CacheQueue: Efficient Cache Queue Usage in a NDN

ABSTRACT

In a Named Data Network (NDN), contents are cached in-network nodes to satisfy interest requested by consumers quickly. Hence, the caching policy is important for the efficient use of cache queues and content delivery. Cache replacement strategy plays an important role in better content search in node cache as well. In our research, we propose a cache eviction policy with the goal of making the best usage of the memory queues of NDN nodes. Our method selects the worst packet for eviction and retains the popular packets, recently requested packets and wipes out unsolicited packets. We provide supporting results produced from simulation study using NS3 based NDN network simulator. We find that our method reduces the average number of hop counts and increases cache hits which is useful for any large scaled distributed heterogeneous network.

CCS Concepts

•**Networks** → *Content Caching;*

Keywords

Caching, NDN

1. INTRODUCTION

In a Named Data Network (NDN), some nodes known as *consumers* those broadcast their *interest*. Some of the nodes called *producers* serve those interests by sending the requested contents to those consumers. Some of these contents can be served from the in-network cache. The network nodes which satisfy these interests send data packet to the consumer nodes [1]. The nodes forward the packets are called *NDN router* nodes. *In-network caching* is a widely used term for content caching at intermediate nodes in NDN [4], [5], [8], and [11]. In-network caching plays very important role in information centric network which can be base of large scale future internet. Again, any good caching algorithm requires good cache replacement policy [9], [12]. Our design goal of cache replacement policy focuses on improving *cache hit ratio* for quick service delivery.

There are broader categories of in-network caching methods [13], [14], and [15], [16], [17]. One is *on-path-caching* which caches the contents at the intermediate nodes those fall on the way back from the server to the requester. The second one is *off-path-caching* where nodes within a certain range or all the nodes in the network are utilized to cache contents collaboratively. Third one is *edge-caching* where only the nodes at the boundary of a network are enabled with caching. However, there are some challenges involved in above mentioned caching strategies. Finding the optimal cache size, finding appropriate nodes for caching, low latency to retrieve a cached content, and finding a suitable cache replacement policy are very challenging issues in NDN caching [10]. Researchers have been working to resolve the challenges in this area for recent years [6].

Our paper is organized as follows. Section 2 provides some relevant research work, Section 3 provides problem formulation and solution approach, Section 4 provides results and analysis, and finally, Section 5 provides conclusion.

xxxx'xx, December, 20xx, xxx, xxxxx

ISBN xxxxxxxxxxx. . . $xx.00

DOI: xxxxxxxxxxxxxxxxxxxx

2. RELATED WORK

An efficient cache replacement policy is needed in any information centric network because cache storage for the nodes are limited. It is required to determine which content to be stored, which content to be replaced. In [6], the authors calculate file-level requests probability and the chunk-level request expectation index of a content. Then, they multiply them to find request expectation index of a content. Their cache replacement policy selects the item with the lowest index for eviction. In an NDN router, the chunk with the minimum index denotes that the chunk has the lowest expectation of being requested in future.

Least Valuable First (LVF) cache management policy utilizes three parameters - delay, popularity and packet drop to set a *value* per NDN requested object [14]. This value is dependent on the history of the user's requests received by in network routers within each operational cycle. However, keeping history is not always possible for the nodes due to limited storage.

For IoT, data are transient and frequently updated by the producers. As a consequence, copies are stored in caching nodes which may become 'out of date' after a short period of time. For content caching, a fixed delay time is used which may delete still valid content or keep out-of-date content in the cache for a long time. A good *cache replacement strategy* should adapt to rapid changes to meet on-demand

data delivery, and cannot be affected by the contents of the temporary alternate. To meet the aforementioned two demands, there is a policy which proposes a cache replacement policy based on content popularity [15]. Selection is decided based on previous popularity and content cache hits in current cycle. This work shows the performance of different caching policies in terms of cache hit ratio and the average server load achieved for a given topology with different cache sizes. The above mentioned scheme performs better compared to *Least Frequently Used* (*LFU*) and *Least Recently Used* (*LRU*). However, this method consumes most of the processing time for updating the table of cache which is not suitable for large scaled network [1], [2].

