NAME-CENTRIC PREFETCHING IN NAMED-DATA NETWORKING

A Thesis

by

MOHD FAISAL KHAN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2020

Major Subject: Computer Engineering

ABSTRACT

This thesis presents Name-Centric Prefetching (NCP) that prefetches data before users request for them to improve user experience. NCP seeks to identify user request patterns solely based on names in prior interests, and without any other knowledge about user applications. As such, NCP can be easily implemented without any modifications for applications. A prototype of NCP has been built within Named-Data Networking (NDN). The implementation includes multiple modules that make it easy to implement and test new prefetching algorithms and to manage the computation, storage, and bandwidth overheads. The utility of NCP is evaluated under two scenarios, one derived from a real-world trace from a Google cluster and the other constructed by mimicking the behaviors of a variety of applications, and three different prefetching algorithms. Testbed emulation results demonstrate that NCP is able to significantly reduce end-to-end latency experienced by users while incurring little additional network traffic.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor I-Hong Hou, Professor Srinivas Shakkottai, Professor Zixiang Xiong of the Department of Electrical and Computer Engineering and Professor Riccardo Bettati of the Department of Computer Science.

The analysis depicted in Chapter V, VI were conducted in part by Siqi Fan of the Department of Electrical and Computer Engineering.

All other work conducted for the thesis was completed by the student independently.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction

Traditional TCP/IP network architecture adopts a host-centric approach that decouples network connections and the applications. As a result, the network is unaware of the information flowing through it. In contrast, Information-Centric Network (ICN) seeks to break the boundary between network and information. In Named-Data Networking (NDN) [1], a leading proposal for ICN architecture, every piece of information is treated as a named object, and the network forwards and requests for packets based on their associated names, rather than the end-hosts. Thus, in NDN, named contents become the first citizen. This architecture has inspired many studies on name-based protocols for routing and caching [2, 3, 4, 5, 6, 7, 8, 9].

Prefetching is the network operation that fetches data before the applications request for them. When prefetching is successful, it can significantly reduce the end-to-end latency experienced by the applications, and hence improve quality of experience (QoE). Prefetching inevitable requires the prediction for future application requests. Most current studies on prefetching either require that applications request data in a deterministic manners, such as variants to the Pub-Sub and Sync protocols [10, 11, 12, 13], or assume a prediction oracle that knows the probability distribution of future requests [14, 15, 16, 17]. Thus, these studies are application-specific, and it may be difficult to generalize them for applications whose request patterns are not well-defined or difficult to learn.

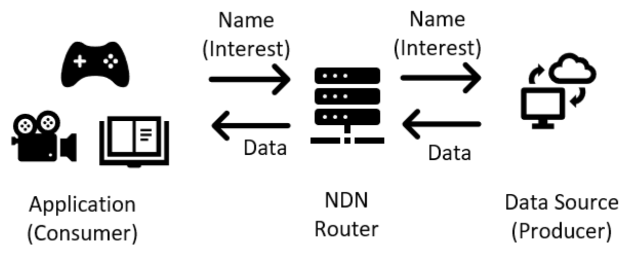
In this thesis, we explore name-based protocols for prefetching. We observe that many applications request data in correlated, though not necessarily deterministic, patterns. Such patterns can be observed by the names in interests that the applications generate. Motivated by this observation, we propose a Name-Centric Prefetching (NCP) protocol. The core of NCP is a name-centric prediction rule. This rule seeks to identify patterns solely based on previously received interests, and use these patterns to predict future ones. It can be implemented easily without any knowledge of the applications. In addition, NCP is fully compatible with NDN, and it includes multiple modules that ensure it can be easily extended to support new prefetching algorithms with limited overheads on computation, storage, and bandwidth usage.

To demonstrate the usefulness of NCP, we implement a NDN prototype with NCP support. In the absence of traces of comprehensive NDN applications, we construct two scenarios, one is derived from an actual trace from a Google cluster, and the other is synthesized by considering the behaviors of a variety of applications. We also implement and evaluate three different prefetching algorithms. Testbed emulation results demonstrate that, in both scenarios, NCP can significantly reduce end-to-end latency by prefetching the correct data in advance. Moreover, NCP incurs little bandwidth overhead, as its false alarm rate is very low.

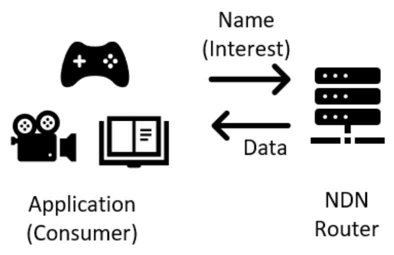
The rest of the thesis is organized as follows: Section-2 discusses the patterns exhibited by a variety of applications, and use them to establish the name-centric prediction rule. Section-3 introduces several fundamental concepts that are central to the design and implementation of NCP. Section-4 discusses the software architecture of NCP, its implementation, and several algorithms of NCP. Section-5,6 describes the emulation scenarios, and presents the emulation results. Section-7 summarizes some related work. Finally, Section-8 concludes the thesis.

2. NAME CENTRIC INTEREST PREDICTION

The core of NCP is a name-centric interest prediction rule that uses the names of recently received interests to predict future ones. In this section, we first provide a brief overview of NDN. We will provide several case studies of interest patterns by various applications that motivate our design. Finally, we introduce our name-centric interest prediction rule.



(a) The router does not have the requested data.



(b) The Router has the requested data.

Figure 2.1: The procedure of data retrieval in NDN.

2.1 Basics of Named-Data Networking

The basic idea of NDN is to communicate via named packets. In NDN, each piece of data is assigned a semantically meaningful name. This name identifies the data itself and is independent of the data container, its location, or the underlying network connectivity. To communicate, data applications (consumers) express requests, called *interests*, which carry the names of desired data generated by data source (producers). Each piece of data has a unique name.

A NDN router has the capability to store recently received data, along with their names, in its own Content Store (CS). When a consumer requests for a piece of data, it first sends an interest to the NDN router. If the NDN router has the data in its CS, it can reply with the data immediately. Otherwise, the NDN router forwards the interest to the backbone network, and waits for the data to arrive. When the requested data arrives, it forwards the data to the client, and may store the data in its own Content Store.

Fig. 2.1 illustrates the procedure to obtain data in NDN, both when the data is cached by the router and when it is not. Obviously, the latency is much smaller when the NDN router already has the requested data. Thus, if the router is capable of predicting and prefetching future interests in advance, then the consumers will experience much less latency, which, in turn, can significantly improve user experience.

