network

MDPI

Article

Enhancing Cache Robustness in Information-Centric Networks: Per-Face Popularity Approaches

John Baugh and Jinhua Guo *

Department of Computer and Information Science, University of Michigan-Dearborn, Dearborn, MI 48128, USA; jpbaugh@umich.edu

* Correspondence: jinhua@umich.edu

Abstract: Information-Centric Networking (ICN) is a new paradigm of network architecture that focuses on content rather than hosts as first-class citizens of the network. As part of these architectures, in-network storage devices are essential to provide end users with close copies of popular content, to reduce latency and improve the overall experience for the user but also to reduce network congestion and load on the content producers. To be effective, in-network storage devices, such as content storage routers, should maintain copies of the most popular content objects. Adversaries that wish to reduce this effectiveness can launch cache pollution attacks to eliminate the benefit of the in-network storage device caches. Therefore, it is crucial to protect these devices and ensure the highest hit rate possible. This paper demonstrates Per-Face Popularity approaches to reducing the effects of cache pollution and improving hit rates by normalizing assessed popularity across all faces of content storage routers. The mechanisms that were developed prevent consumers, whether legitimate or malicious, on any single face or small number of faces from overwhelmingly influencing the content objects that remain in the cache. The results demonstrate that per-face approaches generally have much better hit rates than currently used cache replacement techniques.

Keywords: Named Data Networking (NDN); Information-Centric Networking (ICN); cache pollution; hit rate; popularity; cache robustness

Citation: Baugh, J.; Guo, J. Enhancing Cache Robustness in Information-Centric Networks: Per-Face Popularity Approaches. *Network* **2023**, *3*, 502–521. https://doi.org/10.3390/network3040022

Academic Editor: Pingyi Fan

Received: 30 July 2023
Revised: 18 September 2023
Accepted: 26 October 2023
Published: 1 November 2023

1. Introduction

As the current host-based Internet architecture ages, it is becoming increasingly apparent that it is insufficient to handle the content-driven applications in use today. The original Internet architecture was built largely to accommodate end-to-end communication between mutually trustworthy individuals who wished to share data and expensive resources. Since then, the use of the Internet as a communications network has shifted toward its use as a content delivery apparatus.

Because of these limitations, as technology and user needs have changed, overlay patches providing security, privacy, mobility, and content delivery have been created to sit on top of the underlying network. These application overlays provide services that the original Internet architects would have never imagined: mobile communications, banking, shopping, multiplayer gaming, streaming media, and many other services. However, the application overlays are not able to benefit from any optimizations at the network layer because they are simply not provided.

As a result, many researchers have pushed for a complete redesign of the fundamental architecture of the Internet to focus on content as the primary citizen of the network instead of location. For many applications, users want specific content but do not care where it comes from as long as the content is genuine and transmitted reliably and quickly. One architectural paradigm that seeks to accomplish the goals of a future Internet redesign is Information-Centric Networking (ICN) [1–3].

Several projects within the large umbrella of ICN have sought to achieve these goals with various approaches and to varying degrees of success. Prominent ICN projects include

Data-Oriented Network (DONA) [4], Network of Information (NetInf) [5,6], Content-Centric Networking (CCN) [7,8], and Named Data Networking (NDN) [9].

Many interesting research challenges exist to provide the desired features of ICN. In this paper, we discuss technology related to in-network caching of data, which is a key feature to lowering latency and overall network congestion, using technology at the network layer. While in-network storage devices in ICN seek to provide similar services to those that Content Distribution (or Delivery) Networks (CDNs) [10] do today, in-network storage devices do so at the network layer and take advantage of various optimizations therein. In-network caching is discussed in a variety of recent research studies as a primary goal in ICN and NDN [11–13].

Given the pervasiveness of in-network storage devices in ICN architectures, these devices are prime targets for malicious users. In terms of in-network caching of popular data, an attacker would seek to reduce or remove any benefit experienced by the presence of in-network storage devices, such as content routers. An attack well suited for this type of service degradation is the cache pollution attack [14]. In this type of attack, the malicious attempt is to push truly popular content objects out of the cache in favor of less popular content objects. Ultimately, the goal is to reduce the hit rate experienced by users, such that the content that is genuinely popular is rarely or never in the content store of the in-network storage devices.

These types of attacks are not new—they have been experienced for years in the current IP-based Internet, such as in CDNs and other situations involving proxy caching servers [15]. However, these attacks will certainly be even more of a threat in an architecture where in-network storage is part of the fundamental fabric of the network and not just an overlay.

Several proposed solutions for preventing or reducing the effects of cache pollution attacks in ICN exist. However, most are insufficient for various reasons. Some lack adequate security and can be easily overcome, such as CacheShield [16]. Others assume all abnormal behaviors follow uniform distributions [17], and still others require significant computational or memory overhead, such as ANFIS [18]. Most current techniques attempt to detect attacks, which is often difficult when attackers employ various smart attacks. Additionally, these techniques can yield false positives if a user is even a little more aggressive in their request frequencies, and they may punish them even though they were not being malicious.