In existing cache replacement policies, there remains a number of pitfalls. There are also some issues which have not been considered yet and which influence caching performance. For example, some policies try to evict old contents, where as some policies try to find the less popular contents for eviction [3], [7], [17]. Again, few policies attempt to find the contents which have not been used for a certain time. However, none of them can ensure the best usage of cache.

Least Recently Used (LRU) is a basic caching strategy where the oldest data is replaced by the new coming data [2]. *Priority FIFO* based policy has been used in NDN in a smarter way. Usually, NDN nodes use three *queues* for the best selection for the eviction operation. However, it is not applicable for a large, diversified network topology or IOT network because the volume and types of contents get changed very frequently. If continuous checking is required to compute the priority of the content in the queues it causes a huge computation overload. Therefore, we provide a dynamic weighted system for content packet selection with low priority. We give *age score* for good usage of caches. We call our replacement strategy as *Hybrid Content Store* (*Hybrid-CS*) replacement method.

3. PROBLEM DOMAIN

After studying, existing cache replacement policies, it is evident that each policy considers a particular property of a content. While one works with popularity, other has considered age of a content. Besides some policies have tried with recent usage of contents. We have tried to design a policy considering all such attributes together. Our motto is to make our policy perform better in any kind of scenario like large topology, diversified topology, IOT based network. We are focused on designing a dynamic cache replacement strategy which will make the best usage of cache treating the contents appropriately according to the situation.
In the upcoming subsections, we will discuss about our solution approach. Moreover, we will provide some definitions and discuss the preliminaries related to our model. And firstly, we will start with the preliminaries.

3.1 Preliminaries

We make special use of the underlying data structures built in NDN node's *Content Store* (*CS*). CS's size varies in different nodes. When a node's CS reaches its capacity, our policy starts working for evicting contents to accommodate new contents. Moreover, one *hashmap* and three *queues* have been used to check *Primary State* of contents. We will talk the details as follows.

EntryInfoMap [1], [15] is the total collection of contents of a node. Using this map, content can be searched quickly. *FIFO Queue* is the collection of fresh contents. Recent contents are found in FIFO queue. It should be noted that each content is usually marked with a lifetime. When this lifetime expires content cannot stay in FIFO queue. *Stale Queue* contains the worn-out or time expired contents. The expiration time depends on the lifetime of the content in the *FIFO queue*. If lifetime exceeds *Freshness Period* then the content is sent to Stale queue. *Lifetime* is calculated from current time stamp and recently used time stamp. *Freshness Period* means how much lifetime is allowed to address a fresh or newly arrived content. *Unsolicited Queue* is like stale queue. It carries the most unexpected contents a node faces.

During the period of content selection, EntryInfoMap is accessed iteratively. From the map, the current position of a content can be found. Here, the current position means in which queue the content resides at that time. One of three queues discussed above (FIFO, Stale, Unsolicited) can host contents. At a time, at most one queue can host the content. However, content's position in the queue depends on the remaining lifetime or expired lifetime of the content. Whenever a content's property matches with one of the queue, it takes place in the specific queue and in fact it is found here. Thus, we can say *Primary State* has been determined.

