Discovering in-network Caching Policies in NDN Networks from a Measurement Perspective

Chengyu Fan
Colorado State University
Fort Collins, CO
chengyu.fan@colostate.edu

Susmit Shannigrahi
Tennessee Tech
Cookeville, TN
sshannigrahi@tntech.edu

Christos Papadopoulos
Colorado State University
Fort Collins, CO
christos.papadopoulos@colostate.edu

Craig Partridge
Colorado State University
Fort Collins, CO
craig.partridge@colostate.edu

ABSTRACT

Caching is integral to Named Data Networking (NDN). Routers in NDN networks are encouraged to cache content and serve later requests from their caches.

As NDN has evolved, researchers have realized that different caching schemes work better for different types of content and patterns of content requests. From a measurement perspective, this means that being able to determine the caching schemes in use within an NDN network can be essential to understanding the network's performance.

In this paper, we investigate the feasibility of detecting NDN caching schemes via active measurement (i.e., by sending requests into the network and measuring responses) from edge systems (e.g., by users). We show it is possible to determine what algorithms routers are using to decide what content to cache. Furthermore, for stochastic caching schemes with fixed caching probabilities, we show it is possible to infer the caching probability. Finally, while we do not seek to understand routers' cache replacement policies (which we leave to future work), we find that the methods for determining the caching algorithm are robust to cross traffic that may impact the content of a router's cache.

CCS CONCEPTS

• **Networks** → **Network measurement**; *Network simulations*; Network monitoring.

KEYWORDS

Named Data Networking, network measurement, caching

ACM Reference Format:
Chengyu Fan, Susmit Shannigrahi, Christos Papadopoulos, and Craig Partridge. 2020. Discovering in-network Caching Policies in NDN Networks from a Measurement Perspective. In *7th ACM Conference on Information-Centric Networking (ICN '20), September 29-October 1, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3405656.3418711

ICN '20, September 29-October 1, 2020, Virtual Event, Canada

ACM ISBN 978-1-4503-8040-9/20/09…$15.00
https://doi.org/10.1145/3405656.3418711

1 INTRODUCTION

Caching is a central feature of Named Data Networking (NDN) [26]. In NDN, *consumers* request uniquely named content, using an *Interest* packet, and the network then seeks to fulfill the request with a *Data* packet that contains the uniquely named content. Because content is uniquely named, it can be cached in routers' local storage (in NDN, the *Content Store*). Every time a router sees a Data packet, it must decide whether to cache the packet, hoping that there will be a later Interest for the same content.

Caching can decrease consumers' access latency. Caching can also reduce loads on the *producers* of data. Because duplicate or later requests for the same content can be satisfied from caches in routers, producers are less likely to be overwhelmed when their content is popular.

In-network caching systems' performance depends on many factors, such as content popularity, the routing of content request packets, when routers decide to cache, when the cache is full, and, how routers decide which content to replace with new content. Furthermore, experiments suggest that different caching approaches may better serve different applications. A number of different NDN caching policies have been developed (see section 3). Since a caching policy can be applied to a specific name prefix, ISPs have the potential to provide differentiated caching services or other novel caching-related content services. Different routers in an NDN network may implement different caching policies.

Suppose that you are a user (consumer), or an application developer, or a content creator (producer), and you see signs that the content you are requesting, distributing, or creating is not being distributed efficiently in an NDN network. Clearly, you would like to know how the network's routers are caching the content. The knowledge could help content creators verify their caching agreement with ISPs. Being able to infer caching policies of other ASs might also allow an AS to determine local caching policies effectively and perform traffic engineering.

Broadly, there are three possible approaches for learning the routers' content caching policies:

- *Interrogate* each router and ask that router how it would cache the content;
- *Passively measure* NDN traffic from one or more vantage points and inferring the caching pattern from the observed traffic behavior; and
- *Actively measure* by sending Interests into the NDN network and inferring the caching pattern from the observed responses.

We assume that the caching measurement happens in a future NDN-based Internet that is similar to today's IP-based Internet. Many ISPs cooperate to distribute data to consumers. Each ISP may have its own routing, forwarding, and caching policies. Similar to BGP policies in today's IP networks, each ISP has information about its own caching policies. However, many ISPs are unwilling to reveal their caching policies to others or the public [25]. This work focuses on the active measurement from edge systems. We envision that active measurement at the edge systems (e.g., by users or vantage points in open measurement platforms) could give NDN users the ability to detect the caching policy in use.

To this end, we present the first active measurement scheme to detect caching algorithms in NDN networks in this paper. We show that by periodically repeating probes for the same content, and measuring the difference in response times, we can infer the in-network caching algorithms. The method is robust in the presence of cross traffic, and we can use it to estimate the pre-set probability value for static probabilistic caching decisions.

The rest of the paper is organized as follows. Section 2 gives existing measurement NDN tools. We introduce some common caching mechanisms in NDN networks in Section 3 as background knowledge. Section 4 points out our objectives and describes our method. We show the profile of some common caching decisions in Section 5 and then investigate the robustness of our method in the presence of cross traffic. Section 6 demonstrates the results of apply our method on a real topology, and Section 7 concludes the papers.

2 RELATED WORK

NDN is a new network architecture and network measurement is one of the understudied challenges in NDN.

The shift to a content-centric based communication mechanism fundamentally changes the way to measure NDN networks. The rich in-network functionalities not only improve data distribution performance but also introduce complexity in NDN networks. Due to this complexity, only a few tools are currently available in NDN measurements. These tools give users limited capabilities to understand network performance, and identify and troubleshoot network issues.

In general, network measurement needs tools to measure network performance, traffic, and in-network state. NDN can be seen as a superset of IP. NDN could name data in various ways where the IP address is one feasible format. Therefore, NDN needs measurement tools/methods to cover more aspects than IP.

To the best of our knowledge, NDN has a few measurement tools [2, 7, 11, 22] that cover some aspects. Many of them focus on essential properties such as reachability (e.g. ndnping [22]) and the number of available paths (Contrace [also called CCNinfo] [1, 2], ccnx-trace [17], and NDN-trace [7]). Marchal et al. [11] investigated the server-side performance for generating NDN Data.

Obviously, no tool or method exists to measure NDN-specific network states, i.e., forwarding state, and caching state. Research is needed to allow people to measure NDN-specific network states and understand their effects on network performance. This paper presents the first caching policy detection method to fill the gap in NDN measurements.

3 CACHING POLICIES IN NDN

A number of caching policies have been tested in NDN networks [3, 6, 15, 16, 27].