2.2 Case Studies of Interest Patterns

NDN uses hierarchical naming to identify the content. In many applications, the names of the interests, though being unique, follow some sequential patterns. We provide several examples that show these patterns in their interest names.

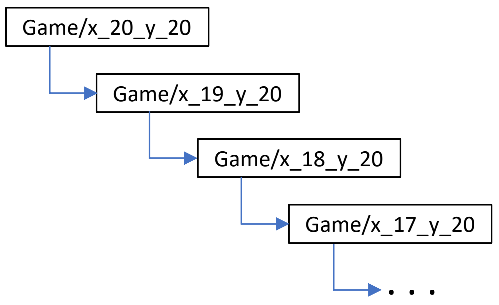
In Virtual Reality (VR) gaming, the game requests the panoramic frame according to the location of the player in the virtual world. The name of the interest contains information about the player's current location. An example of interest of VR gaming type could be

`<Name of Game>/x_<pos.x>_y_<pos.y>`,

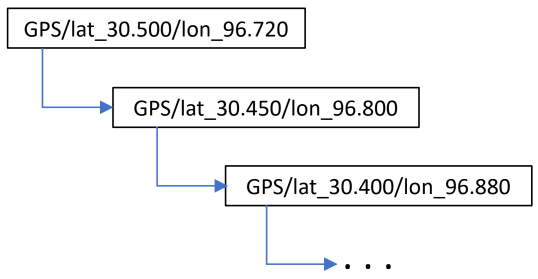
where (`pos.x, pos.y`) is the players location in the virtual world. Now, consider that the player moves in a certain direction, say, east. Then, every time the player moves and requests for a new panoramic image, `pos.x` reduces by one and `pos.y` remains unchanged. Fig. 2.2(a) shows an example of the sequence of interests from the player. After observing the four interests shown in Fig. 2.2(a), one can reasonably predict that the next interest from the player is likely to be `Game/x_16_y_20`. Another example is GPS navigation, where the application requests map data for its surroundings periodically. Each interest contains the current location of the user, which, as is the case of URL naming convention of Google Map, is typically specified by the latitude and longitude values of the user. An example of an interest name for such an application is

`<Map name>/lat_<pos.lat>/lon_<pos.lon>`,

where `pos.lat` and `pos.lon` are the latitude and longitude of the user's current location. Consider a driver that uses cruise control on a highway, then the amount of change in (`pos.lat, pos.lon`) between consecutive interests remains the same. Fig. 2.2(b) shows one such example.

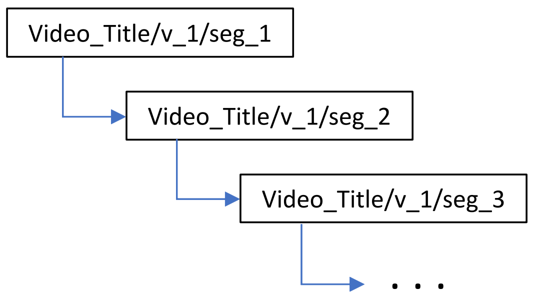


(a) VR game interest pattern

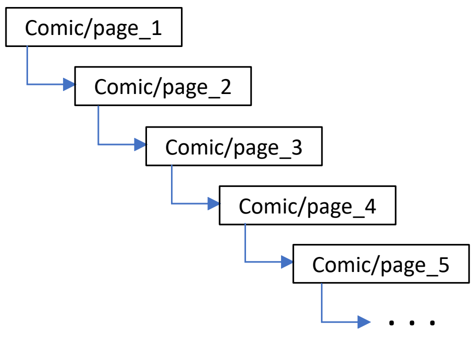


GPS/lat_30.500/lon_96.720
GPS/lat_30.450/lon_96.800
GPS/lat_30.400/lon_96.880
. . .

(b) GPS interest naming pattern



(c) On-demand video streaming interest pattern



(d) Comic book interest pattern

Figure 2.2: The interest patterns of four different applications.

Based on the three interests shown, it appears that the next interest is likely to be
`GPS/lat_30.350/lon_96.960`.

We can only consider on-demand video streaming services. Suppose the streaming services uses Dynamic Adaptive Streaming over HTTP (DASH) protocol. DASH works by breaking video contents into small segments and transferring the segments to the application. Each segment is stored as a separate file, and DASH uses the GET command in HTTP to fetch segments. These segments are then combined and played by the application. A typical naming convention of DASH is
`<Video_Title>/v_<ver_num>/seg_<seg_num>`.
The segments are numbered sequentially. As the video progresses, the segment number increases, resulting a clearly identifiable pattern as shown in Fig. 2.2(c).

Finally, we can consider the application of online comic books. The content can be identified by the name of the page. A user reads a comic book page by page. Each page is a named data with name, `<Book Title>/page_<page number>`. The reader sends a new interest whenever it turns a page. Based on the series of interest shown in Fig. 2.2(d), one can predict that the next requested page is `Comic/page_6`.

From the case study of these four vastly different applications, we observe that many applications exhibit clear patterns in their interests. We therefore seek to identify and exploit such patterns to design a rule for predicting future interests.

2.3 Prediction Rule

We now propose a name-centric prediction rule based on the observations in the previous section. We decompose each interest into an array of string components (`str_comp`) and an array of numerical components (`num_comp`). For example, the first interest in Fig. 2.2(a) can be decomposed into `str_comp = ["Game/x_", "_y_"]` and `num_comp = [20, 20]`. After the decomposition, we note that all four interests in Fig. 2.2(a) have the same `str_comp`. Moreover, the difference of `num_comp` between consecutive interests is always [-1, 0]. We thus propose a name-centric prediction rule as shown below:

Name-Centric Prediction Rule:

IF there exists three received interests, A, B and C, such that

1. A.str_comp = B.str_comp = C.str_comp,

AND

2. C.num_comp - B.num_comp = B.num_comp - A.num_comp =: Δ,

THEN predict interest D with

D.str_comp = C.str_comp

AND

D.num_comp = C.num_comp + Δ

In Fig. 2.3, we show that the first three interests in Fig. 2.2(a) can be used to correctly predict the fourth interest by the proposed name-centric prediction rule. It is straightforward to verify that this rule also works for the other three discussed applications.