Table 1: EntryInfoMap Content Table

Index	EntryInfoMap Content
1	root / interest / data / % 03
2	root / interest / data / % 04
3	root / interest / data / % 05
4	root / interest / data / % 06
5	root / interest / data / % 07
6	root / interest / data / % 08

Table 2: Unsolicited Queue Content Table

Index	Unsolicited Queue Content
1	root / interest / data / % 05

Table 3: Stale Queue Content Table

Index	Stale Queue Content
1	root / interest / data / % 03
2	root / interest / data / % 04

Table 4: FIFO Queue Content Table

Index	FIFO Queue Content
1	root/interest/data/ % 06
2	root/interest/data/ % 07
3	root/interest/data/ % 08

The four tables presented above are depicting previously discussed collections. For simplicity, the hashmap and the

queues used in our policy are discussed with the help of those tables. After searching the four tables infact the four collections the *primary state* of a content is determined. For more clearance, primary state is the current location of a content in any of the three queues. We will describe the scenario with example. We see, Index 1 and Index 2's contents of EntryInfoMap table are in *Stale* queue. Index 3's content is found in *Unsolicited* queue. Index 4,5 and 6's contents are found in *FIFO* queue. Therefore, if we consider Index 3's content, we evaluate its primary state as expired content due to its position in Stale queue. In upcoming processes, it will be treated under a scoring system (discussed in details in Section 2) according to its primary state.

We now introduce a simple data structure for content packet. Here, we call the most useful attribute of this data structure is *LastUsedTime*. It is updated with the help of default timeline of a NDN node. We use other NDN data structures *Pending Interest Table* (*PIT*), *Forward Interest Base* (*FIB*). We now provide our proposed content replacement policy. We explain with a small portion of network topology which is shown in Figure 1. The steps are explained as follows.

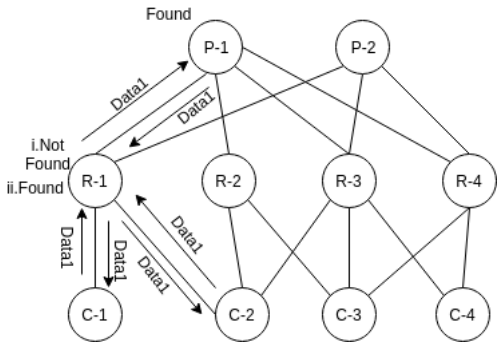

Figure 1: A sample network topology.

Finding Content: In Figure 1, we find that, C-1 requests data to R-1. But the content is not found in the content store (CS) of R-1. Now, R-1 forwards the interest to P-1. In P-1's content store, the content is found.

Forwarding Content: After finding the content, the content follows the reverse path of the request packet to reach its consumer C-1. In the path, firstly it is accessed by R-1. As R-1 does not contain the content in its CS, it allows insertion of the content if it's content store has space. Otherwise, our proposed Hybrid-CS policy is called to select the content for eviction.

3.1.1 Hybrid Policy

We now describe the policy in details. Figure 1 demonstrates another scenario. For example, C-2 also requests for *data1* to R-1. Content is found in CS of R-1. Therefore, it does not need to forward further. Before providing the content to C-2, tasks are performed in this node R-1.
From the above discussions we have found some cases when Hybrid-CS policy is applicable. Caching can be done in two ways: *first insertion* and *periodical refreshing*.

First insertion means pushing a content for the first time in CS. *Periodical refreshing* means a content is already exists in a node and a request has appeared for the content. During *insertion phase*, our proposed model has to perform specific one or two tasks according to the situation. On the other hand, during *refreshing phase* it has only one task. Therefore, for the first time of insertion and each time of periodical refreshing, the *latest used time* of a content is updated. Using this age, we measure *relative elapsed time* of the content in the corresponding queue. After that an exponential function is applied on relative elapsed time for computing another weight. We provide the functions that grow in a dynamic way proportional to their size. Therefore, anytime one has a quantity that grows faster when it is larger, or shrinks slower when it is smaller. That is why we have used exponential function to maximize the score value of the content which has relatively large elapsed time and which has to minimize the score value for those which have relatively small elapsed time. Then two score are multiplied. Finally, a greedy approach is followed for the content with the maximum score and it is evicted. This greedy approach finds the content with the maximum score. Here, the content with highest mark is the most unwanted one at the particular moment and selected as a candidate for eviction.