Typically, NDN caching policies contain two parts: the caching decision and the cache replacement decision [13, 14]. In this paper, we focus on detecting the caching decision algorithms. As the background to understanding our results, this section presents the different caching decision algorithms that have been implemented in NDN networks. We conclude this section with a brief discussion of cache replacement, though we do not measure it in our study.

3.1 Caching Decision

The caching decision is to determine whether a packet will be stored in the CS or discarded. Typically, an NDN node needs to make such a decision when a Data packet arrives at an NDN node that does not have the packet in its CS.

3.1.1 Cache Everything Everywhere (CEE). CEE is the most straightforward caching decision strategy. With CEE as the strategy, NDN nodes attempt to cache every incoming Data packet that is not already in their CS. Caching Data packets everywhere provides the rapid replication of all available Data packets through the network. Due to its simplicity, the CEE strategy is widely used in NDN networks.

CEE leads to high redundancy, especially in a strongly connected network [13]. This redundancy can hurt the overall caching efficiency of a network. In a network with severely limited caching capabilities, CEE is likely to waste precious caching resources.

Caching efficiency can be measured using *diversity* [13], which is the ratio of unique content objects in all caches to unique content producers. Other advanced caching strategies attempt to maximize this metric by ensuring different router saves different Data packets.

3.1.2 Leave Copy Down (LCD). NDN networks with the LCD caching policy always cache Data packets only at the first node to receive the Data [14]. The "first node" is the next hop from the node where the cache hit occurred towards the consumer.

LCD is chosen when the network wants to improve the caching diversity without introducing complex communication into the caching process. LCD tends to keep Data packets close to the producer in the retrieval process, and the saved copy could alleviate the load on the producer as long as it stays on the data retrieval path.

3.1.3 Static Probabilistic Caching. The easiest solution to decrease redundancy is to apply a certain probability when a router needs to decide whether it should cache the incoming Data chunk[1]. The

[1]This paper uses Data chunk and Data packet interchangeably.

probability that an NDN router will store the incoming data chunk could be decided by the network operator. Whenever a router receives a Data chunk, it generates a random number between 0 and 1. If the generated number is smaller than the pre-set probability p, the router saves the chunk in its CS. Otherwise, the router does not cache this chunk, simply forwarding it to the downstream.

Since not every Data chunk is cached at every router, the cache diversity with probabilistic caching across the network will definitely be higher than that with CEE. Chunks requested and sent more often than others have a higher chance of being stored at more routers, as they arrive at routers more often.

Let n be the number of times that a node receives the same Data chunk, the caching probability at that node is $1-(1-p)^n$ [28].

A lower p will result in higher cache diversity, and CEE could be treated as a special case of probabilistic caching with $p = 1$. However, a wide range of value has been considered [20], such as 0.7, 0.5, 0.3, and other values.

3.1.4 Dynamic Probabilistic Caching. Some caching strategies adopt dynamic probability to better adapt the caching behavior to the network state. In these strategies, each router computes a caching probability individually for itself or even for each Data packet, based on available information, such as node-local information, Data chunk information, or even the information from the wider network. For example, local router ID, the position of the caching node in the network topology, and other information can be used to determine the probability.

ProbCache [15] is a dynamic probabilistic caching strategy that computes the caching probability of a given Data chunk based on the position of the caching node. The position is decided by the total number of hops between its producer and the consumer that requested it, as well as the number of hops remaining on the path to the consumer [14]. ProbCache achieves this by extending the Data packet with two new fields, namely, Time Since Birth (TSB) and Time Since Inception (TSI). TSB counts the number of hops since the creation ("birth") of the Data packet. TSB's initial value is 1. Every router increments the value when it forwards the Data packet. TSI is defined as the number of hops between the creation of the corresponding Interest packet and the cache hit. ProbCache calculates the cache weight as TSB/TSI. This way, the network edges would have a higher probability of storing Data chunks. One should notice that the caching probability is re-calculated each time. When a Data chunk is cached, TSI is the hop counts between the consumer and the router that caches the chunk.

Some researchers believe that caching closer to the server could be more beneficial in some topologies [14]. ProbCache-inv, a variant of ProbCache, stores more data chunks at the producer side by simply inverting the caching probability to $rand() < (1 - CacheWeight)$).

3.1.5 Other Strategies. This subsection briefly introduces a number of alternative strategies that consider various types of information in their mechanisms. However, due to their complexity and overhead, we do not describe them in this paper.

Cooperative caching uses more than local information in their caching decisions. The cooperation could be implicit or explicit. Implicit coordination strategies set up prior rules on nodes to avoid the need for explicit coordination. For example, Li et al. [8] propose a label-caching decision, evenly distributing chunks across the network. Users need to choose the subsets number k for chunks. Each router is preassigned a fixed label l at the beginning, where $l < k$. Nodes are only allowed to cache chunks whose segment ID modulo k is equal to l. Users could adjust k to automatically stratify chunks as needed.

In explicit coordination, nodes require continuous communication among the nodes to maintain a consistent state [14]. Information contained in the communication is cache states or cached chunks. The strategy proposed by Liu et al. [9], for example, coordinates between nodes to construct a virtual backbone network with core nodes that are responsible for caching.

Similar to the approach above, other works have used various other information that can help caching decisions. Vural et al. [24] propose a strategy that uses content popularity to calculate the cost function for the caching of incoming chunks. Chai et al. [4] utilize the network topology to save content chunks at the most central node. All caching strategies claim to generate reasonable good performance, but currently, there is no consensus on what information a good strategy should have.

3.2 Cache Replacement Policy

The cache replacement policy is to determine how to choose chunks that need to replace when a router has reached its capacity.

Some commonly used cache replacement decisions are Least Recently Used (LRU), Least Frequently Used (LFU), and Random Replacement (RR). In LRU, the router always chooses the one that was least recently requested. LFU, as a variation on LRU, keeps track of how often the objects in the cache are requested and evicts the least popular ones. As the least complex of the basic cache replacement policies, RR simply evicts random content objects every time it needs to replace a chunk.

NFD forwarder uses Priority-FIFO as the default replacement policy [23]. In this paper, we assume the default cache replacement policy is used.

4 DISCOVERING IN-NETWORK CACHING DECISIONS

This section discusses our objectives and briefly describes our experimental methodology.

