A Probability-based Caching Strategy with Consistent Hash in Named Data Networking

Yang Qin*, Weihong Yang, and Wu Liu
Department of Computer Science,
Harbin Institute of Technology (Shenzhen),
Shenzhen, China,
*corresponding author: csyqin@hit.edu.cn

***Abstract*—In-network caching is one of prominent features of Named Data Networking (NDN), which greatly improves the performance of data transmission. In this paper, we propose a probability-based caching strategy with consistent hash (Prob-CH). Prob-CH makes caching decision based on the probability that calculated by jointly considering content's popularity, node's betweenness, and distance to consumers. The consistent hashing algorithm is used to guarantee that there is at most one copy cached in the network, which can reduce the redundancy of cache. Moreover, Prob-CH forwards a newly arriving Interest according to a dual forwarding strategy, in which an Interest packet will be guided to producer and its cached node calculated by consistent hashing, respectively. The simulation results show that the proposed Prob-CH caching strategy can achieve better performance in terms of cache-hit ratio, hop counts and server load.**

Keywords—named data networking; caching strategy; probability-based; consistent hash

I. INTRODUCTION

The biggest feature of NDN [1] is that the cached Data in the intermediate routers can be requested by other nodes, which greatly increases the reusability of content, saves bandwidth by reducing the repeated transmission of the same Data, and improves utilization of network resources. Named data networking are primarily devoted to the study of scalability issues based on name routing, efficient content distribution strategies, content protection and security issues, and privacy trust models. The most important one is the efficient content distribution strategy. The caching strategy directly determines the performance of the network. Therefore, the research on the caching strategy can improve the content distribution efficiency and reduce the waiting time of consumers. Because the caching in NDN is very different from caching in web, Content Distribution Network (CDN), etc. Thus, the existing caching theory, model, and optimization method cannot be used directly in NDN [2]. Therefore, it is important to study this new cache network.

We proposed a probability-based caching strategy with consistent hashing called Prob-CH. The proposed probability-based caching model jointly considers betweenness of node, popularity of content and the distance between current node and consumer while making caching decision. Then, consistent hashing [3] is used to reduce the cache redundancy. The goal of Prob-CH is to reduce the cache redundancy and increase the diversity of cache contents. In NDN, Data packet only returns along the reverse path of Interest packet. Therefore, Data packet may be cached at the node on the path. There are mainly two ideas while designing caching strategy. The first idea is to cache based on node's information on the path, which does not require additional notification messages for cache coordination. It has little overhead of communication but may lead to high redundancy of cache. The second idea is to cache based on the information of neighbor node. In this case, nodes obtain cache information of neighbors by sending notification messages. This method can reduce the cache redundancy and improve the diversity of cached content by introducing additional overhead. Therefore, we combine the advantages of the above two ideas and design a novel caching strategy called Prob-CH, which has the following features:

- Making caching decision based on the probability that jointly considers the popularity of content, betweenness of node, and the distance between current node and consumer;
- Applying consistent hashing algorithm to avoid the communication overhead caused by coordination of cache, and further reducing the cache redundancy;
- Using a new forwarding strategy to explore cached content in neighbor nodes, and thereby increasing the ratio of cache hit.

The rest of the paper is organized as follows. Section II introduces background of NDN and the existing researches on caching strategies. Section III gives a description of the proposed Prob-CH. We evaluate Prob-CH via simulations in Section IV. Section V concludes the paper.

II. RELATED WORKS

In this section, we first introduce NDN's background related to this paper. Then, a brief overview of existing researches for caching strategy is given.

A. NDN Overview

NDN is a novel network architecture that focuses on the content itself rather than the location. It differs significantly from current TCP/IP-based architecture in many ways. CCN uses a hierarchical name to identify content (e.g., /hit/sz/cs/video.mp4), rather than using IP address. In NDN,

978-1-5386-4870-4/18/$31.00 ©2018 IEEE

communication is driven by the exchange of Interest and Data packet. Consumer issues an Interest packet to request a desired content. Producer responds to this Interest with a Data packet, and the Data packet travels back to consumer by taking the reversed path of Interest. In order to process Interest and Data packet, each router is equipped with three tables: Forwarding Information Base (FIB), Pending Interest Tables (PIT), and Content Store (CS). FIB serves as the routing table; PIT is used to record the unsatisfied Interest; and CS caches Data to satisfy subsequent Interest, which provides in-network caching for CCN. The processing of Interest and Data packet is shown in Fig. 1.