3.2 Proposed Methodology

Here, *entryInfoMap* contains all the data packets of a node. The packets are distributed in *child queues*. We assume that a packet does not arrive at two or more queues at the same which is logical. Usually, when it is time for eviction *Unsolicited* queue gets first priority, then *Stale* and finally, FIFO provided that Unsolicited and Stale queues are empty.

In our policy, packet can be removed from any queue with respect to some value. Firstly, we discuss the scoring process or how to assign the weight. For a packet which is in *Unsolicited* queue, gets a value u. Let us assign $m1 = u$. If the packet is in *Stale* queue, it gets value s. In this case, $m1 = s$. Finally for *FIFO* queue, the value will be f. Therefore, in this case $m1{=}f$. It is logical to assign weight as follows- $u > s > f$.

Typically contention is observed between *Stale* and FIFO. *Stale* gets higher priority because it contains comparatively more undesirable packets. Therefore, marking or scoring process looks quite straight forward which is aligned with the definition of queues. *Unsolicited* queue contains very few contents. When it is the time of decision, Stale gets priority over FIFO. However, it may happen that a node may contain huge contents. In that case, queue's waiting time will be more. For freshness period of a content, it may reside in FIFO queue for long time. But with continuous pushing of new contents it will fall very behind in the queue. Yet it is not thrown to *Stale* queue. In this case, we consider position in FIFO queue. This positional weighted will be multiplied by f in Equation 1. It is as follows.

$$f = f * \frac{contentPosition}{FIFOQueueSize} * P_c \quad (1)$$

We now define the role of P_c. We give an example. Let, the size of FIFO queue be 10000. We can consider 5 contents having position 200, 2000, 4000, 6000, 8000. The ratio of positions are 0.02, 0.2, 0.4, 0.6 and 0.8 respectively. If we only multiply this ratio with f, f becomes less which does not fulfill our motive. Therefore, we normalize. If we assume value for P_c is 10 then this value turns into 0.2, 2, 4, 6, and 8 respectively. This new weighted value is bias free because

any queue may score contents may get large (for example FIFO queue). Here, P_c may be tuned in testing.

For computing $m2$ we need three values- average time to fill up the CS, age of packet, relative elapsed time. The above mentioned three parameters are computed as follows.

$$AverageTime = \frac{CSsizeofNode}{InterestRate} \quad (2)$$

Here, *CS size* of a node is indicated by how many content packets a node can carry in its content store. *Interest rate* means the number of interest packets generated per second from the consumers.

$$Age = CurrentTime - LastUsedTime \quad (3)$$

Last used time refers to the time when a content satisfied an interest.

$$RelativeElapsedTime = \frac{Age}{Averagetime * weightFactor} \quad (4)$$

weightFactor is computed as follows.

$$weightFactor = InterestRate * \frac{1}{k} \quad (5)$$

Therefore, *Relative Elapsed Time* can be expressed as follows:

$$RelativeElapsedTime = \frac{Age * k}{Averagetime * InterestRate} \quad (6)$$

Relative Elapsed Time (RET) is one of the core concepts of our proposed policy. We explain with a scenario which defines the sage of RET. For example, content store size of node 1 is $S1$, content store size of node 2 is $S2$. Interest rate in node 1 is $I1$ and interest rate in node 2 id $I2$. We now consider $S1$ is greater than $S2$ and $I1$ is greater than $I2$. If the age of a content in node 1 is $A1$ and the age of a content in node 2 is $A2$, there may be following possible cases.