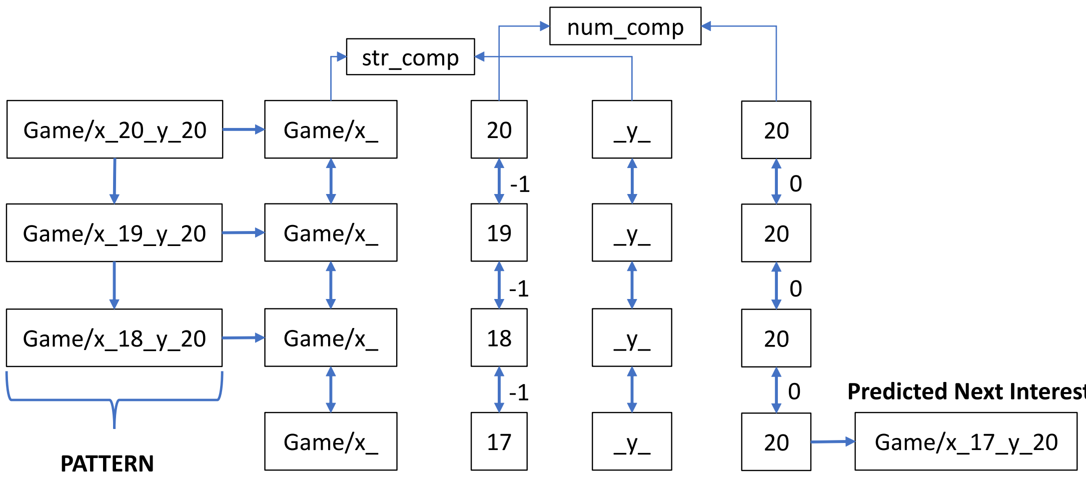


Figure 2.3: Pattern Identification

The proposed name-centric prediction rule has several important features that distinguish it from other existing work on prediction and prefetching:

- *Application-oblivious:* The name-centric prediction rule does not need any knowledge about the application, nor does it require any modifications from application developers. The prediction rule passively observes incoming interests to make predictions.

- *Low complexity:* The name-centric prediction rule is very simple and can be easily carried out with low complexity. Suppose a router stores N prior interests. Upon the arrival of a new interest, the router needs to check whether there are two prior interests that, along with the arriving new interest, satisfy the name-centric prediction rule. A brute-force search only requires $O(N^2)$ complexity. The complexity can be further reduced with the help of hash table.

- *Compatible with fast-changing patterns:* The name-centric prediction rule only requires three prior interests to make a prediction. This feature is important for many interactive applications whose interest patterns can change frequently. For example, the interest pattern of VR gaming changes whenever the player makes a turn, and the interest pattern of GPS navigation changes whenever the driver changes its speed.

3. NCP PRELIMINARIES

Using the name-centric prediction rule, NCP seeks to identify opportunities to prefetch data. As a router needs to serve multiple consumers with very different behaviors simultaneously, there are multiple practical challenges that need to be addressed to develop a fully NDN-compatible protocol that can potentially support a variety of prefetching algorithms with low overheads in terms of computation, memory, and bandwidth. In this section, we introduce some fundamental concepts that are useful for the design and implementation of NCP.

3.1 Pattern

Formally, when there is a sequence of interests where consecutive interests satisfy the name-centric prediction rule, then we say that this sequence of interests form a *pattern*. The *length* of a pattern is the number of received interests in this sequence. For example, the interests shown in Fig. 2.2(a) form a pattern of length four. Intuitively, a long pattern suggests that the consumer indeed generates interests according to the identified pattern, and thus the prediction is likely to be correct.

We also note that different applications generate interests at different rate. The knowledge about interest rate can be useful for the prefetching algorithm to estimate when the predicted interest is supposed to arrive. To capture this information, we assign a sequence number to each interest arrived at the router. We define the *gap* of a pattern as the difference in the sequence numbers of the last two interests in the pattern. A pattern with large gap generates interests at a low rate.

3.2 Interest Capture Table

In order to perform the name-centric prediction rule, the router needs to store some previously received interests. We create an Interest Capture Table (ICT) to store recent interests. Similar to the Pending Interest Table (PIT) in NDN, ICT only stores the names of interests, but not their corresponding data. Thus, the size of an entry in ICT is small. On the other hand, unlike PIT, an entry in ICT may not be deleted even when the corresponding data packet arrives. To avoid

Table 3.1: Future Interest Table (FIT)

Prediction	Str_comp	Num_comp	Delta
Game/x_16_y_20	"Game/x", "_y_"	16, 20	-1,0
GPS/lat_30.350/lon_96.960	"GPS/lat_", ".", "/lon_", "."	30, 350, 96, 960	0, -50, 0, 80
Video_Title/v_1/seg_4	"Video_Title/v_", "/seg_"	1, 4	0,1
Comic/page_6	"Comic/page_"	6	1

excessive usage of memory, the router may impose a limit on the size of ICT.

3.3 Future Interest Table

A Future Interest Table (FIT) is the table that keeps track of all patterns and their predictions. Each FIT entry contains the predicted interest of the pattern, the difference in the `num_comp`, the gap and the length of the pattern, the sequence number of the last received interest, and whether the predicted interest has already been prefetched. The router may impose a limit on the size of FIT.

Suppose a router receives all the interests shown in Fig. 2.2, then its FIT would be similar to Table 3.1. The entries for Gap, length, sequence number and prediction status are not shown but form a part of FIT for each entry. Using the `num_comp` and the delta, we can generate the next predicted interest. A prefetching algorithm then determines which predicted interest in FIT should be prefetched.

4. IMPLEMENTATION AND ALGORITHMS

NDN protocol is implemented using Named Data Networking Forwarder Daemon (NFD). To demonstrate that NCP is fully NDN-compatible, we have implemented NCP within NFD version 0.6.3[18], [19]. For implementing the application, we are using the ndn-cxx C++ library [20], [21]. In this section, we provide a detailed discussion on the design and implementation of NCP.

4.1 Software Architecture

Fig. 4.1 shows the overall software architecture of our NCP implementation. The left part of the figure is a flow chart of the original NFD. In the original NFD, when an interest i arrives, NFD first updates its Pending Interest Table (PIT). Then, NFD checks whether the requested data already exists in its own Content Store (CS). If the requested data already exists, NFD can reply with the data directly. Otherwise, NFD uses its Forwarding Information Base (FIB) to determine which router to forward i to.