4.1 Objectives

Network users can benefit from knowing what caching mechanism the network is using. Misconfiguration of caching policies may happen when an ISP provides differentiated caching services or other novel content services. Mistakenly assigning a lower probability to a content provider who contracted with an ISP for higher probability violates the contract. The caching detection method could help a content provider verify its contract with an ISP. Besides, the caching policy might affect application development. For example, content providers prefer to cache chunks at the server side at the beginning, in order to utilize network storage to help serve a burst of requests. As we discussed earlier, the ProbCache-inv mechanism [15], instead of the default CEE caching mechanism, will work better. Knowing what caching mechanism the network is

using could help developers save time in debugging or tuning application performance. We admit that ISPs could access CS entries on each router to figure out the caching mechanisms. However, this approach needs special privileges or additional management protocols. Passive measurements may work in inferring caching policies by monitoring on-going traffic, but it requires not only special privileges but also sufficient traffic under a name prefix in the network. Relying on third-party databases may not work well either, as they may be incomplete or out-of-date.

Our goal in this work is to let edge-systems detect deployed caching mechanisms without the help of ISPs. Content providers detect the active caching policy in an ISP could be one of the example. When a content provider has contract with an ISP, he/she knows what the caching behaviors are supposed to be. Then the content provider can conduct measurements on chosen vantage nodes in an open measurement platform (e.g. RIPE Atlas [19], M-Lab [5], and other platforms in today's IP networks) to verify if the agreement is enforced.

A caching policy contains the caching decision and the cache replacement policy. Our method aims to detect the caching decision from the end-hosts for several reasons. First, caching decisions directly affect the data retrieval performance. Whenever a Data chunk arrives, NDN routers need to make caching decisions. The cached copies could improve data retrieval performance immediately. Second, end-users have no tools or methods to detect what caching decision policy is in use.

NDN routers are stateful, and naive active measurements may not produce correct results about the network's caching state. Injecting too many probe packets into the NDN network will place pressure on the network state or even change the network state. Our method needs to ensure the correctness by minimizing the measurement overhead.

Caching decision mechanisms may utilize any information to make decisions, such as local random numbers, topology information, data labels, or even traffic information. Our method should allow users to specify any NDN parameters and observe the network's behavior with generated traffic.

4.2 Methodology

An appropriate caching policy could benefit both the consumer and the producer. Caching can decrease consumers' access latency and reduce the loads on the producer. Consumers, producers, and application developers would like to check if the appropriate caching policy is used when they notice data are not distributed efficiently in NDN networks.

To this end, our method assumes that users can access both the client node and the server node so that they can generate the Data packets with various parameters. Users can specify name prefix, Data payload length, and other parameters to observe the caching behavior. The server-side could also deploy a service to answer probing Interests with required parameters following a protocol, but the protocol design is out of the scope of this paper. For the sack of simplicity, we assume that NDN routers use the best-effort forwarding strategy, and the default replacement policy is used in this paper. According to NFD Developer's guide [23], Priority-FIFO is the default replacement policy on NFD forwarder.

To capture the unique distribution of chunks, the client first sends out tens of unique Interests. The client then re-sends the same Interests (duplicates) for several rounds so that it can detect the change of hop counts of Data chunks from one round to another. We show that 50 Interests and 10 rounds are enough for detecting caching decisions in Section 5. Users can tune the number of probing Interests and rounds to minimize the overhead.

To be more consistent with typical use of the architecture, the tool appends a reserved name component (e.g. "/detectcaching") and a nonce to make each measurement session unique, and then the longest prefix matching can help fetch the same Data chunk with duplicate Interests. For example, when the target name prefix is "/ndn/conference/icn20", the tools append the name components "/detectcaching/<nonce>" to form the measurement name prefix "/ndn/conference/icn20/detectcaching/<nonce>" in the measurement. The unique name prefix ensures that no other measurements affect the on-going probing session. A chunk ID is appended to the prefix name of the probing session to make each Interest unique, and thus we can capture the hop count information for each chunk separately. Additionally, the $CanBePrefix$ field [21] is enabled in all the Interests, so that the cached Data chunks can satisfy these incoming Interests. After the client fetches a Data chunk, it saves the hop count. We then plot the hop counts in Violin Plots for caching mechanisms when the measurements are done.

We derive our method from the Leave Copy Down (LCD) caching mechanism [14], and then show that the method can apply to other mechanisms in Section 5. As shown in Figure 1, the client sends out Interests to trigger Data chunks from the server-side. In the first round, the client sends out Interests for data chunks 1, and 2 for Content "/data." These Interests are forwarded to the server. Upon the arrival of these Interests, the server responds with corresponding Data chunks. Since NDN routers have LCD caching decisions configured, the Data chunks carry a field to indicate the hop count away from the storage. According to the hop count field, router R3 will save all chunks locally. Other routers forward Data chunks downstream to the client. In the next round, the client sends out the same number of Interests again. All Interests are duplicate with new parameter values. Router R3 will respond with local copies. For the same reason, caching happens at the next router, R2. Similar caching behavior happens to round three as well.

The caching state and its change in the data retrieval process give a unique profile for caching decisions. The distribution of hop counts for Data chunks demonstrates the feature of retrieval in one round. In each round, the client could perceive the hop changes of each chunk after it fetches them. Specifically, the LCD caching mechanism puts all chunks that belong to the same round in the same hop. From one round to another, LCD gradually moves Data chunks from the upper router to the lower one. In the client's view, it can notice Data chunks in the second round have decremented hop counts. We can claim that the caching decision is LCD if the Data chunks have the same hop count in one round and the subsequent rounds have one less hop.

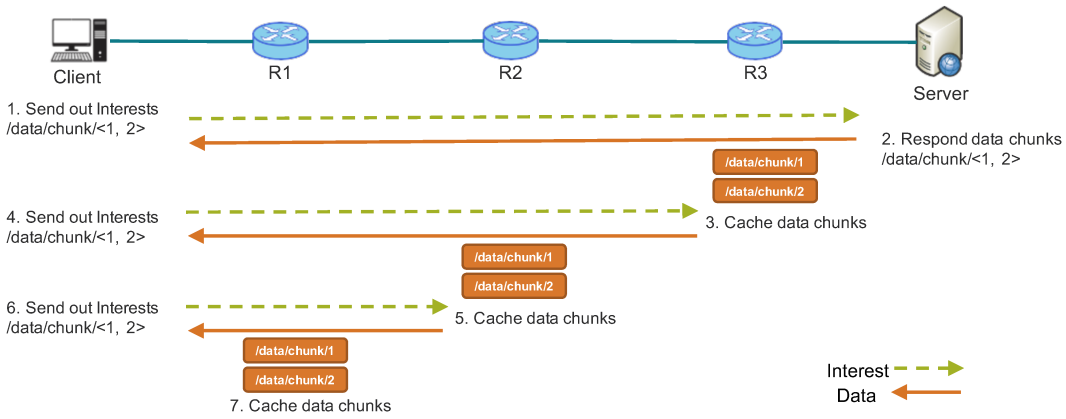


Figure 1: Caching state changes for LCD caching mechanism.