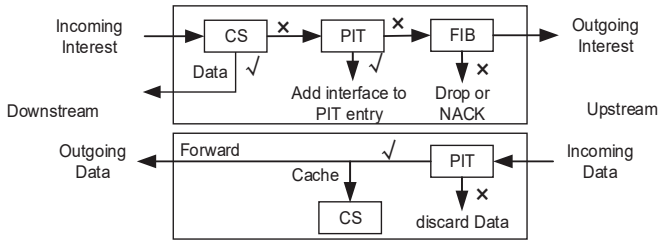

Fig. 1. The processing of Interest and Data.

While receiving an Interest packet, the CS will be checked first. If the corresponding Data packet is found, the Data will be sent back to response this Interest. Otherwise, it inquires its PIT. If an entry of this Interest exists in PIT, router records the incoming interface of Interest; if the router cannot find an entry for this Interest in PIT, it forwards the Interest according to FIB. Once a router receives a Data packet, it will cache the Data into its CS based on the caching strategy. It sends the Data to the next hop according to the interface recorded in PIT.

B. Caching Strategy Overview

In NDN, the default caching policy is to cache everywhere, namely, to cache Data at every node that Data packet passes through. This strategy may cause high cache redundancy. The collaborative strategies between nodes can be divided into two categories: implicitly collaborative strategy and explicitly collaborative strategy.

1) Implicitly collaborative caching

The implicitly collaborative strategy refers to a strategy of caching that nodes do not send notification messages; therefore, it does not require additional communication overhead, and the nodes can make caching decision at wire speed. Due to the lack of explicit collaboration between nodes, the distribution of cache copies may be uneven. The Implicitly collaborative strategies can be divided into two categories: deterministic caching and probability-based caching.

a) Deterministic caching

This type of caching strategy determines whether a packet is cached on a node in a deterministic way. It is relatively simple and easy to implement, but it is not flexible and cannot adjust the cache location when the status of network changes. LCE [4] is a simple deterministic caching strategy, which caches on all nodes through which packets pass. LCD [5], which caches on only the next-hop node of the cache hit node, can gradually move the Data closer to the edge of network with the number of requests increasing. MCD [5] caches at the next hop of cache hit node, and at the same time removes the cache on the hit node to reduce the cache redundancy. Betws [6] caches Data in the node with the greatest betweenness on the returning path to cache Data on important nodes.

b) Probability-based caching

Probability-based strategy caches Data by probability. RCOne strategy [7] select a node with probability $p = 1/k$ to cache on the returning path of Data, where k is the length of path. ProbCache strategy [8] calculates the probability of caching by weighting the distance between the node and the consumer, making the Data packets move closer to the edge of the network. The strategy proposed in [9] establishes a caching model by considering the residual capacity of node, the path length, and the neighboring nodes, which can effectively reduce server load and increase the diversity of content in the network.

2) Explicitly collaborative caching

Explicitly collaborative caching strategies need to coordinate between nodes by sending notification messages. The existing explicitly collaborative strategies are mainly divided into on-path collaboration and neighborhood-based collaboration.

a) On-path collaborative caching

Nodes on the forwarding path need to collaborate to make caching decision according to state information along the path. Cooperative En-Route web Caching (CERC) [10] collects the information about cache status of nodes and the request frequency along the path, then use the dynamic programming to determine the optimal cache node. APDR (Content-aware Placement, Discovery and Replacement) [11] makes caching decision based on residual size of cache and the number of ports that were requested, etc.

b) Neighborhood-based collaborative caching

Neighborhood-based collaborative strategy means that the nodes collaborate with each node by exchanging notification messages to its neighbors. Reference [12] proposed a neighborhood-based method that will only cache a Data packet when none of its neighbor has cached a copy of the Data. The pull with piggybacked push (PPP) strategy [13] pushes a notification message to inform the neighbor nodes proactively that what Data packets it has.