In our implementation, after updating PIT, NFD sends a copy of the interest to NCP through a function call. NCP then determines whether, and what, to prefetch. If NCP decides to prefetch a piece of data, it creates a predicted interest i_p, and then uses FIB to forward i_p. When the prefetched data arrives, NFD stores the data in its CS. We do not create a separate memory for storing prefetched data, nor do we modify how NFD manages its CS. This makes the implementation simpler and reduces NCP's memory usage. The right part of Fig. 4.1 shows the flow chart of NCP, which involves four different modules. We discuss the purpose and implementation of these four modules below. For each module, we also introduce algorithms for each module. Since the purpose of this work is to demonstrate the design and usefulness of NCP, we do not seek to optimize these algorithms. It should be noted that the performance of NCP may be further improved with more intelligent algorithms.

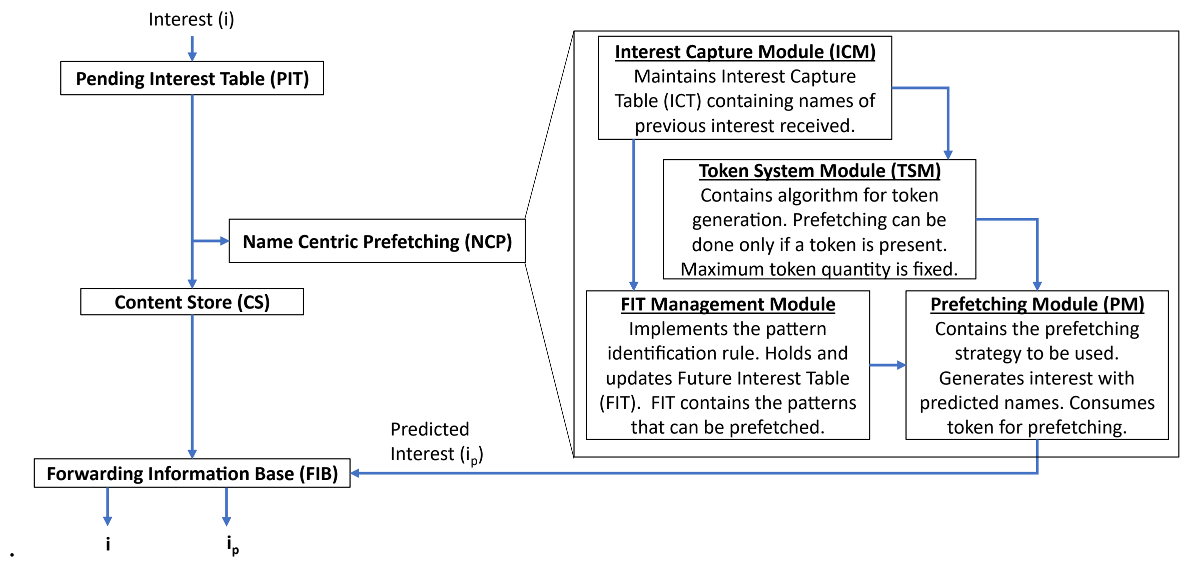

Interest (i)
Pending Interest Table (PIT)
Name Centric Prefetching (NCP)
Content Store (CS)
Forwarding Information Base (FIB)
i
i_p
Predicted Interest (i_p)
Interest Capture Module (ICM)
Maintains Interest Capture Table (ICT) containing names of previous interest received.
Token System Module (TSM)
Contains algorithm for token generation. Prefetching can be done only if a token is present. Maximum token quantity is fixed.
FIT Management Module
Implements the pattern identification rule. Holds and updates Future Interest Table (FIT). FIT contains the patterns that can be prefetched.
Prefetching Module (PM)
Contains the prefetching strategy to be used. Generates interest with predicted names. Consumes token for prefetching.

Figure 4.1: NCP Implementation on top of NFD

4.2 Interest Capture Module (ICM)

The purpose of the Interest Capture Module (ICM) is to manage the Interest Capture Table (ICT). When an interest arrives, ICM inserts its name to ICT. If the size of ICT exceeds its limit, we use a simple first-in-first-out algorithm that removes the oldest entry.

4.3 Token System Module (TSM)

An important constraint of NCP is the need to manage its bandwidth overhead. In the extreme case, one can consider an implementation where NCP prefetches a new interest whenever the name-centric prediction rule predicts one. Such an implementation is likely to result in an excessive amount of bandwidth usage, and many prefetched data may never be requested by consumers.

The purpose of the Token System Module (TSM) is to manage the bandwidth usage of NCP. TSM manages the number of *tokens*. NCP can only prefetch when there is an available token, and each prefetch uses one token.

Our algorithm for TSM is motivated by the leaky bucket model in network calculus. Specifically, TSM generates one new token for every M incoming interests. Moreover, the number of available tokens is upper-limited by B. This guarantees that the traffic generated by NCP is no more than $\frac{1}{M}$ of the traffic generated directly by consumers. The upper-limit B further limits the burstiness of prefetching traffic. The values of M and B can be chosen by the router.

4.4 FIT Management Module

This module manages the Future Interest Table (FIT). When a new interest arrives, this module first checks whether the name of the interest matches the Prediction field in any entry. A match indicates that the prediction by the corresponding pattern is correct, and the pattern should be updated. For example, suppose the current FIT is shown in Table 3.1, and an interest with name `Game/x_16_y_20` arrives. This interest is a match for the first entry in the FIT. The entry should then be updated to have Prediction = `Game/x_15_y_20` and Length = 5. Other fields of this entry should also be updated as appropriate.

On the other hand, if no match is found in the FIT, this module will search in the ICT and see if it can find two prior interests that, along with the new interest, satisfy the name-centric prediction rule. A new pattern can then be formed and added to the FIT.

When the size of the FIT exceeds its limit, this module needs to remove one entry from the FIT. We use a simple algorithm that is similar to the Least-Recently-Used (LRU) algorithm. The algorithm removes the entry with the smallest Seq field, that is, the entry that has not seen a matched interest for the longest time.

4.5 Prefetching Module (PM)

The Prefetching Module (PM) is the place to implement prefetching algorithms. When there is an available token, the PM may select an entry in the PIT and prefetch the corresponding name. When an entry is chosen, the Prefetch field is updated to TRUE, and the number of tokens reduces by 1. PM then creates a predicted interest i_p and uses the FIB to forward i_p. To evaluate performance, this module also generates a log of all prefetches.