In this paper, we present variations of cache replacement techniques as part of larger Per-Face Popularity (PFP) schemes, which seek to normalize the contributions from each face (for our purposes, simply an interface) on a content storage router. This normalization prevents interest message contributions received on any one face from having more effect on what ends up being cached than the contributions on any other face.

The three primary algorithms that we have developed as variants of PFP are the original PFP scheme, PFP-DA, and PFP-β. The original PFP scheme involves a list of content object request frequencies for each face. This algorithm is effective but not very scalable, as these lists are unbounded. Also, our original scheme can be affected by so-called natural cache pollution, wherein a content object may have been very popular at one time, thus having many requests, but has fallen out of being truly popular but remains on the list due to the incredible frequency achieved once upon a time. To remedy the scalability issues and lack of recency, we developed a variation of PFP with dynamic aging added to the content frequency lists and limited the size of these lists to double the size of the actual cache. This resulted in our second scheme, PFP-DA (PFP with dynamic aging). This scheme was used for most of the tests in this work. Finally, we developed a third scheme, PFP-β (PFP with Beta Parameterization), wherein the PFP-DA scheme remained, but the contributions themselves were scaled using a β parameter. We tested different values for β to determine the effects and made many interesting discoveries as a result, including some scenarios with 0% cache pollution and higher hit rates than were experienced with PFP-DA itself, which already outperformed LFU-DA and LRU.

Our experiments demonstrate that PFP-based schemes are not only resistant to the effects of cache pollution attacks but also suggest that PFP-based schemes are better at representing the overall popularity of content objects across multiple faces. Therefore, the schemes work well even when there are no active attacks occurring. In terms of a more general contribution to the state of the art, our schemes employ a novel approach to cache replacement.

2. Related Work

2.1. Cache Pollution Attacks

Fundamentally, cache pollution attacks come in two different forms: locality disruption and false locality. The term locality refers to a phenomenon in which certain content objects from a given requesting region tend to be more popular in that region than other content objects. Thus, a specific arrangement of content objects based on their popularity is representative of a locale or region.

Therefore, locality disruption refers to an attempt by an attacker to simply churn the cache of a given locale in hopes of knocking some of the popular content objects out of the cache. Thus, the attacker will request many different content objects from across the universe of content (the content space). A false locality attack, on the other hand, is an arguably more aggressive attack in which an attacker wishes to push even the most popular content objects out of the cache in favor of unpopular items. Thus, the attacker wants the cache full of content objects that are not at all representative of the desires of users utilizing the cache. Hence, the locality expressed by the cache is false relative to the actual users' desires. It is important to note that most security approaches to cache pollution attacks tend to seek to prevent one (or sometimes both) of these types of attacks [10]. We will discuss these briefly.

2.2. Approaches to Secure against Cache Pollution Attacks

One of the best-known examples in the literature in terms of attempting to fight cache pollution is CacheShield, developed by Xie et al. [12]. This approach records the interest frequency of items that are not cached. As the recorded frequency t increases, the probability that a content object will be cached increases as well due to the shielding function. However, a major weakness of this approach is that an attacker need only determine the threshold of t values and request content objects at a frequency beyond the threshold of t. Then, their own content objects will be cached.

A cache pollution detection approach using randomness checks was proposed by Park et al. [19]. The randomness check approach utilizes the fact that attackers launching a locality disruption attack request contents in a nearly uniform manner across the content universe. The authors reason that it is less likely that attackers will be able to know what contents are popular and that this popularity changes with time. Therefore, they consider false locality attacks far less likely than locality disruption attacks, which do not require detailed knowledge of the popularity of contents. The approach performs matrix operations, during which popular contents are more likely to be removed from the matrices due to the Zipf-like distributions, whereas attacker requests will remain. This causes the system to rank content requests as malicious when they remain with a given rank value.

As an attempt to improve upon CacheShield [16], the authors of [17] assume a slightly more intelligent attack scenario. They enhance their threat model such that the attackers only focus on a small subset of the content space, namely the items at the tail end of the Zipf distribution. This approach uses a machine-learning-based approach. However, it still suffers from weaknesses, including the assumption that all normal behavior is Zipf-like and attackers are always uniform in their distribution of requests.

A neural network (with fuzzy systems) approach was proposed by Karami et al. in [18]. This approach uses techniques that are used in linguistics to determine the degree to which different inputs and outputs are related. Although this approach is reasonably effective at detecting cache pollution attacks (both false locality and locality disruption), the overhead,

both in terms of memory and computational complexity, is significant. For these reasons, this approach is not considered to be very scalable [14].