For first case, where $A1 = A2$, node 2's content relatively older than node 1. Because size of content store and interest rate for node 2 is less than node 1. Due to this limited resource, a content having same age as node 1 can not be considered as fresh as node 1. Similar logic is applied for the third case where A1 < A2. Just for the second case the content can not be said older for node 1. From this concept, RET has arrived. From Equation 6, it can be said that, age has been modified by the denominator. Here, denominator is the multiplication of *Average time* and *Interest Rate*. From Equation 2 denominator can be expressed as CS size of Node. If the denominator or CS size is increased RET will decrease, vice versa. Equation 6 numerator also contains a constant k. Its help is to get a whole number as a multiplier. It can be 10 or 10's multipliers. We are at the very end to get $m2$. Just we have to pass the *Relative Elapsed Time* in exponential function for getting the value of $m2$.

Finally, $m2$ can be computed as follows.

$$m2 = e^{RelativeElapsedTime} \quad (7)$$

From the property of exponential function, we find that it provides high growth rate for small unit change. We use it to get large value for $m2$ with respect to change of RET. There will be difference between the relative elapsed time of contents. Exponential function makes the difference higher. Actually we need large $m2$ which can be satisfied by exponential function. Finally, we multiply $m1$ and $m2$ to get a *total mark* on which decision making depends. After getting all contents' final score, we apply greedy approach to find the packet with the maximum *total mark*. Then, selected packet is removed from the cache. The detail is given in Figure 2.

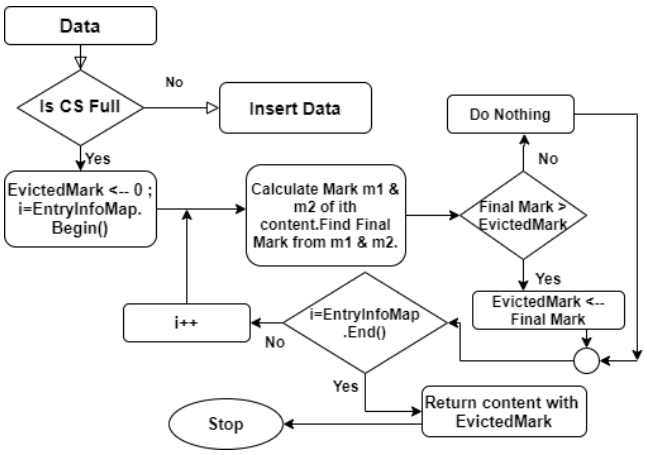

Figure 2: Flow chart of Hybrid-CS policy.

4. RESULTS AND ANALYSIS

In this section, we report the results and analysis. We first describe our testbed. We use the widely used ndnSIM based on NS3 [10]. The simulator is implemented in a modular fashion, it has built-in Pending Interest table (PIT), Forwarding Information Base (FIB), Content Store (CS) etc. We use packet size of 1024 bytes, nodes around 55 including producers and consumers. We evaluate the performance of our proposed method using performance metrics: *cache hit*, the number of *required hops*. We build mesh type topology randomly which has a set of consumer nodes, a set of producer nodes and a set of router nodes. We vary some network parameters to find the impact on the performance metrics. We vary *cache size, interest rate, payload size, topology.*

4.1 Impact on Cache Hit

Here, we report the impact on cache hit by varying different network parameters.

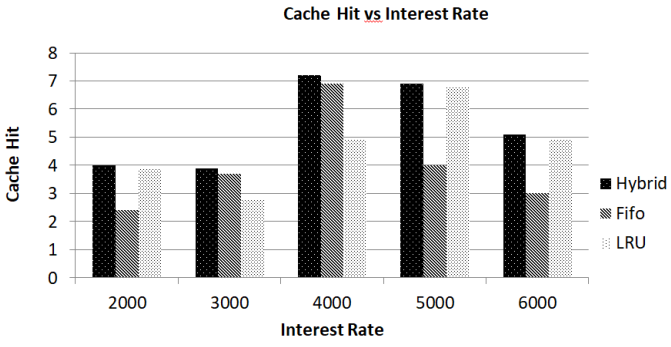

Figure 3: The impact of interest rate on cache hits.