In our implementation, we consider three simple prefetching algorithms as listed below:

- *Prefetch Current*: When an incoming interest causes an update in the FIT, this algorithm will prefetch the updated entry as long as there is an available token.

- *Prefetch Random*: When there is an available token, this algorithm randomly selects an entry in the FIT with Prefetch = FALSE and prefetches this entry.

- *Prefetch Longest Unexpired Pattern*: When there is an available token, this algorithm finds all entries in the FIT with Prefetch = FALSE and Seq + Gap is larger than the sequence number of the incoming interest. Among these entries, this algorithm selects the entry with the largest Length and prefetches it. The intuition of this algorithm is two-fold: First, Seq + Gap is the estimated time that the next interest from this pattern should arrive. Thus, if the sequence number of the incoming interest is larger than Seq + Gap, then we consider this pattern to be expired and should not prefetch for it. Second, among all unexpired patterns, we choose the longest one because we consider the prediction to be more reliable when the length is larger.

5. EMULATION

5.1 Testbed Description

We have implemented NCP in our NDN testbed, which consists of multiple machines running NFD, and have conducted some preliminary experiments. However, we later lost access to our testbed when our lab was closed due to COVID-19. Instead, we replicate the system in our personal computer. In order to create a networked environment within just one machine, we set up two virtual machines in our personal computer. One virtual machine contains a NFD with NCP implementation, and a consumer program that sends interest packets to the NFD. The other virtual machine contains a producer program running on top of a NFD without NCP implementation. The two virtual machines are connected through a UDP socket using VM VirtualBox Router. Fig. 5.1 illustrates the emulation environment.

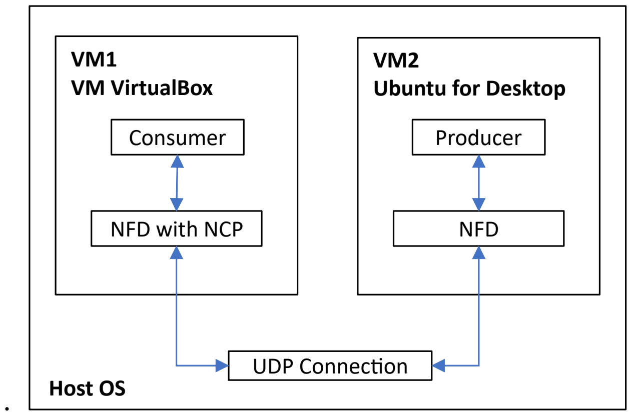


Figure 5.1: Emulation Environment

5.2 Interests Generation

Obviously, the performance of NCP is significantly impacted by the actual arrival patterns of interests. In order to evaluate NCP in a realistic setting, we construct two scenarios. The first scenario is derived from an actual trace of request arrivals at a Google cluster. The second scenario is created based on observations of several different applications.

5.2.1 Scenario of Google Trace

We use a real Google Cluster Trace data set from a Google research blog [22]. This data set records a huge number of requests for different services in a Google cluster. The record includes more than three million requests for 9,218 unique services in a seven-hour span. In this section, we run our simulation for first 30,000 requests.

In the trace file, each request is identified by a `ParentID` that indicates the service for the request. We note that just because several requests are for the same service does not mean that these requests form a pattern. To capture this behavior, we create a name for each request in the format of `<ParentID>_<Req_Seq>`. The value of `Req_Seq` is chosen as follows: When a request arrives, we look for the previous request for the same service. The `Req_Seq` of this new request is the `Req_Seq` of the previous request plus $\lceil$the number of other requests between them/20$\rceil$.

Under our approach, a sequence of requests for the same service will form a pattern if they are evenly spaced. Intuitively, a consumer that generates interests periodically is much more likely to exhibit a pattern than one that generates interests at varying rate. Our approach therefore captures this intuition.

5.2.2 Scenario with Multiple Applications

This scenario considers a system with multiple heterogeneous applications. We discuss our model for these applications below:

- *GPS Navigation*: This application considers a vehicle requesting map data for its surroundings. The application generates one interest packet every 100 milliseconds, and the name in each interest packet contains a number indicating the vehicle's current location. When a vehicle maintains its speed, the difference of locations between interest packets is the same. A vehicle may change its speed based on some random process. When a vehicle is using cruise control, we assume that, on average, it will change its speed once every 100 seconds.

When a vehicle is not using cruise control, we assume that, on average, it will change its speed once every second.

- *VR Gaming*: This application considers a player of a VR game that requests panoramic images periodically based on its current location in the VR world. The application generates one interest packet every 25 milliseconds, which corresponds to a frame rate of 40 frame-per-second. The name in each interest packet contains a number indicating the player's location. The difference of locations between interest packets remain the same when the player moves in the same direction. We consider two types of VR games, intensive games and leisure games. On average, players in intensive games change their movements once every 250 milliseconds, and players in leisure games change their movements once every 5 seconds.

- *Video Streaming*: At 40 frame-per-second, a video streaming application generates one interest packet every 25 milliseconds. The name in each interest packet contains a segment number. The segment number increases by 1 in every new interest packet.

- *Random*: This simulates applications that do not exhibit clear patterns. The name in each interest packet contains the name of the applications, the name of user, and a sequence number. In every new interest packet, the sequence number increases by a random integer uniformly in $[1, 10]$. Thus, it is possible for interest packets from this application to form a pattern according to our prediction rule. However, the prediction is likely to be wrong. These applications generate one interest packet every 10 milliseconds.

We construct a scenario with five vehicles using cruise control, five vehicles not using cruise control, two intensive VR players, two leisure VR players, one video stream, and two random applications. In total, these consumers generate 500 interests per second, out of them 200 interests per second are generated by the two random applications. We create a one-minute-long trace of interest packets based on this setting.

6. EMULATION ENVIRONMENTS AND PERFORMANCE METRICS

Table 6.1 summarizes the parameters we use in emulations. For each scenario, we evaluate all three prefetching algorithms discussed in Section 4.5. The Token System Module generates one token for every M incoming interest. The value of M varies from 1 to 10.

Table 6.1: Emulation Environment

Configuration Parameter	Value
Number of Interest	30000
CS Size	200
ICT Size	300
FIT size	20
Max Token at a time	10

When a consumer generates an interest, and the interest is already available in the Content Store due to an earlier prefetch, then we say that the corresponding prefetch is a *hit*. If a prefetch does not result in a hit, then we say that the prefetch is a *false alarm*. A prefetch can be a false alarm either because the consumer never generates the predicted interest, or because NFD deletes the prefetched data before the consumer generates the predicted interest.

