and (2,3) with each possible action.

Max[Q((1,2),E),Q((1,2),NE)] and

Max[Q((2,3),E) , Q((2,3),NE)]

Max(14.4,0 and 0,0)

i.e. 14.4 it means the next state is C(1,2)

then the next (1,2)

nextQ=14.4+0.9(-10+40)+0.85(0-14.4)

nextQ=14.4+27+(-12.24) nextQ=58.32

now current state is (1,2)

Table 10: Updated Q-table after process the request 2

State of cache memory	Evict	Not to Evict i.e. keep
C(1,1)	22.5	13.5
C(1,2) current state	14.4	0
A(1,3)	58.32	0
C(2,2)	0	0
C(2,3)	0	0
C(3,3)	0	0

Since Q-learning is iterative process, it will update the Q-table and take the decision accordingly.

7 EXPERIMENTAL SETUP

7.1 Overview of ndnSIM:

ndnSIM is an open source simulator for named data networking. It run on the Linux, Mac OS platform. The researchers need c++ programming language to simulate their own custom cache replacement algorithm. It is NS-3 based and the core operation is handled with NFD[4]**.** Where NFD has an intelligent forwarding module which includes of forwarding pipelines and forwarding strategies. Forwarding pipelines is nothing a series of processing unit of packet. It come to picture when particular event triggered like receiving an interest, when data is coming, outgoing interest, content store hit event, content store miss event ,outgoing data and unsolicited data related events etc. These pipelines used the strategy to take the decision like when and where packets to be forwarded. We can say that forwarding pipeline is basically collection of different pipelines and the packets are transferred from one pipeline to another pipeline, as shown in figure 6.we can calculate the round trip time (RTT) of each individual arrived data item from producer with the help of forwarding pipeline. Since it deals with packet data that's why it comes in network layer concept.