III. Probabilistic-Based Caching with Consistent Hash

In-network caching can reduce network traffic and save bandwidth; however, it may lead to a large amount of cache redundancy. Using hashing method to map content name to a unique node and caching the content in this node can limit the number of copy to at most one, which can result in low redundancy and high cache diversity.

A. Consistent Hash

The modular hashing algorithm is simple, efficient, and easy to implement; however, it may cause the problem of

rehashing when the number of nodes changes. For example, when a node is added to or removed from the network, all cache locations will change and the cached contents become unavailable. All cached Data will be deleted or moved, which results in high network delay and traffic overhead. Thus, we use Consistent Hashing (CH) [3] algorithm to solve this problem. The CH algorithm is summarized as follows.

First, based on the number of content objects in the network, we set a ring bucket for mapping nodes and content objects to the ring bucket. Assume that the size of ring bucket is *M*, as shown in Fig. 2. Then, nodes and content's names are mapped into ring buckets using different modular hashing operation, so that each node and name has a unique bucket on the ring. To cache a Data packet, the location of this Data will be found on the ring bucket by consistent hashing, then Data will be cached in the node that it first encounters in the clockwise direction on the ring. For example, content C will be cached at node 4 in Fig. 2.

When a new node is added to network, it is first mapped into the ring as shown in Fig. 2. In this case, the content between node 4 and new node will be cached in the new node according to CH. When a node is removed (e.g., node 1 in Fig. 2), content located between node 1 and 6 on the ring will be cached on node 2, namely, content A will be cached on node 2.

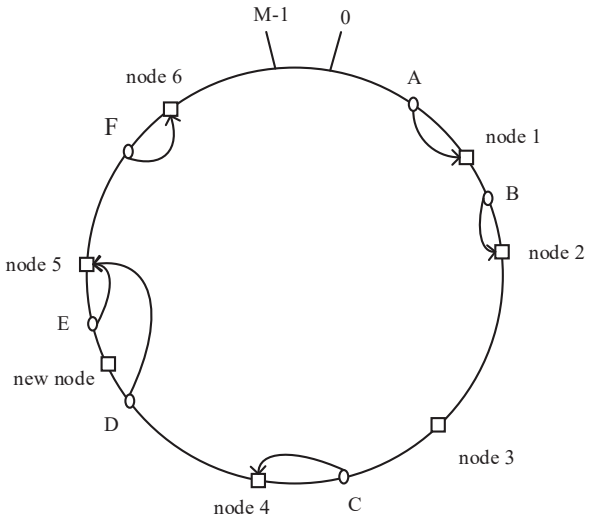

Fig. 2. Consistent hashing algorithm.

B. Prob-CH Caching Strategy

1) The format of packet

To realize Prob-CH strategy, we modify the format of Interest and Data by adding several new fields, which is shown in Fig. 3. For Interest packet, we add a hop counts field to record the hop counts of this Interest, so that the current node can know the distance between the current node and the consumer during Data packet returns. When forwarding flag field is set to *false*, the Interest is forwarded using the shortest path algorithm. When the flag is *true*, a hash-based neighborhood lookup algorithm is used for forwarding. For Data packet, the popularity and hop count fields are used to calculate the forwarding probability. The Time Update field is used to record the latest update time of the popularity.

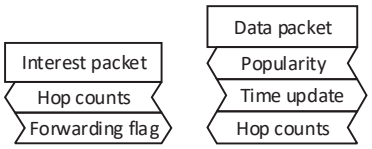

Fig. 3. The modified Interest and Data packet.

2) Design of Prob-CH

a) The forwarding of Interest

Nodes are classified into Edge Router (ER) and Core Router (CR).