For each emulation, we evaluate the following three performance metrics:

$$HitPercentage = \frac{Number\_of\_Hits}{Total\_Number\_of\_Prefetches} * 100$$

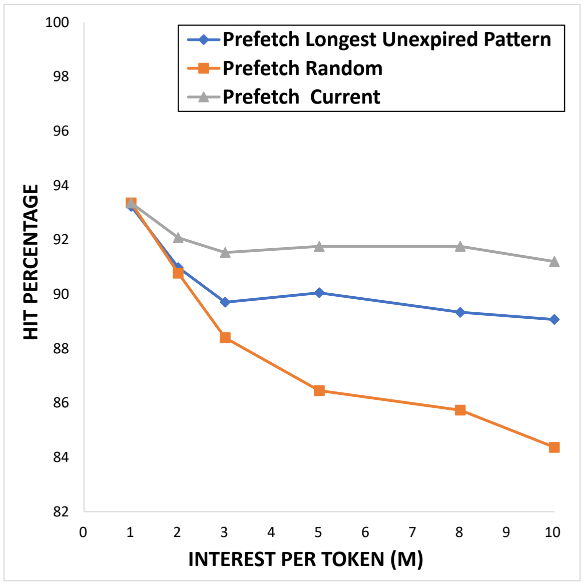
$$HitRate = \frac{Number\_of\_Hits}{Total\_Number\_of\_Interests}$$

$$FalseAlarmRate = \frac{Number\_of\_False\_Alarms}{Total\_Number\_of\_Interests}$$

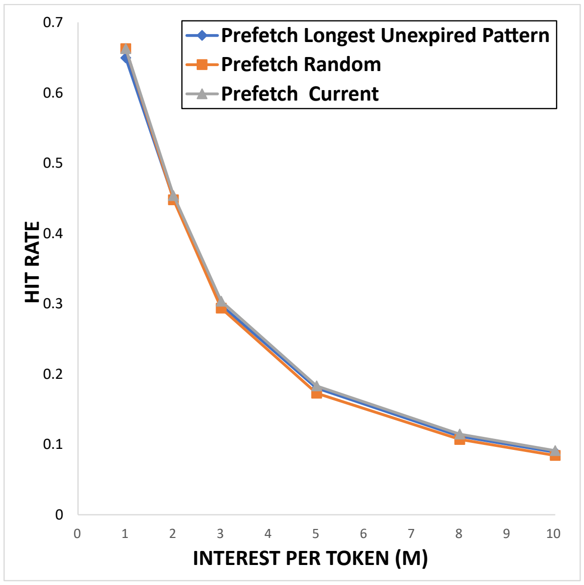
When there is a hit, the consumer can obtain its data with a much smaller latency. Thus, $HitRate$ represents the portion of interests that NCP is able to serve with low latency. In addition to latency, we also evaluate the actual bandwidth usage by NCP. We note that a hit does not result in any additional bandwidth usage. Without NCP, the piece of data is still going to be fetched. NCP only fetches the data earlier. In contrast, a false alarm represents a transmission that is not actually needed. On average, NCP forwards $(1 + FalseAlarmRate)$ interests for every incoming interest. In comparison, NFD without NCP will forward one interest for every incoming interest.

6.1 Emulation Results

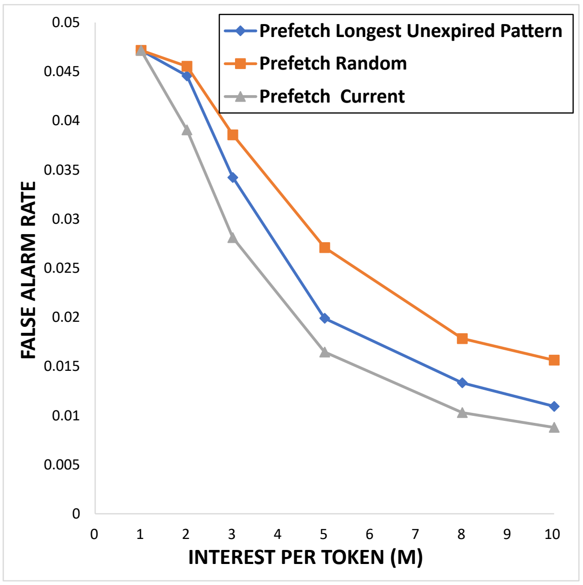
The emulation results for the two scenarios are shown in Figs. 6.1 and 6.2. When $M = 1$, the Token System Module generates one token for every incoming interest. Thus, NCP always has tokens for prefetch, and all three prefetching algorithms will prefetch every time the name-centric prediction rule produces a prediction. We can then learn some basic properties about the two scenarios by studying the case when $M = 1$. As can be seen in the figures, when $M = 1$, the $HitPercentage$ is larger than 90$\%$ for the scenario of Google trace, and is about 60$\%$ for the scenario with multiple applications. This is because the second scenario includes two random applications that generate a lot of false alarms. In this sense, the second scenario represents a more challenging scenario for NCP. We also note that the first scenario is derived from a real-world trace file. The fact that it has a high $HitPercentage$ suggests that the name-centric prediction rule may be very accurate in real world.