5 EMPIRICAL RESULTS

We use ndnSIM-2.7 [12] to validate our caching decision detection method. This section presents the experiment settings and empirical results.

5.1 Experiment settings

Our simulation assumes the content store (CS) is installed on all the routers except the first one, as that is usually the consumer's localhost. To simplify the scenario, we assume only one consumer and one producer, and the Best Route Strategy is configured on all the NDN nodes.

We also assume that the uniform caching policy is used. That means that the same caching policy is using on all the routers in each experiment. To the best of our knowledge, the benefits of using hybrid policies have not been studied, and operators tend to use the same policy for a name prefix. Managing the uniform policy is straightforward. Admittedly, operators may use hybrid policies intentionally or unintentionally, which we leave to future work. We envision that our method could detect the use of hybrid policies by comparing the results with uniform policy scenarios.

We first use a simple linear topology with 11 nodes to capture the profile for some common caching decisions. The topology has enough hops to plot chunk distributions for these caching decisions. Since the network is configured with the Best Route Strategy, the applications do not utilize the potential alternative paths provisioned by a complex topology.

In each round, the client sends out 50 Interests as probing packets. The client repeats sending duplicate probing messages 10 times to ensure that the changes in hop count distribution could be detected. Since data chunks in the first round must be from the server-side, we expect data chunks collected by other rounds show the hop count distribution and its changes. We then depict the hop count distribution and its changes in Violin Plots.

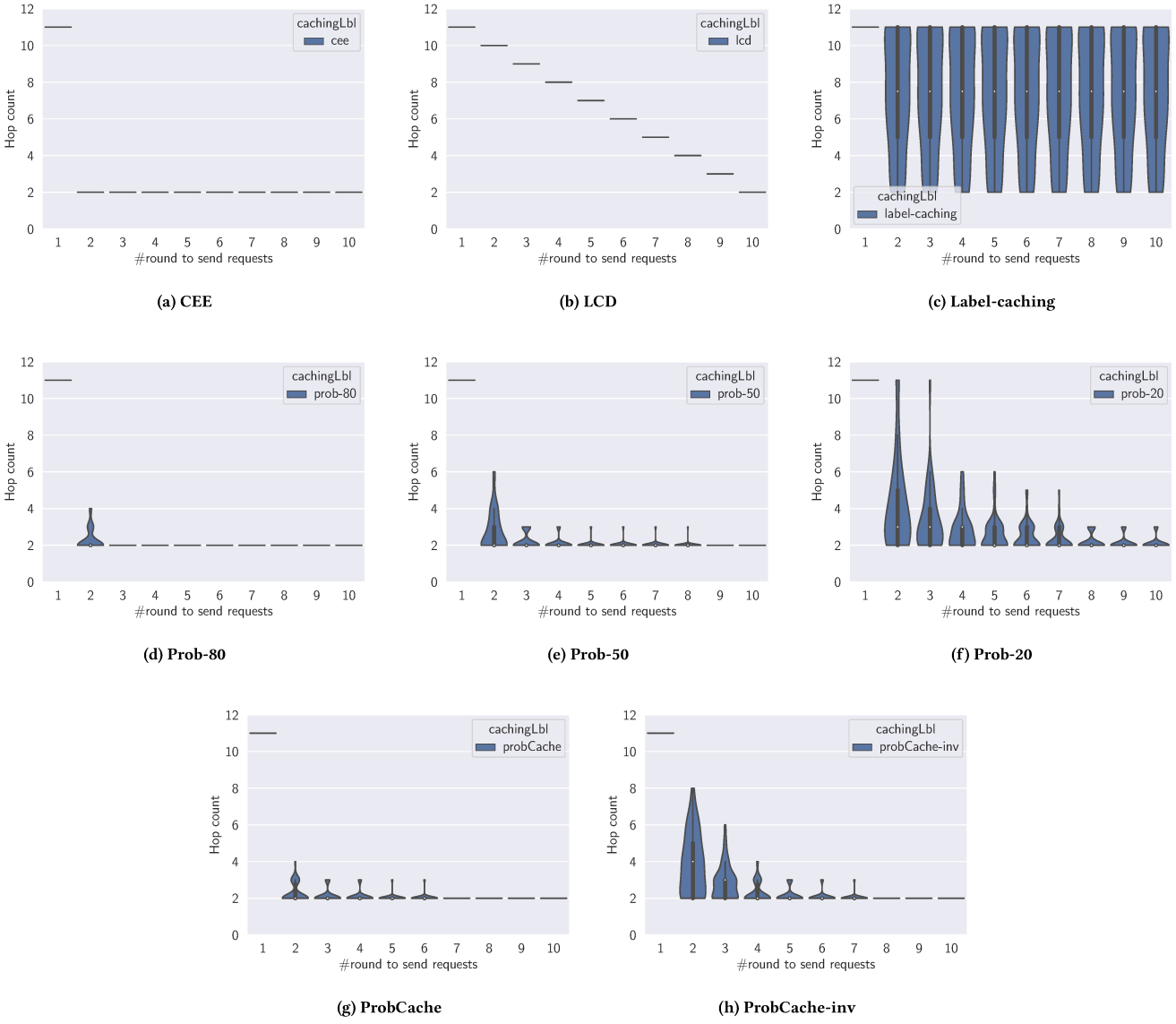
We use Violin Plots to demonstrate both the distribution for Data chunks and its change across multiple rounds for several reasons. First, chunks may appear on any routers that subject to the caching probability. Formulas are difficult to represent the chunk distribution. Second, other commonly-used plots cannot present both the metrics. For example, Box Plot tends to hide important details about how values in the data are distributed. On the contrary, a Violin Plot is a combination of a Box Plot and a Density Plot. The Violin Plot can show the shape of the distribution, the median value, and the thick black bar in the center to represent the interquartile range. Aside from showing the aforementioned statistics, a Violin Plot also shows the entire distribution shape of the data, which can be easily captured visually.

5.2 Caching decision profiles

Since ndnSIM [12] provides hop count information for each Data chunk, we plot Violin Plot for each caching decision mechanism. Figure 2 shows the profiles for some common caching decision mechanisms.