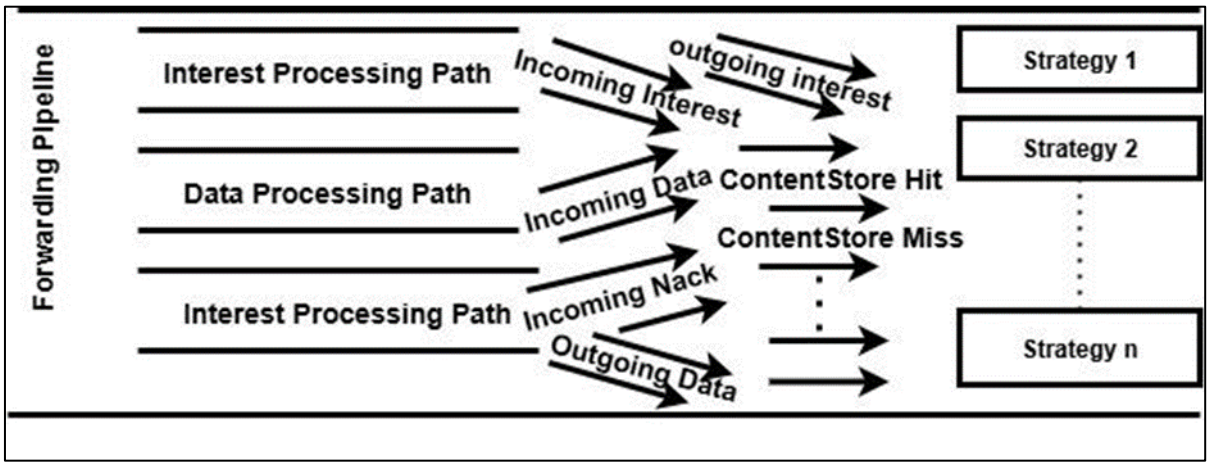

Figure 6: Overview of forwarding pipeline

7.2 Directory Structure of ndnSIM

The directory structure of the ndnSIM is as follows in terms of the implementation of the PIT-based cache replacement strategy. Direct information of such types is not available. Researchers need to do a little bit of coding in C++ to get the information related to PIT. Researchers need to declare their own variables and data structures in the pit-in-record.hpp header file of the in-record file so that they can use these variables and data structures in pit-in-record.cpp file. As we know, the in-record file is needed to obtain the information regarding the total number of the same requests from different face IDs, and the total number of different face IDs that generated the same request. The main objective for choosing the in-record is that, in coding-wise, it has a default function called inface. With this information, we can analyse the current situation of the named data network. To get this information used the data structure of type map that store the value in terms of key-value pairs, but when we want to count different numbers of keys from the total interest request, then we also require a set, which is also an associative container that stores unique elements. To create their own new policy for cache replacement, researchers need to create two files: one is a new-policy.hpp header file, and the second is a new policy.cpp class File. To create their own new replacement strategy, researchers need to inherit the cs-policy.hpp header File because this header File has declared the functions which are needed during implementation of new replacement strategy, the function look like (a) void afterInsert(EntryRef i); function is called to provide the acceptance of new data item inside the cache memory (b) void afterRefresh(EntryRef i); this function called the data item already present in cache memory and it just refresh the memory for to maintain integrity or we can say that take the action which reflect that same data has arrived again (c) void beforeErase(EntryRef i); this function is responsible do the task before evicting the particular data item from the cache (d) void beforeUse(EntryRef i); this function is called when the data is already in cache and going to satisfy the incoming interest, it basically use for the policy to update its meta data and (e) pure virtual function i.e. virtual void evictEntries()=0; this function is use to do the task that maintain the capacity limit of content store; these are all used to provide the action on content store. Now, depending on the researchers' needs, they can give the definition of the above functions. Sometimes few functions are enough for the implementation of the replacement, so all other functions do not need to be left as it is. To delete the data item from the content store, the emitSignal is required, which sends the signal with the proper index position to delete the item from memory. At last, when the researchers complete the development of a new policy then it should be

registered inside the ndn-stack-helper.cpp class File which is available in the helper sub directory of second level of ndnSIM directory. The figure 7 and 8 gives the idea about ndnSIM directory.