(a) Hit Percentage

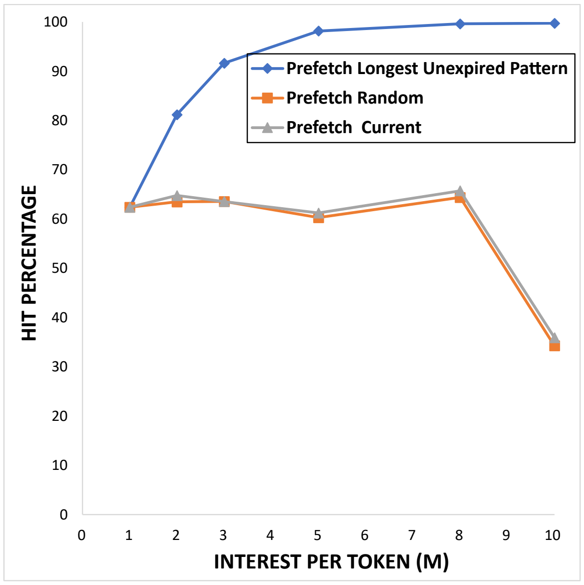


(b) Hit Rate

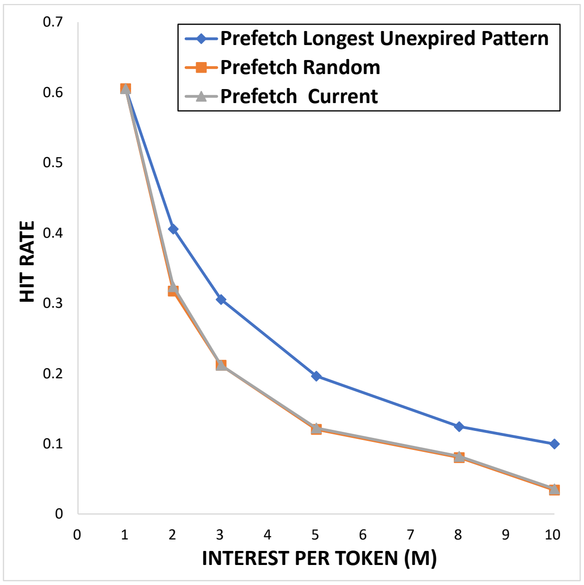


(c) False Alarm Rate

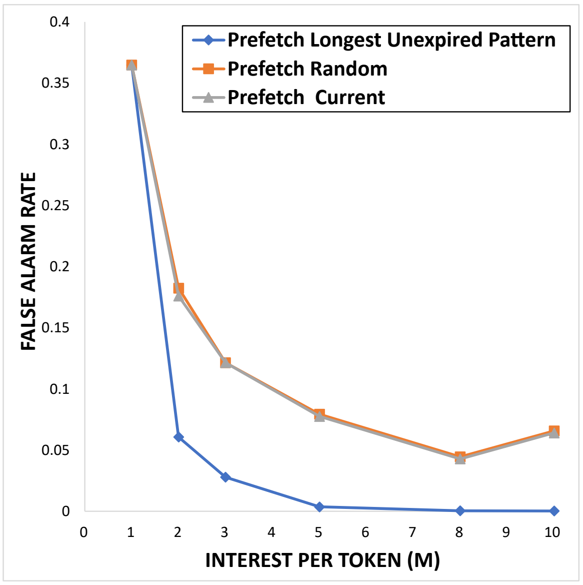
Figure 6.1: Emulation results for the Google trace scenario.



(a) Hit Percentage



(b) Hit Rate



(c) False Alarm Rate

Figure 6.2: Emulation results for the scenario with multiple applications

From Figs. 6.1 and 6.2, we also observe that the Prefetch Longest Unexpired Pattern algorithm performs very well. When M is at least 3, the $HitPercentage$ of this algorithm is always higher than 90%. Its $FalseAlarmRate$ is always lower than 0.05, showing that it causes little bandwidth overhead. Moreover, its $HitRate$ is close to $\frac{1}{M}$, suggesting that it is able to considerably reduce latency.

7. RELATED WORK

In all Information Centric Networking (ICN) architecture design [23],[24],[25],[1], the network identifies content items using unique names in contrast to hosts and endpoints with IP address. This opened up boundaries for in-network caching of named content. The availability of data content locally at the Access Point (AP) greatly improve the performance (latency, throughput) [26], [27], [28], [29]. An optimal cache policy can help in achieving this improvement. Psaras et al. [30] uses a probabilistic approach for caching object at each node present in the path. Age-Based Cooperation (ABC) [31] uses age as a component to evict cache entries. Similarly freshness is used by Jost'e Quevedo et al. [32] to design a cache policy. WAVE [33] suggest routers to dynamically adjust the cache window based on the content popularity. Progressive Caching [34] uses content popularity to cache it closer to the edge node. Popularity based caching has been extensively researched in literature [35], [36], [37].

Caching and prefetching are intertwined with each other. Prefetch operation can reduce the latency by making future data available in the cache even before the user requests it. Mobility scenario utilises prefetching extensively. Making data available in the cache in the next AP where a user is going to move-to, can extensively improve the performance. Edge Buffer [38] predicts client mobility and performs prefetching on the router to which the user is approaching. It also uses popularity based caching policy. Markov model is also used to predict mobility [39], [40]. ICN-FOG [41] uses layering approach to prefetch the data on the designated router.

Prefetching if done for video application enhances the QoE of the user. Video files are transferred as chunks, making the future chunks available on the edge router can reduce latency experienced by the end user. DASH [42] protocol is the popular method used for video streaming. The naming scheme in DASH can be used to predict future interest and perform prefetching [43], [44]. ICN-PEP [45], [46] uses segments present as part of file name to perform prediction and prefetching of content.

The prefetching strategies mentioned are strictly application specific. NCP overcomes this limitation by its unique prediction scheme. NCP is also simplistic in implementation and doesn't require any modifications on the application side.

8. CONCLUSION

We have presented Name-Centric Prefetching (NCP), a protocol that aims to identify application interest patterns solely based on previously received interests, and uses the identified patterns to prefetch data. NCP is application-oblivious, and is able to identify and prefetch for applications that generate correlated, but not perfectly deterministic, interest patterns. Moreover, NCP can be directly implemented within Named-Data Networking (NDN), with multiple modules to ensure flexibility and the ease to manage computation, storage, and bandwidth overheads. Testbed emulation results suggest that NCP can be a promising approach to reduce user experienced latency.

It should be noted that this research work does not attempt to optimize its algorithms for managing various modules and prefetching data. Rather, the thesis presents a platform that makes it possible to implement and test new algorithms. Developing new algorithms that leverage patterns identified by NCP to further improve performance can be an interesting future research direction.

REFERENCES

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[2] O. Hahm, E. Baccelli, T. C. Schmidt, M. Wählisch, C. Adjih, and L. Massoulié, "Low-power internet of things with ndn & cooperative caching," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 98–108, 2017.

[3] I. Moiseenko and D. Oran, "Path switching in content centric and named data networks," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 66–76, 2017.

[4] J. Pfender, A. Valera, and W. K. Seah, "Performance comparison of caching strategies for information-centric iot," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pp. 43–53, 2018.

[5] J. A. Khan, C. Westphal, J. Garcia-Luna-Aceves, and Y. Ghamri-Doudane, "Nice: Network-oriented information-centric centrality for efficiency in cache management," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pp. 31–42, 2018.

[6] H. B. Abraham, J. Parwatikar, J. DeHart, A. Drescher, and P. Crowley, "Decoupling information and connectivity via information-centric transport," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pp. 54–66, 2018.