CEE, LCD, and Label-caching stand out for their distinct Violin Plot shapes. CEE caches everything everywhere. LCD always caches Data chunks at the next hop from the node where the cache hit occurred. When Lable-caching is in use, an NDN router only caches Data chunks whose IDs modulo k are equal to the assigned label l. It is obvious that both CEE and LCD deterministically save all Data chunks at a specific hop. The difference is the hop counts for each round. CEE starts to serve requests with the local copy since the second round. In Figure 2a, these chunks stay at the 2nd-hop for the rest of the rounds as the 2nd-hop router is the closest node equipped with CS. When LCD is using, the hop count for one round is always one hop closer to the client, until it arrives at the 2nd-hop router (Figure 2b). Lable-caching, however, caches chunks on multiple routers. This caching mechanism consistently keep chunks at specific hops. Duplicate Interests do not change the caching state. Figure 2c shows that all the rounds produce the same violin shape.

Static probabilistic caching is the easiest way to achieve higher cache diversity without increasing the complexity. Figure 2 shows the profile of static probabilistic caching decisions with probability 20, 50, and 80, respectively, dubbed as Prob-20, Prob-50, and Prob-80. Comparing with CEE and LCD, a major difference is that cached chunks have various hop count within one round when static probabilistic caching decisions are in use. Data chunks disperse along the path with some probability and form a violin shape. From one



(a) CEE **(b) LCD** **(c) Label-caching**

(d) Prob-80 **(e) Prob-50** **(f) Prob-20**

(g) ProbCache **(h) ProbCache-inv**

Figure 2: Profiles for various caching decisions.

round to another, duplicate Interests pull back Data chunks that are not cached on the 2nd-hop router, and the violin shapes change accordingly. The samples in the first round form a line, which indicates all chunks are from the server. The second round has the tallest violin shape in a specific static probabilistic caching. With subsequent rounds, the violin shapes become progressively shorter. Duplicate Interests let chunks go through the downstream routers, and they have a higher chance of being cached. The change of violin shape between rounds also represents the static probability. When the probability is 80 (Figure 2d), data chunks are quickly cached at the closest hop to the client. Probability 50 makes the process much slower. Even when most chunks are cached one hop away, it still takes another six probing rounds to cache all chunks at the 2nd-hop router (Figure 2e). Prob-20 is the worst one in terms of the time to cache all chunks at the 2nd-hop router (Figure 2f). After ten rounds, some chunks are still not cached at the closest router. We note that the height of the interquartile range for round two indicates the pre-set probability value for static probabilistic caching decision mechanisms. We present the approach to estimating the probability value in the next subsection.

We also investigate dynamic probabilistic caching mechanisms, which dynamically compute a caching probability for each individual node or even for each content chunk. ProbCache [15] is one of such caching mechanisms introduced in section 3. It computes the caching probability of a given Data chunk based on the total number of hops between its producer and the consumer that requested

it, as well as the number of hops remaining on the path to the consumer. In other words, the caching probability is larger when a chunk gets closer to the consumer in each round. Figure 2g shows that the height of the interquartile range for ProbCache is similar to Prob-80 when the client sends duplicate Interests for the first time. That is because ProbCache assigns a larger probability to routers that is closer to the client when the path is long. Apart from static probabilistic caching mechanisms, ProbCache recalculates the probability value for each round. We can find that caching all chunks at the closest router, which is the 2nd-hop router, takes another four rounds. The calculation rule makes the probability value drop when most chunks are cached around. If a chunk is cached at the 3rd hop in our topology, the probability to be cached at the 2nd hop is 2/3, which is less than the probability 80. ProbCache-inv [15] is identical to ProbCache in every way except that the final caching probability is inverted as $(1 - CacheWeight)$. Figure 2h demonstrates that ProbCache-inv has a taller violin shape, comparing with ProbCache. ProbCache-inv tries to cache chunks closer to the producer, so the probability drops when chunks get closer to the client. From the interquartile range at the 2nd round, ProbCache-inv is like a static probability between 50 and 20. However, ProbCache-inv assigns a higher probability when the router is closer to the Data sink (cache or producer). When the cache gets closer to the client, the caching probability on routers increases accordingly. Therefore, ProbCache-inv takes fewer rounds to cache all chunks at hop two.

To help readers capture the differences between profiles, Table 1 summarizes the key features of the profiles for the above caching decision policies. When CEE or LCD policy is using, the distribution of hop count at the 2nd round forms a line. However, other policies have violin shapes to represent the hop count distribution. The length of the black bar (i.e. the interquartile range) in Violin plots at the 2nd round varies from one policy to another. Label-caching, Prob-20, and ProbCache-inv have long black bars, but Prob-80 and ProbCache have short ones. Converge speed is the speed of caching all the Data chunks at the closest hop. CEE uses just one round to converge. LCD moves chunks one hop down towards the client gradually, and Label-caching never converges. We can also use the converge speed to identify the differences when Violin shapes are similar. Prob-20 converges slower than ProbCache-inv, and Prob-80 is much faster than ProbCache.

5.3 Estimation of static probability

Static probabilistic caching may accept any probability value between 0 and 1. The misconfiguration of probability value may dramatically change the behavior of caching behavior. Having a precise probability number could help end-users predict application performance or debug applications. Specifically, if we know the pre-set probability p, and a router receives the same chunk n times, we can estimate the probability of a chunk to be saved as $1 - (1 - p)^n$.

We could leverage the Violin Plot to estimate the static probability value as well. Since the probability value is applied to each incoming Data chunk, the overall cache diversity is improved. When the path between the client and the server is long enough, most Data chunks are likely to be cached. The ideal case is that all chunks are cached in the network. Comparing the plot shapes in measurements with the ideal case, we are able to estimate the pre-set probability value.

In a network that all routers have the content store (CS) installed, we can calculate, p_i, the percentage of chunks that the client can get from the i-th router as Formula 1 for the ideal case.

$$p_i = p(1-p)^{i-1}, i \in 1, ..., n. \quad (1)$$

Where:

- p: the pre-set probability for static probabilistic caching mechanisms;
- i: the hop number that starts from the client toward the server.

After getting the p_i, we can estimate the number of chunks that are supposed to be cached on the i-th router as np_i. Then, it is straightforward to plot a Violin Plot using the numbers of cached chunks on a router.

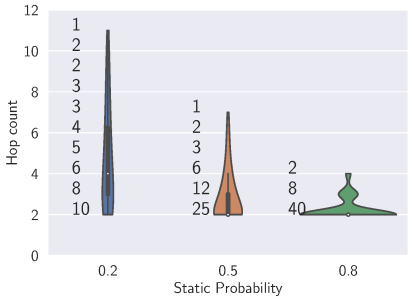


Figure 3: Estimated profile for three static probabilistic caching.