3. Enhancing Cache Robustness via Per-Face Popularity

In this section, we discuss three Per-Face Popularity, or PFP, approaches to increasing cache robustness by means of normalizing popularity contributions across all faces. Firstly, we discuss the original Per-Face Popularity algorithm, or simply, PFP. Then, we look at a more efficient and scalable approach that also considers the recency of requests, known as Per-Face Popularity with Dynamic Aging (PFP-DA). Finally, we discuss a parameterized approach in which exponentiation is used on contributions to reduce numeric distance among the overall popularity values of ranked contents, known as Per-Face Popularity with Beta-Parameterization, or PFP-β (also, PFP-Beta).

3.1. Per-Face Popularity (PFP) Scheme

With schemes like LFU-DA, all received interests are used much like votes, regardless of their origin. For example, if 100 interests for content A arrive on face 1, and 20 interests for content A arrive on face 2, face 1 has contributed five times the number of interests as face 2. Thus, face 1 contributes more to determining whether content A is cached or not than does face 2. If an attacker manages to overwhelm a router with one face with far more interests than any other face, then the attacker can cache essentially whatever he wants. Although some mechanisms, explored earlier, try to detect an attack based on content request frequency, this is not always appropriate and can lead to false positive identification of attackers where none exist. A face might simply receive more requests for a given content object than other faces, which does not immediately indicate attacker behavior.

The original PFP scheme we developed was based on the concept of ensuring fairness of contributions across all faces on the content storage router. Therefore, no individual face has any more influence on what content objects are cached than any other face. The frequency with which a content object is requested is normalized such that the interests arriving over face 1 do not contribute any more to the cache replacement decisions than the interests arriving over face 2, and so on.

The reasoning for our design of PFP is twofold. Firstly, the effect an attacker can have on a cache is mitigated, thus reducing cache pollution and increasing the hit rate by legitimate users, especially the hit rate of the most popular items. Secondly, a latent benefit of our attack effect mitigation efforts, specifically normalization of contributions across faces, is that it is more likely that content that is truly popular across more faces will remain in the cache.

To implement our scheme, we made relatively significant additions and modifications to the ndnSIM environment. Although this will be discussed in more detail later, it is important to note that several design decisions had to be considered. As far as being implementable, we could have given each face its own cache (as a fair portion of the overall cache size), but from a performance and scalability perspective, this would have been a poor design decision. Instead, maintaining a single cache and normalizing the contributions was the best approach.

At the core of our scheme is a class known as the popularity manager (PopMan), which maintains lists of content request frequencies based on each face. The name of the content object being requested is recorded with the initial interest from a given face, and subsequent interests for that content object (on that face) cause the frequency value being recorded to be increased. This naturally creates a ranking of the most popular items for each face.

Cache Replacement Algorithm

When the actual content storage router's cache is full, a cache replacement algorithm is initiated to determine what must be replaced to make room for new content. In PFP, the cache replacement algorithm uses the normalized contributions for content objects across

all faces to determine true overall popularity values, and thus to determine what should be removed from the cache. We make use of an item's rank for each face and denote it as r. As the list of most popular items (on a given face) is maintained, essentially sorted by requests (with dynamic aging element), each item therefore has a rank. This is where the value r comes from, that is, from the position within the ranked list of items on a given face. Note that each face contributes to the overall calculation by using r, which is normalized by taking its reciprocal. This results in the Per-Face Popularity contribution. That is, each face contributes the reciprocal of r to overall popularity.

$$\textit{Per Face Popularity Contribution} = \frac{1}{r} \tag{1}$$

The overall popularity of a content object, C, across all faces is the sum of the Per-Face Popularity contributions from all faces. When a face does not have a given content object ranked, the overall contribution is considered 0. The following equation shows the overall popularity calculation, with r_i being the rank of C from the content frequency list for face i.

That is,

$$P(C) = \sum_{i=1}^{n} \frac{1}{r_i} \tag{2}$$

3.2. Per-Face Popularity with Dynamic Aging (PFP-DA)

Although the original PFP scheme showed promise, a couple lingering issues were identified. Namely, that the scheme did not consider recency at all, and that the content frequency lists maintained by the popularity manager could be arbitrarily large. The first issue is a problem since a content object that might have been popular a significant time ago would remain in the cache, even if it was not currently popular, simply due to the extreme number of requests received once upon a time. This, of course, reduces the reflection of true locality for a given cache and contributes to natural cache pollution, in which there is no attacker, yet content objects continue to reside in the cache even though they no longer improve the hit rate significantly. The second issue, the ever-increasing content frequency list size, is clearly a significant problem when considering performance and scalability issues. Therefore, a revised scheme was needed to address these issues.