[7] O. Ascigil, S. Rene, I. Psaras, and G. Pavlou, "On-demand routing for scalable name-based forwarding," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, pp. 67–76, 2018.

[8] A. Rodrigues, P. Steenkiste, and A. Aguiar, "Analysis and improvement of name-based packet forwarding over flat id network architectures," in *Proceedings of the 5th ACM Conference on*

Information-Centric Networking, pp. 148–158, 2018.

[9] E. Newberry and B. Zhang, “On the power of in-network caching in the hadoop distributed file system,” in *Proceedings of the 6th ACM Conference on Information-Centric Networking*, pp. 89–99, 2019.

[10] M. Zhang, V. Lehman, and L. Wang, “Scalable name-based data synchronization for named data networking,” in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9, IEEE, 2017.

[11] W. Drira and F. Filali, “A pub/sub extension to ndn for efficient data collection and dissemination in v2x networks,” in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pp. 1–7, IEEE, 2014.

[12] W. Shang, A. Gawande, M. Zhang, A. Afanasyev, J. Burke, L. Wang, and L. Zhang, “Publish-subscribe communication in building management systems over named data networking,” in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–10, IEEE, 2019.

[13] Z. Zhu and A. Afanasyev, “Let’s chronosync: Decentralized dataset state synchronization in named data networking,” in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, IEEE, 2013.

[14] J. Tadrous, A. Eryilmaz, and H. El Gamal, “Proactive content download and user demand shaping for data networks,” *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1917–1930, 2014.

[15] J. Tadrous and A. Eryilmaz, “On optimal proactive caching for mobile networks with demand uncertainties,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2715–2727, 2015.

[16] K. Chen and L. Huang, “Timely-throughput optimal scheduling with prediction,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2457–2470, 2018.

[17] R. Liu, E. Yeh, and A. Eryilmaz, “Proactive caching for low access-delay services under uncertain predictions,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, pp. 1–46, 2019.

[18] [n.d.], “Named data networking.” `http://named-data.net/`.

[19] [n.d.], “Named data networking codebase.” `https://github.com/named-data/`.

[20] [n.d.], “Ndn-cxx library.” `http://named-data.net/doc/ndn-cxx/current/`.

[21] [n.d.], “Ndn-cxx codebase.” `https://github.com/named-data/ndn-cxx`.

[22] J. L. Hellerstein., “Google cluster data,” *Google Research Blog.*, Jan. 2010.

[23] V. Dimitrov and V. Koptchev, “Psirp project–publish-subscribe internet routing paradigm: new ideas for future internet,” in *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, pp. 167–171, 2010.

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

[25] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A data-oriented (and beyond) network architecture,” in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 181–192, 2007.

[26] A. Chandra, J. Weissman, and B. Heintz, “Decentralized edge clouds,” *IEEE Internet Computing*, vol. 17, no. 5, pp. 70–73, 2013.

[27] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, “Packetcloud: an open platform for elastic in-network services,” in *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture*, pp. 17–22, 2013.

[28] S. K. Dandapat, S. Pradhan, N. Ganguly, and R. Roy Choudhury, "Sprinkler: distributed content storage for just-in-time streaming," in *Proceeding of the 2013 workshop on Cellular networks: operations, challenges, and future design*, pp. 19–24, 2013.

[29] P. Deshpande, A. Kashyap, C. Sung, and S. R. Das, "Predictive methods for improved vehicular wifi access," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, pp. 263–276, 2009.

[30] 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*, pp. 55–60, 2012.

[31] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networks," in *2012 Proceedings IEEE INFOCOM Workshops*, pp. 268–273, IEEE, 2012.

[32] J. Quevedo, D. Corujo, and R. Aguiar, "Consumer driven information freshness approach for content centric networking," in *2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pp. 482–487, IEEE, 2014.

[33] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *2012 Proceedings IEEE INFOCOM Workshops*, pp. 316–321, IEEE, 2012.

[34] J. M. Wang and B. Bensaou, "Progressive caching in ccn," in *2012 IEEE global communications conference (GLOBECOM)*, pp. 2727–2732, IEEE, 2012.

[35] J. Zhang, R. Izmailov, D. Reininger, and M. Ott, "Web caching framework: Analytical models and beyond," in *Proceedings 1999 IEEE Workshop on Internet Applications (Cat. No. PR00197)*, pp. 132–141, IEEE, 1999.

[36] C. Bernardini, T. Silverston, and O. Festor, "Mpc: Popularity-based caching strategy for content centric networks," in *2013 IEEE international conference on communications (ICC)*, pp. 3619–3623, IEEE, 2013.

[37] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, “Globally distributed content delivery,” *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[38] F. Zhang, C. Xu, Y. Zhang, K. Ramakrishnan, S. Mukherjee, R. Yates, and T. Nguyen, “Edge-buffer: Caching and prefetching content at the edge in the mobilityfirst future internet architecture,” in *2015 IEEE 16th International Symposium on a World of wireless, mobile and multimedia networks (WoWMoM)*, pp. 1–9, IEEE, 2015.

[39] N. Abani, T. Braun, and M. Gerla, “Proactive caching with mobility prediction under uncertainty in information-centric networks,” in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pp. 88–97, 2017.

[40] H. N. Arifin, L. V. Yovita, and N. R. Syambas, “Proactive caching of mobility prediction prefetch and non-prefetch in icn,” in *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*, pp. 418–422, IEEE, 2019.

[41] D. Nguyen, Z. Shen, J. Jin, and A. Tagami, “Icn-fog: An information-centric fog-to-fog architecture for data communications,” in *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2017.

[42] T. Stockhammer, “Dynamic adaptive streaming over http– standards and design principles,” in *Proceedings of the second annual ACM conference on Multimedia systems*, pp. 133–144, 2011.

[43] Y.-T. Yu, F. Bronzino, R. Fan, C. Westphal, and M. Gerla, “Congestion-aware edge caching for adaptive video streaming in information-centric networks,” in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pp. 588–596, IEEE, 2015.

[44] T. Muto, K. Kanai, and J. Katto, “Implementation evaluation of proactive content caching using dash-ndn-js,” in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2239–2244, IEEE, 2015.

[45] K. Ueda and A. Tagami, “Icn-pep: Icn performance enhancing proxy for the efficient global content distribution,” in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[46] K. Ueda and A. Tagami, “Icn performance enhancing proxies for the global content distribution,” in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pp. 253–254, IEEE, 2018.