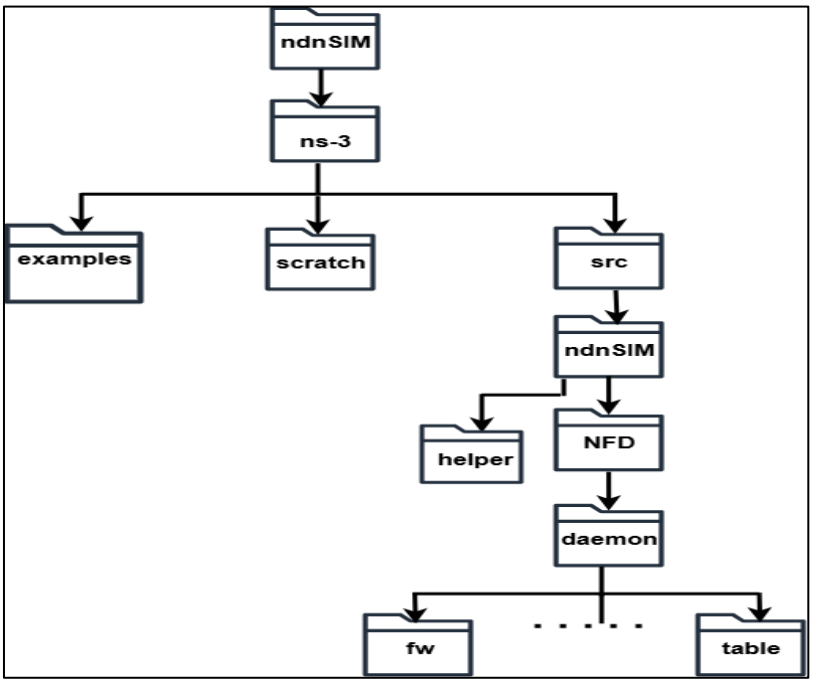

Figure 7: Directory Structure of ndnSIM

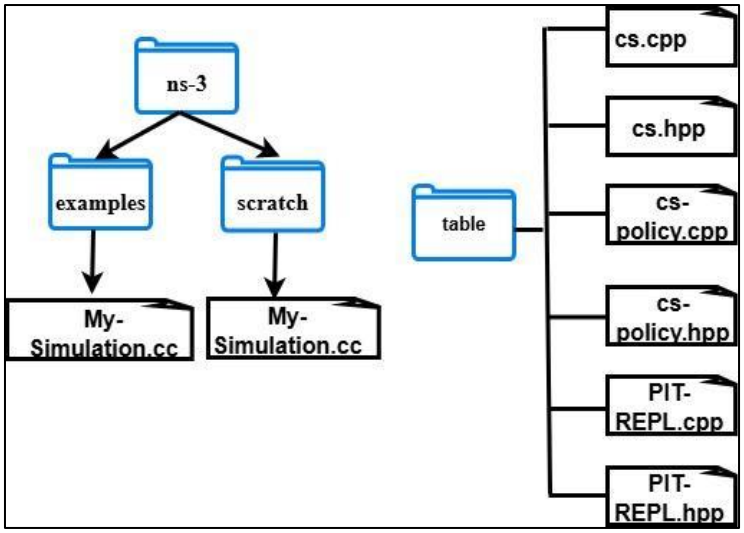

Figure 8: Information of c++ files in ndnSIM

7.3 Topology

In this experiment, the tree topology and linear topology are used where in tree topology there are four consumers, one producer and two routers arranged in a manner where the producer is on the root node, as shown below, i.e. fig. 9.In fig. 10, linear topology shown where 3 consumers, 2 routers and 3 producers are arranged in linear fashion. Each consumer sends the request as: /root/1, /root/2 to the producer, and when the producer gets the request, it will send the data packets to consumers through router 1 and router 2. All data items will be cached at the router node.