Borrowing insight from the Least-Frequently Used with Dynamic Aging (LFU-DA) [20] algorithm, we modified our original PFP algorithm to include recency considerations, resulting in the Per-Face Popularity with Dynamic Aging (PFP-DA) scheme. The dynamic aging aspect of LFU-DA directly affects the cache itself. However, in PFP-DA, the dynamic aging aspect affects each of the content frequency lists. Therefore, aging affects the actual cache, but indirectly. In the PFP-DA scheme, the content frequency lists are limited to a finite size based on the actual cache size. In our implementation, the lists are twice the size of the true cache (e.g., 200-size lists for a 100-size cache). For the majority of cache sizes, this will essentially solve our first problem, in that the list sizes are no longer theoretically unbounded.

Instead of using the frequency of interests for a given content when determining if it belongs in the cache itself, PFP-DA uses the key, K_i for a given content object i. This key is calculated as follows:

$$K_i = F_i + L \tag{3}$$

The value F_i is the frequency for the given content object, i, and L is the *age factor*. L is initialized to 0, and then whenever a content item is to be replaced, L is updated to be equal to the key, K_f, of item f, that is, the item being removed. Succinctly, this means $L = K_f$. With respect to PFP-DA, aging is utilized with the rankings for each of the faces rather than overall content caching.

3.3. Per-Face Popularity with β-Parameterization

PFP-DA yields very good results, as we will see later in this article. However, even slight improvements on top of PFP-DA would be welcome. Therefore, we take advantage of the fact that the overall popularity calculation looks like the formula for the Zipf distribution. Therefore, we considered the impact that introducing a parameter similar to the α value used in Zipf-like distributions would have on the cache, including pollution and hit rates. Thus, our formula for calculating the overall popularity became the following:

$$P(C,\ \beta) = \sum_{\substack{i=1 \\ 0 \le \beta \le 1}}^{n} \frac{1}{r_i{}^{\beta}} \tag{4}$$

The parameter, β, is introduced as being between 0 and 1, inclusive. Thus, our regular PFP-DA would be considered a special case of the above equation with β = 1. This new parameterized scheme is designated PFP-Beta, or PFP-β. It still maintains dynamic aging, as in PFP-DA, but with the additional parameter introduced.

As we would expect, the numeric distance between the top-ranked item and subsequently ranked items is reduced as β is reduced. To obtain a more intuitive feel for this behavior, let us consider an example. If we consider the regular PFP-DA case, with β = 1, the top-ranked item from a face will contribute 1/1 = 1, the second-ranked item will contribute 1/2 = 0.50, and the third-ranked item will contribute 1/3 ≈ 0.33. So, the respective distances are shown in Table 1.

Table 1. Numeric distance between items and top item, β = 1.

Rank	Numeric Distance from Top Item
1	0
2	0.5
3	0.67
4	0.75
5	0.80

The distance is lessened if β is set to 0.5, or 0.1, as shown in Table 2.

Table 2. Numeric distance between items and top item, (a) β = 0.5 and (b) β = 0.1.

	Rank	Numeric Distance from Top Item		Rank	Numeric Distance from Top Item
(a)	1	0	(b)	1	0
	2	0.29		2	0.07
	3	0.42		3	0.10
	4	0.50		4	0.13
	5	0.55		5	0.15

As β is reduced, thus yielding a small distance between each item and the top item, as seen in the above tables, the overall contributions for the ranked items themselves are larger (e.g., for β = 1, the 4th item contributes 0.25; for β = 0.5, the contribution for the fourth item is 0.50; and for β = 0.1, the contribution is 0.87). What this means is that the popular items are all contributing more, and the difference between a fourth or a fifth item is not as significant for a β = 0.1 as it would be for items ranked with β = 1. Therefore, with these larger numbers, it stands to reason that a PFP scheme using a small β will favor items to be cached that have contributions from multiple faces even more than a PFP scheme using a larger β.

We will see that this holds true as we discuss the experimental results later in this article. However, we will also see that a small β value in a PFP scheme also works best

under certain circumstances and far worse in others, involving situations with a higher percentage of attackers.

4. Methodology

The PFP-DA and PFP-β schemes were implemented using ndnSIM v. 2.5 [21,22], which is an NDN simulator based on ns-3 [23]. The ns-3 simulator is well known for its use with both traditional and experimental networks. Its code is entirely open source, which makes it appealing, as the simulator itself, not just scenarios, can be modified.

For our hardware setup, we used a System76 Gazelle with a quad-core 6 MB cache, 16 GB of DDR4 RAM, and a 3.5 GHz i7-6700HQ CPU. The operating system on this machine is Ubuntu 16.04 LTS. The point-to-point data rate was set to 50 Mbps, and the point-to-point channel delay was set to 10 ms. Note that bandwidth usage was not considered a metric of interest for this study.