We now vary *interest rate*. Cache can store 60000 packets and the payload size is 1024 bytes. In Figure 3, we find

that, our policy always wins against FIFO and LRU policy. In the beginning when interest rate is low, content store is available. Therefore, LRU competes better. However, with the increase of interest rate, our Hybrid-CS model gives better performance. In high interest rate, CS size of a node should be used properly. In this scenario, our policy makes the best use of CS size.

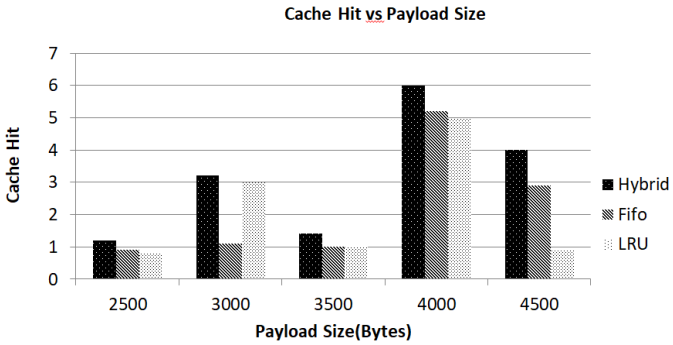

Figure 4: The impact of payload size on cache hits.

In Figure 4, we vary the payload size to see the impact on cache hit, cache size is of 60000 packets capacity, interest rate is 3000 packets/sec. Here, we find the better performance of our strategy as well.

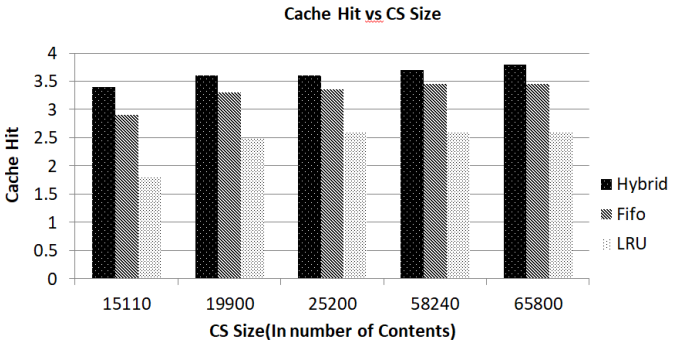

Figure 5: The impact of CS size on cache hits.

We report the impact on cache hit by varying cache sizes in Figure 5. Here, we set the payload size of 1024 bytes, interest rate to be 3000. With the increase of CS size, basically cache hit gets better (shown in Figure 5). Here, our policy also dominates the other two policies.

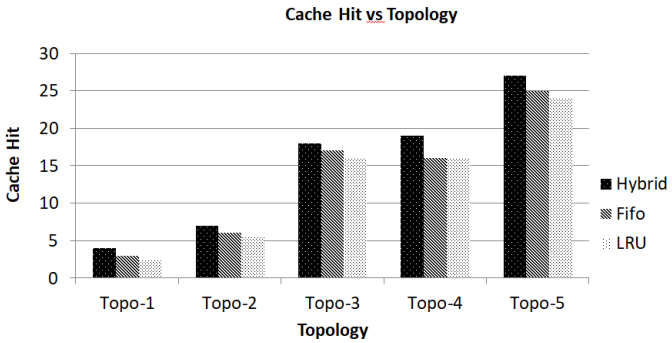

Figure 6: Topology and cache hits.

We record cache hits for different topology. We set cache size to 6000, payload size to 1024 bytes, interest rate to 3000. We have worked with five different types of topology. We also have varied number of nodes from 18 to 54. Each topology contains different sets of consumers, producers and forwarding routers. In different topology, our policy also performs better than LRU and FIFO (shown in Figure 6).

4.2 Impact on Hop Count

In this section, we provide the impact on hop count after varying different parameters of NDN.