Using the above method, we plot the profile for the three static probabilistic caching in Figure 3. The plotted profile is a Violin Plot that shows the ideal distribution of cached chunks for the second round, which contains the largest number of samples. Since the first router has no CS installed in our experiments, the violin shape starts at hop two. The number of cached chunks at a hop is also annotated aside from the violin shape. As shown in Figure 3, the median for Prob-20 is around four, while Prob-50 and Prob-80 are zero. The interquartile range in these three probabilistic caching decisions is different too. Prob-20 has the longest interquartile range, while the interquartile range for Prob-50 is much shorter. Prob-80 has most chunks cached at the closest router, and thus the range height is zero. Comparing Figure 3 with Figure 2, we can find that Prob-50 and Prob-80 have similar plot shapes, but Prob-20 has slight differences between the simulation and the ideal case. The median for Prob-20 in Figure 2 is three. The bottom of Prob-20's interquartile range is two in our simulations, while the ideal case is three. However, the median, the interquartile range, and together with the violin shape form a unique fingerprint for caching mechanisms. The feature helps us approximately identify static probabilistic caching mechanisms with pre-set values.

Table 1: Summary of Caching Decision Policy Profiles

Caching decision policies	Cache Everything Everywhere (CEE)	Leave Copy Down (LCD)	Label-caching	Static Probabilistic Caching			Dynamic Probabilistic Caching	
				Prob-20	Prob-50	Prob-80	ProbCache	ProbCache-inv
Hop cnt Dist shape at 2nd Rd	line	line	violin	violin	violin	violin	violin	violin
len(Black bar) in Violin plot at 2nd Rd	N/A	N/A	long	long	medium	short	short	long
Converge Speed	1 round	hops of path	N/A	slow	medium	fast	medium	medium

5.4 The effects of cross traffic

Traffic sent by other applications may lead to competition on shared network resources (bandwidth, content store, and others). Competition on the bandwidth will trigger more packet drops. A large volume of data in the same direction (between client and server) may use out of the content store (CS) and trigger cache replacement events. When a large number of replacement events happen in a short time, many Data chunks will be evicted. Since our method needs cached chunks to plot Violin Plots, competition in the same direction may kick out Data chunks for measurement too often. In such case, the tool cannot plot reasonable good Violin Plots for detection. For the sake of simplicity, we assume that such a worst case does not happen in this paper.

We are more interested in the correctness of our method when encountering cross traffic, as it is more common. To introduce cross traffic, we attach the traffic generator at router eight and three in the linear topology. Cross traffic could happen at any router, but we believe putting traffic on either the server-side and the client-side could help us understand its effect separately. In our simulation, we only turn on one cross traffic at a time. To let cache replacement happen more often, we set the CS size to 100. The traffic generator produces sufficient traffic to overload the router.

In our simulation, most mechanisms are not sensitive to cross traffic. They may leave more data chunks at the server-side, and thus they have taller interquartile range, but that does not affect the correctness of detection. The only exception is LCD.

However, we can still use the method to identify the LCD mechanism. Figure 4b demonstrates that LCD cannot move forward to lower hops after hop nine (client node is the first hop) in the presence of cross traffic at router eight. LCD initializes the forwarding tag to one when a cached chunk is pulled out. If the Data chunk is evicted, the router is unable to attach such a tag, and the next-hop cannot receive a chunk that contains a caching signal. When the cross traffic happens at router three, the plot becomes a Violin shape when chunks reach the third router (Figure 4a). We can see that samples are not stuck at hop four, but round 8, 9, and 10 contain samples with deviations. In this case, the competition happens close to the client. The delay between the client and the router is small, and thus duplicate Interests get a chance to reach the router before eviction happens. In contrast, the delay between the client and the router eight is much larger. When Interests arrive, cross traffic has evicted all the chunks in the CS.

The profile of caching policies may vary when other cache replacement policies are in use. However, the default cache replacement policy is straightforward and efficient to evict cached probing packets, which gives a good scenario for verifying the robustness of the detection method. We plan to study the behaviors of our approach with caching replacement policies in the future. A function of the amount of cross traffic, the cache size, and the replacement policy may be useful in this direction.

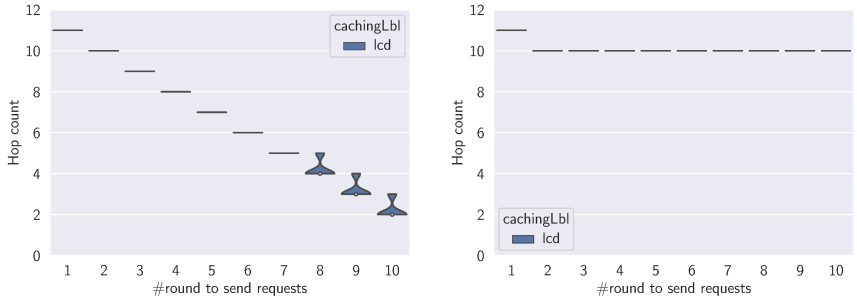
6 DETECTING ON A REAL TOPOLOGY

The previous section shows that using hop counts with Violin Plot could profile a caching decision mechanism. The profile can be used to estimate the probability value for static probabilistic caching mechanisms, and the method is robust in the presence of cross traffic. However, the NDN stack does not explicitly expose the hop count information to applications.

The client could use *HopLimit* to figure out how many hops it needs to fetch a specific chunk, but it may slow down the measurement and introduce a lot of overhead. The HopLimit is an Interest field to limit the number of hops the Interest is allowed to be forwarded [21]. To figure out the hop count of a chunk, the client needs to send out an Interest that starts with HopLimit one. If no data is received, it increments the HopLimit value to two and sends out the same interest until a data packet comes back. This approach has two issues. First, the client must send Interests to cover all hops until an Interest reaches the router that contains the data chunk. Most Interests are wasted without any Data chunks returned. These Interests introduce overhead not only into the bandwidth but also the PIT. Second, the measurement with HopLimit will be time-consuming. Interest has to wait until it times out when no Data comes back. No better way to speed up the measurement process without affecting the caching states. To this end, we use measured RTT at the client as the indicator of hops. When RTT does not work, we apply k-means [10] to estimate hop counts for chunks.

We simulate the measurement process using ndnSIM [12] on Rocketfuel topology 7018 [18]. The real topology contains delays and queuing size for each link, perfect for validating our method. We do not introduce other traffic in the network, as the point of this simulation is not for figuring out the effect of cross traffic. The simulation contains just one client and one server to exchange messages. They are randomly assigned to two nodes on the topology. Similar to the method mentioned before, the client sends out Interests to fetch Data chunks. Unlike previous experiments, the client calculates the RTT for each Data chunk. After collecting all samples, we use the RTT information to plot the Violin Plot.