Regular consumers make requests following Zipf–Mandelbrot distribution, as implemented in ndnSIM. Attackers make uniform requests for what are considered (legitimately) less popular items (according to the Zipf–Mandelbrot distribution).

For our tests, we will provide scenarios to test three primary variables in order to see how they affect the general cache hit percentage, the top 100 hit percentage, and the top 50 hit percentage. The three primary variables are the cache size, attacker request frequency, and number of attackers.

First, the cache size variations always have an attacker with a request frequency of 720 requests/second (req/s), while regular consumers are 120 req/s. The number of content objects requested by the attacker is equal to the size of the cache in each scenario, so it is hypothetically possible for the attacker to fill the cache with unpopular items.

Second, the attacker's request frequency is varied. A single attacker will make requests varying from 360 req/s up to 1200 req/s, with regular consumers requesting 120/req/s. The attacker starts at a faster rate than a normal consumer in order to give him a better chance of overwhelming the caching algorithm and providing more useful data.

Third, the number of attackers ranges from zero attackers (for a baseline, attacker-free scenario) to one attacker, three attackers, five attackers, and seven attackers, all out of a total of ten consumers. This is to test how the algorithms respond, with respect to hit percentages and pollution, to an increasing number of attackers in the environment. The attackers request at 720 req/s, with normal consumers requesting at 120 req/s, again, to provide simulation of aggressive attackers attempting to overwhelm the caching algorithms.

4.1. Per-Face Popularity with β-Parameterization

The typical workflow with ndnSIM involves creating scenarios, which are implemented in typical C++ source files. The WAF build automation utility [24] is then used to build and run the scenarios and support files. The scenarios are also often accompanied by the use of extensions, which are simply modifications (usually using inheritance from base classes of the simulator) or additional classes needed for the scenarios to function. The scenario file itself is where the entry point for the application lives.

One extension that we created was based on the Policy class. The Policy class in ndnSIM is the actual decision maker as far as what items remain and what items are removed from the content store's cache. We created a PopularityPolicy class as a subclass of Policy to define our cache replacement policy. In ndnSIM, data are automatically added to the cache, which is physically slightly bigger than the logical limit of the cache size. In Policy's doAfterInsert method, a pure virtual function (method) is called immediately after a content object is added to the cache. Then, in this method, the decision can be made to remove any content object according to the cache replacement algorithm.

The PopularityPolicy class that we created maintains a pointer to the popularity manager (PopMan), which is also maintained in the ndnSIM's NFD Forwarder class, which we modified. Therefore, the PopMan acts as an essential bridge from the forwarding strategy and fabric to the cache replacement policy. The PopMan records when interests

arrive and over which face, so that ranking can be accomplished. This interest and face data are ultimately maintained in FaceContentFrequency objects, which maintain, as the name suggests, the frequency of requests for content organized by specific faces. Each FaceContentFrequency object maintains all content requests and a ranked list for its face. This is the class responsible for determining an individual face's contribution to overall popularity. The PopMan, in turn, utilizes the contributions from its list of FaceContentFrequency objects to determine what should be removed when it must decide on the next item to remove from the cache.

So, to summarize, the Policy (PopularityPolicy), in its doAfterInsert method, determines when the cache is full after an insert of a content object obtained from a returned data message. If the cache is full, the PopularityPolicy asks the popularity manager what content object must be removed from the cache. As described earlier, this decision is based on the overall contributions from all faces, normalized such that no single face has more influence than any other face. When this information is obtained, the PopularityPolicy sends an eviction signal to the content store, which evicts the corresponding object from the cache.

4.2. Topologies

We use three topologies in total for the data we have collected. One topology is called the Simple Topology (ST). Another is the Advanced Topology (AT). And finally, for testing PFP- β, we used a slightly larger version of the ST called LST, or Larger Simple Topology.

Although Figure 1 implies a single attacker (the red node), the number of attackers increased during some tests to determine the effect of increasing percentages of attackers. The Large Simple Topology used with the PFP-β is the same as ST, but with 10 additional consumers, yielding a total of 20 consumers.

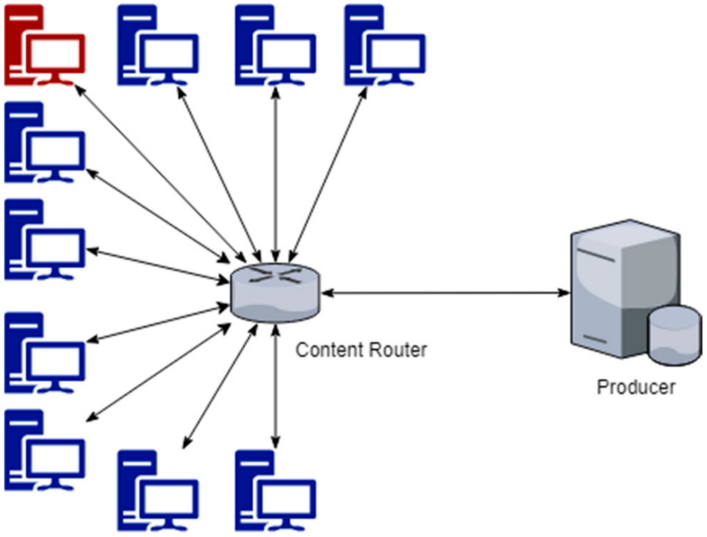

Figure 1. Simple Topology (ST) with 10 total consumers, 1 router, and 1 producer.