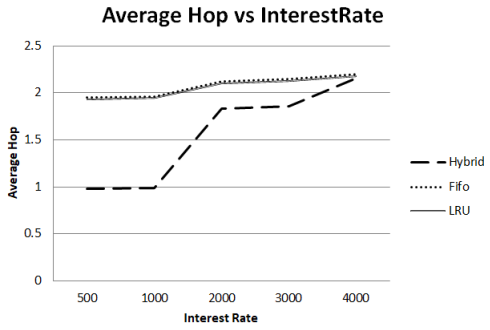

Figure 7: Average hop count vs. interest rate.

Here, interest rate has been varied for getting the impact on the average hop count. The number of nodes is set to 18, cache size is set to 6000 (cache can contain 6000 packets), payload size is of 1024 bytes. With the increase of interest rate, the average hop also increases which is natural. However, our proposed method requires the less number of hops than the other cache replacement policies.

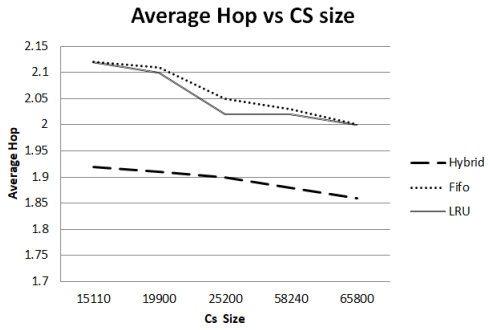

Figure 8: The impact of CS size on the hop count.

Average hop count with respect to different CS sizes can be reflected in Figure 8. Our method outperforms LRU and priority FIFO. Here, with the increase of CS size, the average hop decreases for priority FIFO and LRU. Large CS size is responsible for it. We find the less number of required hop counts than LRU and priority FIFO.

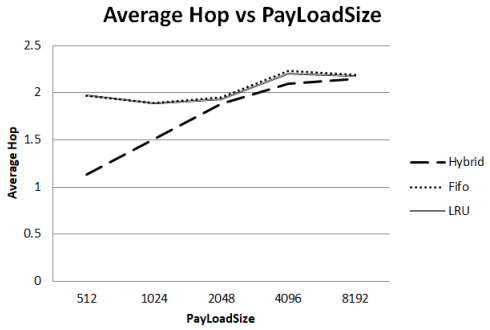

Figure 9: The impact of payload size on the hop count.

We now provide the impact of payload size on the hop count. Payload size has been varied in Figure 9. We find that our policy also responses well in terms of the average number of hop counts.

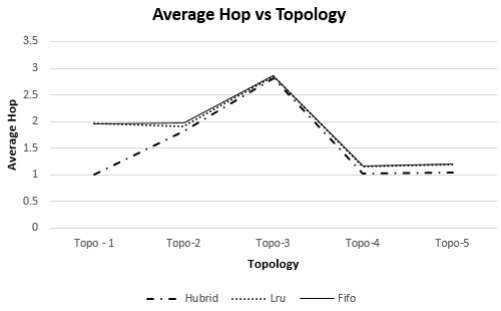

Figure 10: Topology and the hop count.

We now provide the impact of topology on the hop count. We find that our policy performs better when we scale up the number of nodes. Here, the average hop count has been reported for different topology in the following environment where cache size is of 60000, payload size is of 1024 bytes, and interest rate is of 3000packets/sec. In Figure 10, the impact of topology on hop count is depicted where we find our hybrid method works better.

5. CONCLUSION

We provide an efficient cache replacement policy based on the importance of the content packet in queues in cache. We find better usage of the space and duration in cache queue. Our method contributes to better content search service in a NDN. We find the reflection in cache hit and it required the less number of hop counts. Our method is scalable, and applicable for any large scale distributed heterogeneous network, specially for IOT where content demands change frequently.

6. REFERENCES

[1] Named Data Networking. Available: http://www.named-data.net/ucla. Last visited March 10, 2020.