When edge router receives an Interest, it first checks for matching Data in its Content Store (CS). If it exists, Data is returned on the interface from which the Interest came; otherwise, hop counts flag in Interest is increased by one and Interest will be forwarded according to the FIB. Then, the name of Interest will be mapped using consistent hashing. If the mapped node is the current node, this process ends. Otherwise, the Forwarding flag in Interest is set to *true*, and this Interest will be forwarded to the mapped node. This process is shown in Algorithm 1.

Algorithm 1

```
1:  while receiving Interest do
2:  if  find Data in CS then
3:    forward Data;
4:    return;
5:  else
6:    increase Hop count by 1;
7:    forward Interest by FIB, create new entry in PIT,
8:    calculate mapped node by consistent hashing;
9:     if current node == mapped node then
10:      return;
11:    else
12:      set forwarding flag to true;
13:      forward Interest to mapped node;
14:    end if
15:  end if
16:  end while
```

When a core router receives an Interest, it first checks the forwarding flags. If the flag is *false*, this Interest will be forwarded as a normal Interest; otherwise, the mapped node of this Interest is calculated using consistent hashing. If the current node is the mapped node, the Data packet will be returned if it is found in CS. If there is no matching Data found, the Forwarding flag is set to *false*, and the Interest is forwarded according to FIB. If the current node is not a mapped node of this Interest, Interest packet is forwarded to its mapped node via the shortest path. This process is shown in Algorithm 2.

Algorithm 2

```
1:  while receiving Interest do
2:  if  forwarding flag is false then
3:    if  has record in PIT then
```

```
4:      add incoming interface in PIT;
5:    else
6:      forward Interest by FIB;
7:      create new item for Interest in PIT;
8:    end if
9:  else
10:     calculate mapped node by consistent hashing;
11:     if current node == mapped node then
12:       if find Data in CS then
13:         forward Data;
14:       else
15:         set forwarding flag to false;
16:         forward Interest by FIB;
17:       end if
18:     else
19:       forward Interest to mapped node;
20:     end if
21:   end if
22: end while
```

b) The caching strategy

When the Data packet returns, PIT is checked first. If there is no record in PIT, this Data is discarded; otherwise, the Data packet is forwarded according to PIT. Then, the mapped node of the Data packet is calculated using consistent hashing. If the current node is not the mapped node, the Data will not be cached; otherwise, the Data packet is cached according to a probability-based caching strategy. The caching probability can be calculated as follows:

$$p = \alpha U + \beta C + \gamma D \tag{1}$$

where U is the normalized popularity, C is the normalized betweenness of node, and D is the normalized distance between the current node and consumer. α , β , γ are weights, and $\alpha + \beta + \gamma = 1$. The process of caching is shown in Algorithm 3.

Algorithm 3

```
1:  while receiving Data do
2:  if  find record in PIT then
3:    forward Data by PIT;
4:    calculate mapped node by consistent hashing;
5:    if current node == mapped node then
6:      calculate the caching probability by (1);
7:      make caching decision based on caching probability;
8:    else
9:       discard Data;
10:   end if
11:   discard Data;
12:  end if
13: end while
```

The popularity U is updated as

$$U = \omega_1 U_{old} + \omega_2 \frac{1}{\Delta t}\frac{N}{\delta} \tag{2}$$

where U_{old} is the popularity updated last time; Δt is the time that has elapsed since the last time the popularity was updated; N is the number of requests for this Data since the last time the popularity was updated; δ is the threshold of the number that this Data packet is requested. Threshold δ is used to normalize the popularity, and if $N \geq \delta$, we set $N/\delta = 1$. When Δt is longer than the predefined threshold ε , the popularity will be updated according to (2). The node's betweenness is calculated as $\sum_{x \neq y \neq i} \sigma_{x,y}(i) / \sigma_{x,y}$, where $\sigma_{x,y}$ is total number of the shortest paths from node x to node y, and $\sigma_{x,y}(i)$ is the number of those paths that pass through node i. Then, we normalize the betweenness as follows

$$C(i) = \frac{2 \times \sum_{x \neq y \neq i} \sigma_{x,y}(i) / \sigma_{x,y}}{(n-1) \times (n-2)} \tag{3}$$

where n is the total number of nodes in the network. The distance between the current node and consumer can be normalized as $D = 1/d$.