The Advanced Topology (AT) seen in Figure 2 is like Xie's merging scenario in [12]. This topology was used to discover the effects of the different cache replacement schemes on the less busy edge routers (e.g., R2, R4) and intermediate routers (R3, R0). However, the focus remains on R1, as it has the most attached consumers, is not on the path from any other router to the producer node, and is thus not affected in the same way by interest aggregation and other phenomena inherent to NDN.

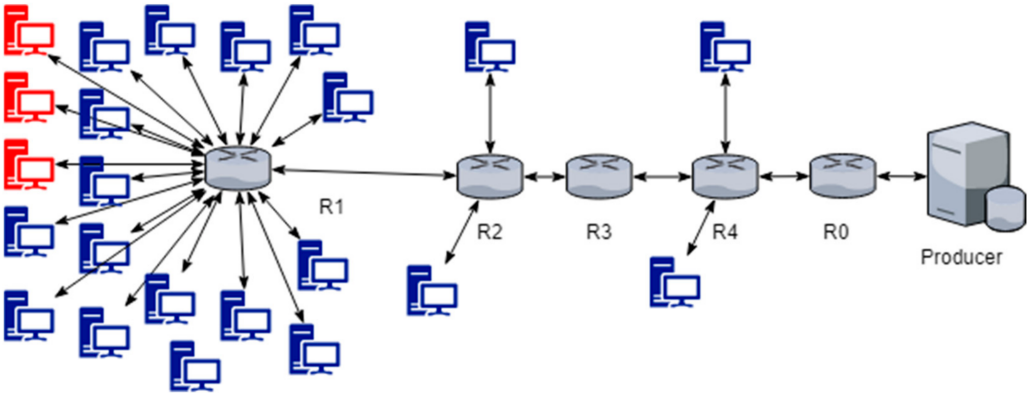

Figure 2. Advanced Topology (AT) with five routers, twenty-three consumers, and a producer.

5. Results

Firstly, we will discuss most of our results, which are derived from tests using the standard PFP-DA scheme. Then, we will discuss some of the early results we've obtained from the PFP-β experiments. We will demonstrate that our PFP-based schemes tend to perform substantially better than the current LFU-DA and LRU algorithms under most circumstances. The content space used in these tests consists of 10,000 content objects, using the Zipf–Mandelbrot distribution as implemented in ndnSIM. Unless otherwise noted, the following sections involve the PFP-DA scheme.

It should be noted that regular consumers will generally follow the Zipf–Mandelbrot distribution of requests across all faces. In other words, the top requested content object would almost certainly be the top requested content object on each of the faces. It is possible, however, that once the Zipf–Mandelbrot rankings are lower, one face may request one item more than another that has a "higher ranking" based on the probabilities implemented in ndnSIM. In other words, item ranked 150 might be requested on one face more times than item 148, and vice versa. The distribution is not truly strict as implemented.

Attackers are expected to have some degree of coordination when spread across multiple faces, and in fact, we have implemented the scenarios so the attackers request in a generally uniform fashion across a small set of less popular items (with the Zipf–Mandelbrot distribution).

5.1. Varying Cache Size

For these tests, one attacker is present for both the simple and advanced topologies, with a request frequency of 720 requests/second. Regular consumers request 120 requests/second. The reason for these frequencies was to give the attacker an overwhelming advantage as an individual item requester. For these tests in this section, only the cache size was varied, ranging from 100 items (1% of the total content space) to 500 items (5% of the total content space). The content space consists of all the items that can be cached within the domain of discourse. The top 100 and top 50, respectively, refer to the percentage of how many of the expected *actually popular* items that should be in the cache if there was no attack being perpetrated.

The number of content objects requested by an attacker is equal to the cache size, so it is possible for the attacker to attempt to completely overwhelm the cache. For example, it is clear that if all the consumers attached to a router were attackers, then the cache would be entirely filled with only attacker content. The tests in this section and subsequent sections are to test variations and determine the robustness of the cache replacement algorithms.