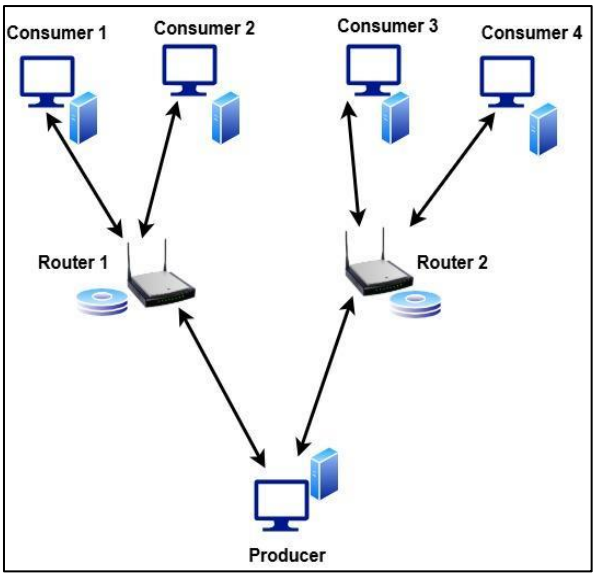

Figure 9: Tree Topology

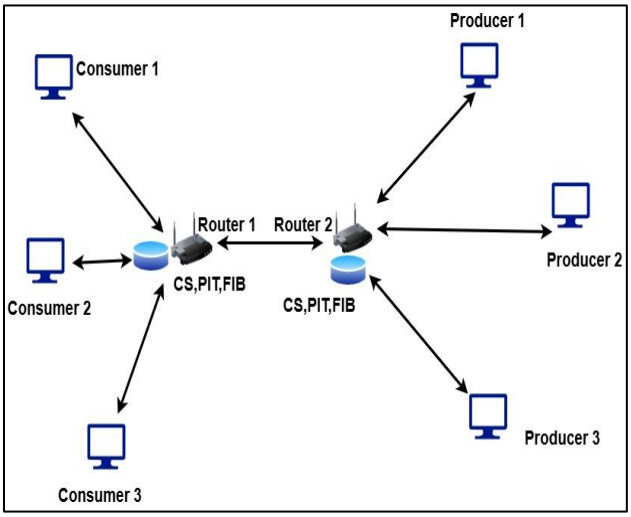

Figure 10: linear Topology

8 SIMULATION EXPERIMENTS

8.1 Simulation Parameters

To start the simulation with above concept we setup the environment with the help some parameters which is given below in table 11.In this experiment set the content store varying size as per table for to get the results.

Table 11: Simulation parameters

Parameter	Value
Request Rate	100 request per sec.
Number of contents	100 (Tree Topology)
	500(Linear Topology)
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	PIT-REPL, RTT-Cache-PIT-REPL,PITQ-REPL
Forwarding Strategy	Best Route (tree topology)
	Multicast (linear topology)
Simulation Time	100 s
Number of topology nodes	7(Tree Topology)
	8 (Linear Topology)
Learning rate	0.9
Discount Factor	0.85

8.2 Compilation and Execution

Compile the developed simulation class first go to the directory of ndnSIM then ns-3 after this, compile and execute with the help of ndnSIM command *./waf --run=createdclass --vis*

9 RESULTS AND DISCUSSION

The results obtained from the simulation are presented in the tables 12 and 13 as below.

Table 12: Average Cache Hit Ratio and Average Latency : PIT-REPL, RTT-Cache& PIT-REPL and PITQ-REPL in 0Tree Topology

CS-Size	Avg CHR of PIT-REPL	Avg. CHR (RTT-Cache-PIT-REPL)	Avg. CHR of (PITQ-REPL)	Avg. Latency of PIT-REPL	Avg. Latency of RTT-Cache-PIT-REPL	Avg. Latency of PITQ-REPL
10	29.01	40.67	54.39	5140.531 us	4721.138 us	4257.686 us
20	52.51	58.16	66.41	4344.910 us	4121.633 us	3879.663 us
30	64.80	68.66	73.11	3934.439 us	3777.335 us	3675.899 us
40	72.97	75.97	78.62	3677.132 us	3556.047 us	3506.217 us
50	79.24	81.51	82.86	3477.035 us	3389.189 us	3372.097 us
60	84.03	85.61	86.38	3330.270 us	3264.737 us	3261.026 us
70	87.81	88.96	89.83	3212.071 us	3170.751 us	3153.029 us
80	90.92	92.52	92.66	3111.103 us	3045.527 us	3059.560 us
90	93.5	94.52	95.44	3025.553 us	2993.447 us	2975.831 us
100	96.05	97.35	97.42	2943.701 us	2896.142 us	2894.200 us

Table 13: Average Cache Hit Ratio and Average Latency: PIT-REPL, RTT-Cache& PIT-REPL and PITQ-REPL in Linear Topology

CS-Size	Avg CHR of PIT-REPL	Avg. CHR (RTT-Cache-PIT-REPL)	Avg. CHR of (PITQ-REPL)	Avg. Latency of PIT-REPL	Avg. Latency of RTT-Cache-PIT-REPL	Avg. Latency of PITQ-REPL
10	0	5.97	18.14	60440.784 us	55723.358 us	49441.205 us
20	5.52	13.23	25.68	56486.965 us	51210.265 us	45208.356 us
30	11.58	18.77	29.95	52598.824 us	47950.069 us	42976.318 us
40	16.02	22.64	33.61	49787.065 us	45938.121 us	41156.818 us
50	19.97	25.91	36.61	47564.782 us	44163.587 us	39778.254 us
60	22.92	29.28	38.78	45863.762 us	42672.425 us	38751.734 us
70	26.34	32.25	40.93	44249.011 us	41401.015 us	37759.832 us
80	28.74	34.42	42.76	43044.743 us	40442.511 us	36967.699 us
90	30.89	36.41	44.13	41995.473 us	39574.629 us	36415.437 us
100	33.32	37.47	45.66	40959.823 us	38952.227 us	35727.409 us