C. Analysis of Prob-CH

We use an example (as shown in Fig. 4) to analyze Prob-CH in detail. In Fig. 4, Node 1 and Node 8 are edge routers, while the rest of nodes are core routers.

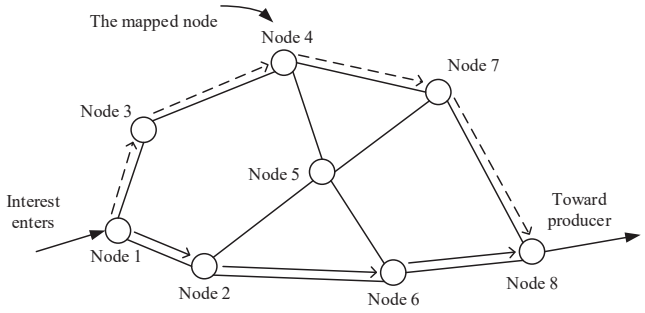

Fig. 4. An illustrated example of Prob-CH.

Prob-CH combines with the merits of probability-based and hash-based caching strategy. Prob-CH makes caching decision by probability that calculated based on content's popularity, node's betweenness, and the distance to consumer, while the problem of cache redundancy can be solved by consistent hashing. The common strategy always forwards Interest to the corresponding mapped node, and only forwards by the shortest path while cache miss occurs. Thus, hash-based strategy may cause a longer delay though it can achieve lower cache redundancy. Prob-CH uses a dual forwarding strategy to not only guarantee the shortest path while forwarding, but also solve the problem of finding the mapped node of hashing.

Prob-CH only applies dual forwarding when the Interest packet reaches the edge router for the first time. Core routers choose forwarding strategy to forward Interest according to forwarding flag. As shown in Fig. 4, the Interest first arrives at the network via node 1, and then it is forwarded by dual forwarding strategy toward both the mapped node 4 (indicated by dash line) and the producer (indicated by solid line). If the

Interest cannot find the requested Data in the mapped node 4, it needs be forwarded toward producer by using FIB.

In another case, we assume that the mapped node is node 5. When node 2 receives an Interest from node 1 for the first time, this Interest is forwarded by dual forwarding strategy. Then, if node 2 receives the Interest requesting for the same Data, it first checks the forwarding flag. When the forwarding flag is *true*, the mapping node of this Interest is calculated by the consistent hashing. Node 2 is not a mapped node; therefore, the Interest packet will be forwarded to its mapped node (i.e., node 5).

During the returning of Data packet shown in Fig. 4, node 8 forwards two copies of the Data according to PIT record. One of the copies will return to consumer via the shortest path, and the other one is forwarded to consumer passing through the mapped node. Then, the Data will be cached in the mapped node.

IV. Simulations

In this section, we use Icarus [14] simulator to compare the proposed Prob-CH strategy with probability-based strategy (denoted as Prob-BP), Hash strategy, and Always strategy. Prob-BP caches Data based on probability calculated by (1). Always strategy caches Data at every node along the path. We study the cache hit ratio, hop counts and server load by varying cache size from 1% to 10% of the sum of all content sizes. Cache size is the ratio of the number of contents that each node can cache to the total number of contents. This simulation uses the European Research and Educational Network (GEANT) network topology [15], which consists of 44 nodes and 61 links. We use the Dijkstra algorithm to calculate the shortest path. We adopt Least Recently Used (LRU) as cache replacement strategy.