[2] A. Afanasyev, I. Moiseenko, and L. Zhang, “ndnSIM: NDN simulator for NS-3,” Dept. Computer Science, University of California at Los Angeles, CA, USA, Tech. Rep. NDN-0005, 2012.

[3] W. K. Chai, D. He, I. Psaras, and G. Pavlou, “Cache “less for more” in information-centric networks,” Computer Communication, Elsevier, vol. 36, no. 7, pp. 758-770, 2013.

[4] J. Dai, Z. Hu, B. Li, J. Liu, and B. Li, “Collaborative hierarchical caching with dynamic request routing for massive content distribution,” in Proc. of IEEE International Conference on Computer Communications (INFOCOM), Florida, USA, March 2012.

[5] J. Ji, M. Xu, and Y. Yang, “Content-hierarchical intra-domain cooperative caching for information-centric networks,” in Proc. of ACM International Conference on Future Internet Technology, Tokyo, Japan, June 2014.

[6] H. Li, H. Nakazato and S. H. Ahmed, ”Request expectation index based cache replacement algorithm for streaming content delivery over ICN”, Future Internet, MDPI, vol. 9, no. 83, pp.1-15, 2017.

[7] Z. Ming, M. Xu, and D. Wang, “Age-based cooperative caching in information-centric networks,” in Proc. of IEEE Conference on. Computer Communications and Networks (ICCCN), Shanghai, China, August 2014.

[8] H. Mirsadeghei, N.B. Mandyam, and A. Reznik, “Joint caching and pricing strategies for popular content caching in information centric networks,” Journal on Selected Areas of Communication, IEEE, vol. 35, no. 3, pp. 654-667, 2017.

[9] I. Psaras, W. K. Chai, and G. Pavlou, “Probabilistic in-network caching for information-centric networks,” in Proc. of ACM Workshop on Information Centric Networking (ICN), Helsinki, Finland, August 2012.

[10] A. V. Vasilakos, Z. Li, G. Simon, and W. You, “Information centric network: research challenges and opportunities,” Journal of Network and Computer Applications, Elsevier, vol. 52, no. 1, pp. 1-10, 2015.

[11] J. M. Wang, J. Zhang, and B. Bensaou, “Intra-AS cooperative caching for content-centric networks,” in

Proc. of ACM Workshop on Information Centric Networking (ICN) co-located with ACM SIGCOMM, Hong Kong, China, August 2013.

[12] Y. Wang, M. Xu, and Z. Feng, "Hop-based probabilistic caching for information-centric networks," in Proc. of IEEE Global Communication Conference (GLOBECOM), GA, USA, December 2013.

[13] W. Wang, Y. Sun, Y. Guo, D. Kaafar, J. Jin, J. Li and Z. Li, "CRCache: Exploiting the correlation between content popularity and network topology information for ICN caching," in Proc. of IEEE International Conference on Communications (ICC), Sydney, Australia, June 2014.

[14] H. Yan, D. Gao, W. Su, F. Chuan, H. Zhang, and A. Vasilakos, "Caching Strategy Based on hierarchical cluster for named data networking," IEEE Access, pp. 1-1. 10.1109/ACCESS.2017.2694045, 2017.

[15] X. Zhu, J. Wang, Li. Wang, W. Qi, "Popularity-based neighborhood collaborative caching for information-centric networks," in Proc. of IEEE International Performance Computing and Communications Conference (IPCCC), CA, USA, December 2017.

[16] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, L. Dong, "Popularity-driven coordinated caching in named data networking," in Proc. of ACM/IEEE Symposium on Architecture, Network and Communication Systems (ANCS), TX, USA, October 2012.

[17] J. Ren, W. Qi, C. Westphal. J. Wang, K. Lu, S. Liu and S. Wang, "MAGIC: A distributed max-gain in-network caching strategy in information-centric networks," in Proc. of IEEE International Conference on Computer Communications (INFOCOM) Workshop, Toronto, Canada, May 2014.