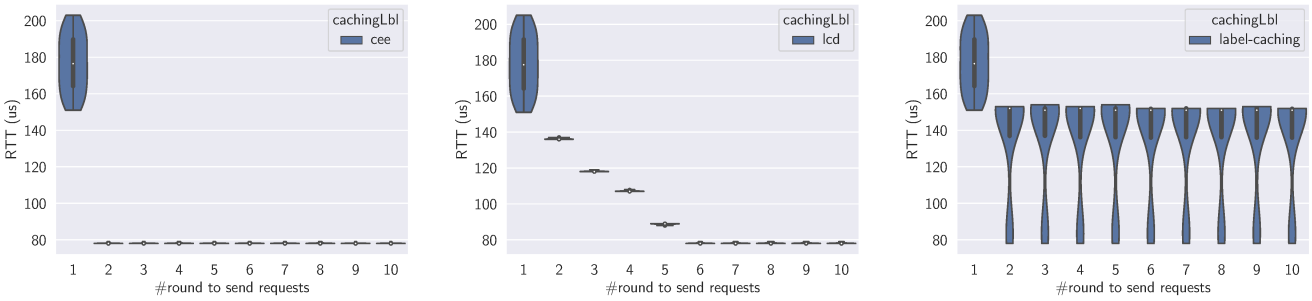
Figure 5 shows that simply using RTT in Violin Plot could detect some caching decisions for the chosen nodes. The delays for chunks from the server varies, but they do not affect us to identify caching mechanisms. Figure 5a clearly points out that CEE keeps all chunks at the closest hop since round two. The distribution of



(a) Cross traffic at client side

(b) Cross traffic at server side

Figure 4: Profiles for LCD mechanism when cross traffic happens.



(a) CEE

(b) LCD

(c) Label-caching

Figure 5: Profiles for various caching decisions using RTT.

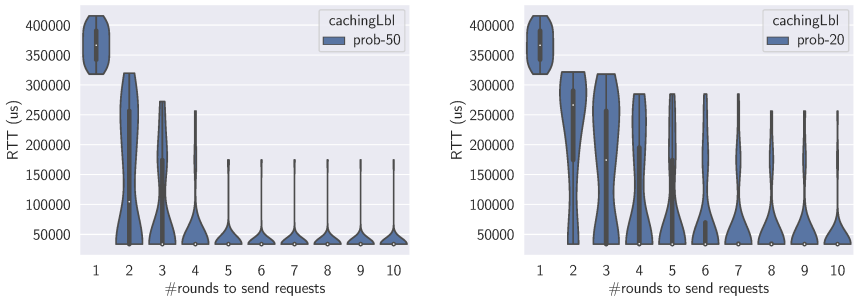
cached chunks never changes after that. Just like using hop counts, LCD moves data chunks hop by hop towards the client when using RTT (Figure 5b). Some samples have slight variances, but they do not change the plot shapes too much. Finally, Figure 5c demonstrates a similar violin shape for all rounds, which indicates the caching decisions do not change when encountering duplicate Interests. Among all the caching decisions, we know the label-caching mechanism is the one who has that unique feature.

RTT may not always be good enough to identify the deployed caching decisions. We choose other nodes to deploy the client and the server, and Figure 6 shows that the generated ViolinPlots for static probabilistic caching are misleading for the new pair of nodes. The reason is that some links have small delays, while others have large delays. RTT values group some samples visually, but they cannot represent the hop counts correctly.

We argue that the samples in the same group are from the same router. The rationale behind this is that the chunks from the same router go through the same links, and the Interests to pull these chunks use the same links as well. After grouping samples with similar RTTs, we can rank groups by their RTT values, and then each group can represent a hop.

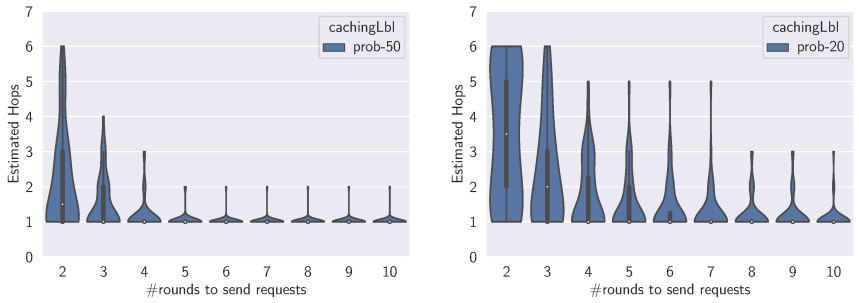
To this end, we apply the k-means clustering algorithm [10] to group samples. The k-means algorithm takes the collected RTT data and the target cluster number k as input, splitting the data into a fixed number (k) of clusters. The algorithm yields a cluster id associated with each sample in the data. We can then sort the clusters by the median RTTs. Each cluster is assigned a hop number, starting from hop one. Specifically, in our experiments, we specify six as the k value, and the generated plots present the reasonably good estimated hop counts. Figure 7 shows that the generated plots are similar to the ones in our experiments using hop counts (Figure 2). The probability applied on each Data chunk makes the violin shapes not exactly the same from one round to another. However, increasing the number of probing packets can reduce the deviation and minimize the differences.

In summary, we claim that using RTT with the k-means algorithm in our method is enough to identify caching decisions on the real topology. We are aware that the k-means clustering algorithm has the difficulty of deciding perfect k-value. The plot shapes generated by incorrect k-value is misleading in caching policy detection. For example, when the probability value 80 is using, the static probabilistic caching decision saves chunks on the first four hops from the client. In this case, we cannot produce the correct shapes with k-value six. Fortunately, the collected data contains RTT information for chunks. As long as we know the link delays as prior, we can estimate the range of hop counts as the k-value.



(a) Prob-50 **(b) Prob-20**

Figure 6: Profiles for various static probabilistic caching decisions.



(a) Prob-50 **(b) Prob-20**

Figure 7: Profiles for various static probabilistic caching decisions with estimated hops.

NDN-trace [7] is a perfect candidate tool to figure out the hop count and the link delays between the client and the server. We can use NDN-trace to set up a mapping between the hop count and the delays.

Many other clustering algorithms are available to group samples with similar delays in the same clusters. We can use any available algorithms for estimating the hop counts. Some clustering algorithms are designed to improve scalability. Some algorithms focus on large-scale data processing or high dimensional data processing. However, our experiments do not require high scalability. Our dataset has small size, typically hundreds of samples. Delay is the only dimension that the algorithms need to support.