A. Cache Hit Ratio

In Fig. 5, the cache hit ratio increases with the increase of cache's size. Among those four caching strategies, the Always strategy has the lowest cache hit ratio because it causes a large amount of redundancy. The Prob-BP strategy reduces the redundancy to a certain extent, but Prob-BP only considers the cache information of nodes along the forwarding path and does not consider the caches of neighbor nodes. Thus, its hit ratio is lower compared with Hash strategy and Prob-CH strategy. Both Hash strategy and Prob-CH strategy have only one cached copy in the whole network, which can greatly improve the content diversity, and finally result in higher average hit ratio. The difference between the Prob-CH and Hash in hit ratio is very small. The main reason is that the diversity of cached contents is almost the same with these two strategies; however, Prob-CH outperforms Hash strategy in that it is optimized by the probability-based caching strategy.

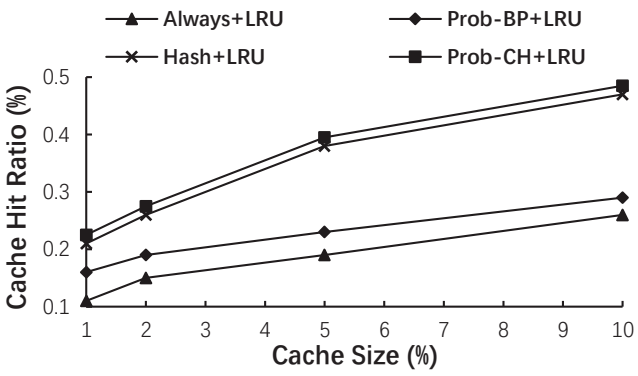

Fig. 5. Cache hit ratio with different cache size.

B. Hop Counts

In Fig. 6, as the size of cache increases, the hop count gradually decreases. Hash strategy forward the Interest to the mapping node, and only forwards it according to the FIB when fails to get Data at the mapping node. The forwarding of Interest and Data may become inefficient, which leads to larger number of hop counts. The high redundancy introduced by Always strategy lowers the cache hit ratio, and finally results in more hop counts. The trend of Prob-BP's curve in Fig. 6 is similar to that of Hash strategy because both strategies only consider the cooperation of cache along the paths, rather than among neighborhood. Prob-BP considers the factors such as betweenness; therefore, it can outperform Always strategy in terms of hop counts. By combining the advantage of Prob-BP and consistent hashing, Prob-CH greatly reduces the number of hop counts. Moreover, Prob-CH can increase cache diversity, which means more Interest packets can be satisfied while forwarding. When the size of cache becomes larger, the diversity of content will become richer, along with high probability of cache hit. Thus, the hop count of Prob-CH reduces faster than other strategies.

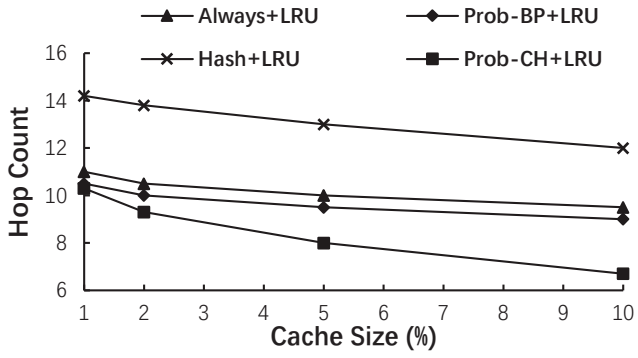

Fig. 6. Hop counts with different cache size.

C. Server Load

From Fig. 7, we can see that server's load decreases when the cache size increases. This is because a node can cache more Data packets with larger size of cache, and higher probability

of cache hit can be achieved. The number of Interest packets arriving at producer will become less; thus, the load of server can be reduced. Prob-BP and Always strategy only consider the cache along the forwarding path, and neglect the cache information of neighbors. Therefore, the Interest has to be forwarded to producer when cache miss occurs along the forwarding path. Prob-CH and Hash strategy are aware of the location where Data packets may be cached, which increases the availability of resources. Meanwhile, because of its high cache diversity, a large cache hit ratio can be obtained. From In Fig. 7, it can be seen that the load of Prob-CH and Hash strategy are significantly lower than that of Prob-BP and Always strategy. Though Prob-CH somewhat outperforms Hash strategy, performances of Prob-CH and Hash strategy are closed. This is because the types of cached content are almost the same.