The data in Table 3 give us an interesting picture of how the cache size (measured in the number of packets that the cache can accommodate) can affect both pollution and hit rates. The pollution percentage indicates the percentage of the cache at the router occupied by attacker content. The hit percentage (hit rate) is the percentage of legitimate consumer interests that are able to be satisfied by the content store's cache upon arrival. We also look

at the top 100 and top 50 items (according to Zipf-like distribution) to see how stable the algorithms are at maintaining the truly popular contents in the cache.

Table 3. ST with varying cache sizes and single attacker at frequency of 720.

Algorithm	Cache Size (# of Packets)	Cache %	Pollution %	Hit %	Top 100	Top 50
PFP-DA	100	1%	15.00%	19.02%	62.80%	75.53%
LFU-DA	100	1%	36.00%	12.67%	41.17%	50.00%
LRU	100	1%	36.00%	8.82%	28.72%	34.64%
PFP-DA	200	2%	14.00%	25.57%	80.02%	91.53%
LFU-DA	200	2%	39.00%	17.08%	52.83%	62.73%
LRU	200	2%	37.00%	14.66%	46.04%	54.76%
PFP-DA	300	3%	12.67%	29.32%	87.76%	97.96%
LFU-DA	300	3%	42.00%	20.53%	62.67%	72.19%
LRU	300	3%	38.00%	18.58%	56.46%	66.13%
PFP-DA	500	5%	12.40%	34.89%	93.65%	98.96%
LFU-DA	500	5%	39.40%	25.84%	73.58%	81.03%
LRU	500	5%	39.00%	24.53%	70.51%	80.25%

For all cache sizes, PFP-DA has both the lowest cache pollution percentages and the highest hit rates. For example, even considering the smallest cache size (100, or 1%), LFU-DA does reasonably well, with an overall hit rate of 12.67%. But this is over 6% less than PFP-DA. The percentages for the top 100 and top 50 objects are even more remarkable. LFU-DA, for example, maintains 50% to 81.03%, which is good, but PFP-DA maintains percentages from around 75% to nearly 99% for the top 50 items. This shows overall excellent performance and, in specific, good resistance to the single attacker.

Now, let us consider the Advanced Topology (AT) results.

Both Tables 4 and 5 show the data collected and summarized for the Advanced Topology (AT). The R1 router is the router with most of the consumers attached and with the single attacker attached as well. The routers R0, R2, R3, and R4 all demonstrate very low hit percentages and sometimes higher cache pollution.

Table 4. AT with varying cache sizes, single attacker, and 720 frequency—pollution %.

			Pollution % by Router				
Algorithm	**Cache Size**	**Cache %**	**R0**	**R1**	**R2**	**R3**	**R4**
PFP-DA	100	1%	18.00%	8.00%	23.00%	16.00%	8.00%
LFU-DA	100	1%	17.00%	22.00%	17.00%	22.00%	14.00%
LRU	100	1%	18.00%	18.00%	17.00%	19.00%	17.00%
PFP-DA	200	2%	22.00%	7.00%	24.00%	23.00%	7.50%
LFU-DA	200	2%	21.50%	23.50%	21.50%	21.50%	19.50%
LRU	200	2%	17.00%	22.00%	19.00%	22.00%	16.50%
PFP-DA	300	3%	22.33%	7.00%	14.67%	21.67%	8.67%
LFU-DA	300	3%	22.33%	22.33%	21.67%	22.67%	22.00%
LRU	300	3%	17.33%	22.33%	19.67%	19.67%	17.67%
PFP-DA	500	5%	23.40%	6.60%	12.40%	19.20%	9.20%
LFU-DA	500	5%	23.00%	24.60%	22.80%	23.40%	23.20%
LRU	500	5%	19.80%	25.00%	22.80%	22.20%	20.60%

An interesting discovery is that the overall cache pollution percentage is not necessarily an indicator of the hit rate. For example, on R2, PFP-DA maintains higher cache pollution percentages than LFU-DA or LRU, but the hit rates are still better for PFP-DA than the other algorithms. This is because the cache pollution percentage does not capture information about the quality of items being cached (i.e., what rank in the Zipf distribution they have). However, in terms of hit rate, PFP-DA dominates all hit rate metrics across all routers. Again, we see the top item hit rates on R1 in the 70% to nearly 100% range for PFP-DA.

Table 5. AT with varying cache sizes, single attacker, and 720 frequency—hit %.