7 CONCLUSION

The new networking paradigm introduced by NDN, in particular, its stateful data plane with caching and name-based forwarding, require a solution to detect caching mechanisms. In this paper, we present the first active measurement scheme to detect caching decisions. Our method lets the client send out a small number of Interests to request Data packets under the target name prefix.

After repeating the measurements several rounds, the client can collect necessary data chunk information to produce profiles. The profile contains the hop counts distribution and the distribution changes across rounds to identify a caching decision uniquely. We show that our method can estimate the probability value for static probabilistic caching mechanisms, and it is robust to cross traffic. We also apply our method on the real topology, and our results demonstrate that we can detect active caching decisions by mapping delays to hop counts.

This paper produces profiles for some common available caching decision mechanisms. Other mechanisms, such as the explicit cooperative caching decisions and caching decisions based on betweenness centrality of the caching node, may be deployed in NDN networks. Generating profiles for these mechanisms could benefit caching detection. Additionally, the mixed-use of caching decision schemes may be used intentionally or unintentionally. It may cause conflicts in saving chunks. We envision that comparing measurements with the ideal fingerprints could help identify misconfigured policies. Moreover, NDN has other forwarding strategies (e.g., Multicast, Load-balance, etc.) available, and we plan to study detecting caching decisions in the presence of other forwarding strategies.

In the future, we plan to apply the proposed method to NDN testbed for further study. We notice that it may not be straightforward to build a caching policy detection tool based on the plots' visual variation. We will propose the design for the caching detection tool in future work.

ACKNOWLEDGMENTS

The authors gratefully acknowledge support from the National Science Foundation grants OAC-1659403 as well as the anonymous reviewers for their comments and feedback.

REFERENCES

[1] Hitoshi Asaeda, Kazuhisa Matsuzono, and Thierry Turletti. 2015. Contrace: a tool for measuring and tracing content-centric networks. *IEEE Communications Magazine* 53, 3 (2015), 182–188.

[2] Hitoshi Asaeda and X Shao. 2018. *CCNinfo: Discovering content and network information in content-centric networks.* Technical Report. IRTF Internet Draft (work in progress).

[3] Giovanna Carofiglio, Vinicius Gehlen, and Diego Perino. 2011. Experimental evaluation of memory management in content-centric networking. In *2011 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[4] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. 2013. Cache "less for more" in information-centric networks (extended version). *Computer Communications* 36, 7 (2013), 758–770.

[5] Constantine Dovrolis, Krishna Gummadi, Aleksandar Kuzmanovic, and Sascha D Meinrath. 2010. Measurement lab: Overview and an invitation to the research community.

[6] Xiaoyan Hu, Jian Gong, Guang Cheng, and Chengyu Fan. 2015. Enhancing in-network caching by coupling cache placement, replacement and location. In *2015 IEEE International Conference on Communications (ICC)*. IEEE, 5672–5678.

[7] Siham Khoussi, Davide Pesavento, Lotfi Benmohamed, and Abdella Battou. 2017. NDN-trace: a path tracing utility for named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 116–122.

[8] Zhe Li and Gwendal Simon. 2011. Time-shifted tv in content centric networks: The case for cooperative in-network caching. In *2011 IEEE international conference on communications (ICC)*. IEEE, 1–6.

[9] Yinlong Liu, Dali Zhu, and Wei Ma. 2016. A novel cooperative caching scheme for content centric mobile ad hoc networks. In *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 824–829.

[10] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[11] Xavier Marchal, Thibault Cholez, and Olivier Festor. 2016. Server-side performance evaluation of NDN. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*. ACM, 148–153.

[12] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. 2015. ndnSIM 2.0: A new version of the NDN simulator for NS-3. *NDN, Technical Report NDN-0028* (2015).

[13] Jakob Pfender, Alvin Valera, and Winston KG Seah. 2018. Performance comparison of caching strategies for information-centric IoT. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*. 43–53.

[14] Jakob Pfender, Alvin Valera, and Winston KG Seah. 2019. Content Delivery Latency of Caching Strategies for Information-Centric IoT. *arXiv preprint arXiv:1905.01011* (2019).

[15] Ioannis Psaras, Wei Koong Chai, and George Pavlou. 2012. Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*. 55–60.

[16] Ioannis Psaras, Richard G Clegg, Raul Landa, Wei Koong Chai, and George Pavlou. 2011. Modelling and evaluation of CCN-caching trees. In *International Conference on Research in Networking*. Springer, 78–91.

[17] Susmit Shannigrahi, Dan Massey, and Christos Papadopoulos. 2017. *Traceroute for Named Data Networking.* Technical Report. Technical Report. NDN, Technical Report NDN-0055, Revision 2. https://named ....

[18] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review* 32, 4 (2002), 133–145.

[19] RIPE NCC Staff. 2015. Ripe atlas: A global internet measurement network. *Internet Protocol Journal* 18, 3 (2015).

[20] Saran Tarnoi, Kalika Suksomboon, Wuttipong Kumwilaisak, and Yusheng Ji. 2014. Performance of probabilistic caching and cache replacement policies for content-centric networks. In *39th Annual IEEE Conference on Local Computer Networks*. IEEE, 99–106.

[21] NDN Team. [n.d.]. *Interest Format.* Retrieved Aug 21th, 2020 from https://named-data.net/doc/NDN-packet-spec/current/interest.html

[22] NDN Team. 2015. *NDN Essential Tools.* Retrieved Dec 9th, 2019 from https://github.com/named-data/ndn-tools

[23] NFD Team. 2018. NFD Developer's Guide. *Technical Report, NDN-0021* Revision 9 (2018).

[24] Serdar Vural, Ning Wang, Pirabakaran Navaratnam, and Rahim Tafazolli. 2016. Caching transient data in internet content routers. *IEEE/ACM Transactions on Networking* 25, 2 (2016), 1048–1061.

[25] Feng Wang and Lixin Gao. 2003. On inferring and characterizing internet routing policies. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. 15–26.

[26] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.

[27] Tiankui Zhang, Xiaogeng Xu, Le Zhou, Xinwei Jiang, and Jonathan Loo. 2018. Cache space efficient caching scheme for content-centric mobile ad hoc networks. *IEEE Systems Journal* 13, 1 (2018), 530–541.

[28] Yanyong Zhang, Dipankar Raychadhuri, Luigi Alfredo Grieco, Emmanuel Baccelli, Jeff Burke, Ravishankar Ravindran, Guoqiang Wang, Anders Lindgren, Bengt Ahlgren, and Olov Schelén. 2015. Requirements and Challenges for IoT over ICN. (2015).