Tables 12 and 13 are given the results after simulating the above three strategies in ndnSIM, the performance of above three strategies are evaluated in terms of average cache hit ratio and latency. (a) Performance Analysis in Tree Topology: As shown in Table 12, the average cache hit ratio of strategy PITQ-REPL is higher than that of strategies RTT-cache-PIT-REPL and PIT-REPL in the tree topology. In terms of latency, the PITQ-REPL has the lowest latency among RTT-cache-PIT-REPL and PIT-REPL across all content store sizes. Where RTT-cache-PIT-REPL has increased the CHR and decreased the average delay in PIT-REPL. The same is depicted in figures 11 and 13, in terms of average CHR and average latency respectively.

(b) Performance Analysis in Linear Topology: In linear topology as per table 13, where more diverse request is generated by consumer in order to create the unfavourable environment to all caching strategies via use of more than two namespaces, the PITQ-REPL has still achieved highest average CHR and lowest average delay obtained w.r.t RTT-cache-PIT-REPL and PIT-REPL. The same is depicted in figures 12 and 14, in terms of average CHR and average latency respectively. So in both topologies the RTT-cache-PIT-REPL has consistently achieves the second best performance in terms of average CHR and average latency, outperforming PIT-REPL while remaining below than PITQ-REPL.

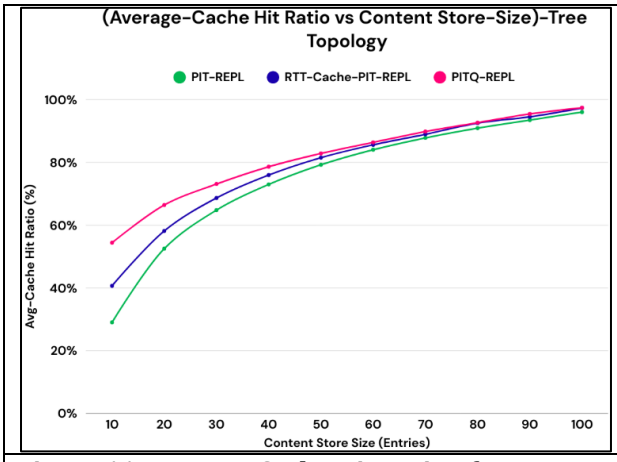

Figure 11: Average Cache Hit Ratio of PIT-REPL,RTT-Cache with PIT-REPL and PITQ-REPL in tree topology

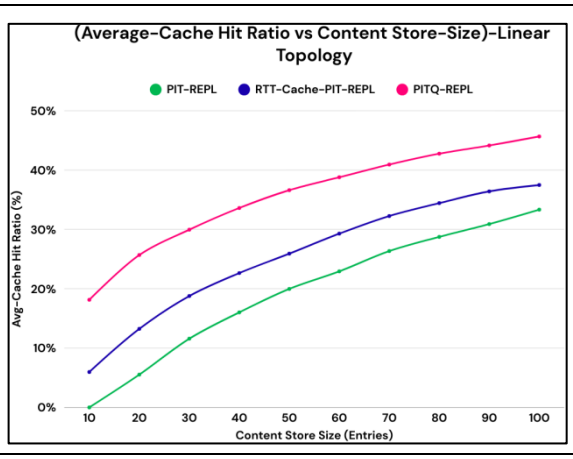

Figure 12: Average Cache Hit Ratio of PIT-REPL,RTT-Cache with PIT-REPL and PITQ-REPL in linear topology