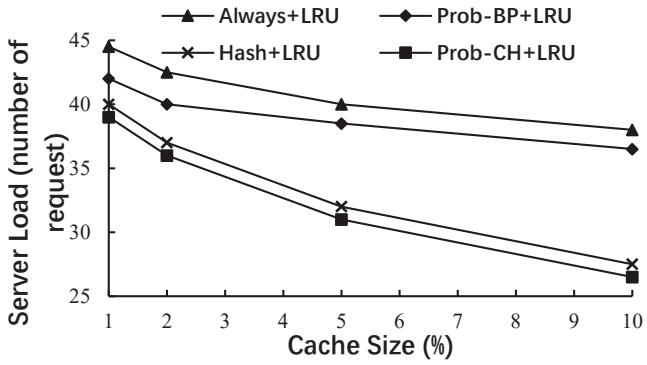

Fig. 7. Server load with different cache size.

V. CONCLUSION

In order to improve the performance of caching in NDN, this paper presents a probability-based caching strategy with consistent hashing (Prob-CH). Prob-CH adopts a probability-based caching to ensure that each packet can be cached to the appropriate node, and uses consistent hashing to reduce the cache redundancy without introducing extra communication overhead to the network. We implement Prob-CH in Icarus simulator and evaluate its performance by comparing with other caching strategy. The results show that Prob-CH outperforms others in terms of cache hit ratio, hop counts, and producer load.

ACKNOWLEDGMENT

This work was supported by the Science and Technology Fundament Research Fund of Shenzhen under grant JCYJ20160318095218091, JCYJ20170307151807788.

REFERENCES

[1] L. Zhang *et al.*, "Named Data Networking (NDN) Project," *October*, pp. 1–26, 2010.

[2] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Comput. Networks*, vol. 57, no. 16, pp. 3128–3141, 2013.

[3] D. Karger *et al.*, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," *STOC '97 Proc. twenty-ninth Annu. ACM Symp. Theory Comput.*, pp. 654–663, 1997.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, R. L. Braynard, and N. H. Briggs, "Networking Named Content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12.

[5] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Perform. Eval.*, vol. 63, no. 7, pp. 609–634, 2006.

[6] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks (extended version)," *Comput. Commun.*, vol. 36, no. 7, pp. 758–770, 2013.

[7] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga, "CATT: potential based routing with content caching for ICN," *Proc. Second Ed. ICN Work. Information-centric Netw. - ICN '12*, p. 49, 2012.

[8] 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 - ICN '12*, 2012, p. 55.

[9] R. Huo, J. Liu, T. Hhuang, J. Chen, and Y. Liu, "Cooperative Caching Strategy Based on Correlation Probability in Information Centric Networking," *J. BEIJING Univ. POSTS Telecommun.*, vol. 38, no. 1, pp. 16–20, 2015.

[10] X. Tang and S. T. Chanson, "Coordinated en-route web caching," *IEEE Trans. Comput.*, vol. 51, no. 6, pp. 595–607, 2002.

[11] W. Liu, S. Yu, J. Cai, and Y. Gao, "Scheme for Cooperative Caching in ICN," *J. Softw.*, vol. 24, no. 8, pp. 1947–1962, 2014.

[12] E. J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," in *Proceedings - IEEE INFOCOM*, 2009, pp. 2631–2635.

[13] K. T. Yang and G. M. Chiu, "A hybrid pull-based with piggybacked push protocol for cache sharing," *Comput. J.*, vol. 54, no. 12, pp. 2017–2032, 2011.

[14] L. Saino, I. Psaras, and G. Pavlou, "Icarus: a Caching Simulator for Information Centric Networking (ICN)," *Proc. Seventh Int. Conf. Simul. Tools Tech.*, pp. 66–75, 2014.

[15] Geant. GEANT_Project_Topology. Accessed: Jun. 2018. [Online]. Available:https://geant3plus.archive.geant.net/Resources/Media_Library/Documents/GEANT_Project_TopologyMAR14_Web.pdf.