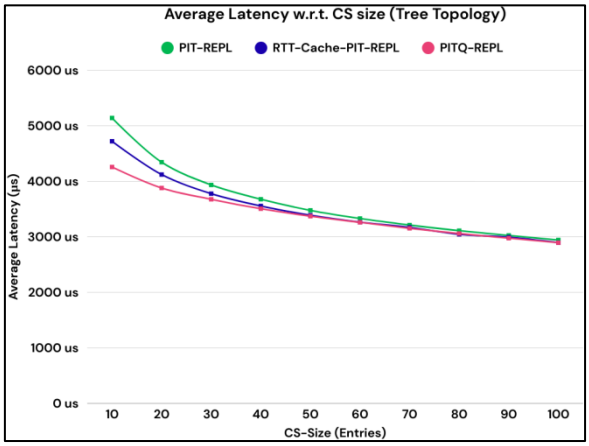

Figure 13: Average latency of PIT-REPL,RTT-Cache with PIT-REPL and PITQ-REPL in tree topology

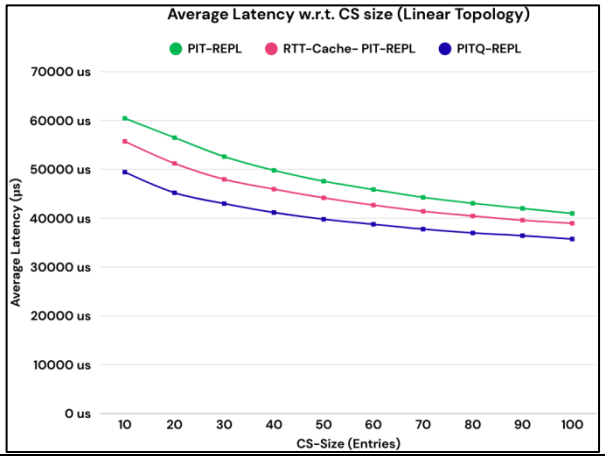

Figure 14: Average latency of PIT-REPL,RTT-Cache with PIT-REPL and PITQ-REPL in linear topology

10 CONCLUSION AND FUTURE WORK

No doubt, Named Data Networking is the future architecture for data sharing over the internet. In this architecture, every router maintains three fundamental data structures, namely PIT, FIB, and CS. As we understand, NDN's performance primarily depends on effective caching of data for consumers, so that for the same request, the nearest router can fulfil it without reconnecting to producers every time. But the problem with this concept is that the content store's memory is limited, so when it becomes full, the cache replacement algorithm helps replace old data with new incoming data. As we know, traditional caching strategies are not useful in this case because they are static, so the Pending Interest Table can be used to design new cache replacement algorithms for the new type that support network adaptability. In this research paper, we found that the hybrid strategy PITQ-REPL has a high average cache hit ratio and low average latency compared to the PIT-REPL, RTT-Cache-PIT-REPL strategy; hence, we conclude that the cache hit ratio of the PIT-based cache replacement technique can be improved with the help of both machine learning and a selective cache strategy.

In this work, equal weight is given to the eviction Q-value (qEvict) and the PIT-based score when calculating the final cache-replacement score. This reflects a balanced approach to both learned content utility and network-based demand. In future work, research could be explore on adaptive weighting of both parameters to obtain the optimal score in order to cache replacement strategy and also researchers are needed to design various types of cache placement and replacement strategies in addition with diversity of content in memory that will be based on a combination of two data structures, like PIT & FIB table, and try to find which concept of forwarding strategies will be helpful during the enhancement performance of named data networking.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Acknowledgments

The authors thank Suresh Gyan Vihar University for institutional support. All research design, analysis, interpretation, and intellectual contributions are the authors' original work.

Disclosure of conflict of interest

No conflict of interest to be disclosed.

REFERENCES

[1] H. Dai, B. Liu, H. Yuan, P. Crowley, and J. Lu, “Analysis of tandem PIT and CS with non-zero download delay,” in IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, Atlanta, GA, USA: IEEE, May 2017, pp. 1–9. doi: 10.1109/INFOCOM.2017.8057191.

[2] L. Zhang et al., “Named data networking,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, pp. 66–73, July 2014, doi: 10.1145/2656877.2656887.

[3] R. M. Negara, N. Rachmana Syambas, and E. Mulyana, "Integration Of Content Size Adjustment And Caching Policy to Improve Named Data Networking Performance," in 2022 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob), Bandung, Indonesia: IEEE, Dec. 2022, pp. 1–5. doi: 10.1109/apwimob56856.2022.10014284.

[4] 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.

[5] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in Proceedings of the second edition of the ICN workshop on Information-centric networking, Helsinki Finland: ACM, Aug. 2012, pp. 55–60. doi: 10.1145/2342488.2342501.

[6] J. Hou, H. Xia, H. Lu, and A. Nayak, "A Graph Neural Network Approach for Caching Performance Optimization in NDN Networks," IEEE Access, vol. 10, pp. 112657–112668, 2022, doi: 10.1109/ACCESS.2022.3217236.

[7] M. A. Naeem, M. A. U. Rehman, R. Ullah, and B.-S. Kim, "A Comparative Performance Analysis of Popularity-Based Caching Strategies in Named Data Networking," IEEE Access, vol. 8, pp. 50057–50077, 2020, doi: 10.1109/ACCESS.2020.2980385.

[8] H. Zhang, R. Xie, S. Zhu, T. Huang, and Y. Liu, "DENA: An Intelligent Content Discovery System Used in Named Data Networking," IEEE Access, vol. 4, pp. 9093–9107, 2016, doi: 10.1109/ACCESS.2016.2638474.

[9] M. Wasim Abbas Ashraf, A. Raza, A. R. Singh, R. Singh Rathore, I. W. Damaj, and H. Herbert Song, "Intelligent Caching Based on Popular Content in Vehicular Networks: A Deep Transfer Learning Approach," IEEE Trans. Intell. Transport. Syst., vol. 25, no. 12, pp. 20643–20656, Dec. 2024, doi: 10.1109/tits.2024.3445640.

[10] N. Hazrati, S. Pirahesh, B. Arasteh, S. S. Sefati, O. Fratu, and S. Halunga, "Cache Aging with Learning (CAL): A Freshness-Based Data Caching Method for Information-Centric Networking on the Internet of Things (IoT)," Future Internet, vol. 17, no. 1, p. 11, Jan. 2025, doi: 10.3390/fi17010011.

[11] N. V. R. Guduri, S. Sethu, R. Shalinirajan, R. Reena, K. M, and G. S. Uthayakumar, "Intelligent Caching Strategies for 5G Edge Networks using Machine Learning," in 2023 4th International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India: IEEE, Sept. 2023, pp. 44–49. doi: 10.1109/icosec58147.2023.10275847.

[12] 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.

[13] S. Al-Ahmadi, "A New Efficient Cache Replacement Strategy for Named Data Networking," IJCNC, vol. 13, no. 5, pp. 19–35, Sept. 2021, doi: 10.5121/ijcnc.2021.13502.

[14] L. C. M. Hurali and A. P. Patil, "Application Areas of Information-Centric Networking: State-of-the-Art and Challenges," IEEE Access, vol. 10, pp. 122431–122446, 2022, doi: 10.1109/ACCESS.2022.3223667.

[15] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the Evolution of ndnSIM," vol. 47, no. 3, 2017.

[16] Alexander Afanasyev and Junxiao Shi, "NFD Developer's Guide," Named Data Networking Project, Aug. 2021. [Online]. Available: https://named-data.net/publications/techreports/

[17] S Manju and M Punithavalli, "An analysis of Q-learning algorithms with strategies of reward function," International Journal on Computer Science and Engineering, vol. 3, no. 2, pp. 814–820, 2011.