		Hit %					R1 Special Hit %	
Algorithm	**Cache Size**	**R0**	**R1**	**R2**	**R3**	**R4**	**Top 100**	**Top 50**
PFP-DA	100	0.55%	16.78%	4.97%	1.56%	2.52%	67.17%	81.07%
LFU-DA	100	0.01%	10.78%	2.19%	0.58%	2.35%	42.61%	53.34%
LRU	100	0.00%	6.89%	1.20%	0.05%	1.19%	26.05%	31.98%
PFP-DA	200	0.74%	21.98%	6.07%	1.48%	4.05%	81.55%	93.42%
LFU-DA	200	0.04%	15.36%	3.57%	0.52%	3.23%	59.16%	71.41%
LRU	200	0.00%	11.94%	2.13%	0.12%	1.86%	44.07%	53.03%
PFP-DA	300	0.79%	25.40%	7.64%	1.57%	5.20%	89.16%	97.69%
LFU-DA	300	0.01%	18.10%	4.23%	0.51%	3.92%	65.42%	77.52%
LRU	300	0.00%	15.54%	2.82%	0.15%	2.52%	54.99%	65.08%
PFP-DA	500	0.88%	30.93%	10.83%	2.26%	6.33%	96.15%	99.62%
LFU-DA	500	0.01%	23.36%	5.65%	0.83%	5.01%	78.37%	88.69%
LRU	500	0.00%	21.57%	3.94%	0.18%	3.61%	70.49%	80.84%

5.2. Varying Attacker Request Frequencies (Single Attacker)

For these tests, one attacker is still present, but the attacker's aggression has increased. Consumers still have a request frequency of 120 requests per second, while attackers' speeds (request frequency) vary from 360 up to 1200. That is to say, the speed varies from three times to ten times that of the normal consumer.

With the Simple Topology (ST), we observe in Table 6 that the pollution percentages of the cache running PFP-DA remain essentially unaffected by the change in attacker request frequency. This is expected behavior, since the contributions from the faces are normalized. Thus, a single attacker (or, more correctly, a single face being used as a conduit of attack or realized attack vector), no matter how aggressive, cannot have any extreme effect on the cache other than what they would normally have if they were minimally or not at all aggressive. The story is different for LFU-DA and LRU. While the percentage differences are not enormous, it is clear that both algorithms are susceptible to aggressive attacks.

Table 6. ST with size 100 cache, single attacker, and varying request frequency of attacker.

Algorithm	**Attacker Speed**	**Pollution %**	**Hit %**	**Top 100**	**Top 50**
PFP-DA	360	16.00%	19.04%	62.85%	75.60%
LFU-DA	360	22.00%	13.36%	42.15%	50.74%
LRU	360	24.00%	10.13%	32.74%	39.34%
PFP-DA	720	15.00%	19.02%	62.80%	75.53%
LFU-DA	720	36.00%	12.67%	41.17%	50.00%
LRU	720	36.00%	8.82%	28.72%	34.64%
PFP-DA	960	15.00%	19.03%	62.83%	75.56%
LFU-DA	960	41.00%	11.67%	38.11%	45.92%
LRU	960	39.00%	8.23%	26.85%	32.41%
PFP-DA	1200	15.00%	19.03%	62.84%	75.58%
LFU-DA	1200	44.00%	11.65%	38.41%	46.39%
LRU	1200	45.00%	7.77%	25.43%	30.71%

Figures 3–5 all graphically demonstrate the data recorded in Table 6. While the LFU-DA and LRU are affected by the increased attacker aggression, the PFP-DA remains virtually unaffected. This is clearest when considering the pollution percentage in Figure 3. While PFP-DA maintains a low, steady pollution percentage, both LFU-DA and LRU both increase significantly. And, while LFU-DA does maintain a reasonably good hit rate throughout, the percentage does drop, and interestingly, it starts out substantially lower than with PFP-DA.

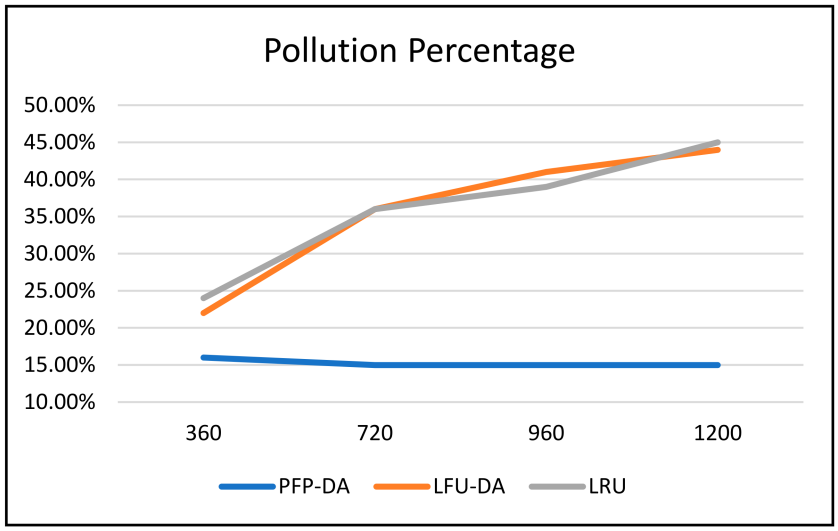

Figure 3. ST with 100 cache size, single attacker, and varying attacker request frequency—pollution %.

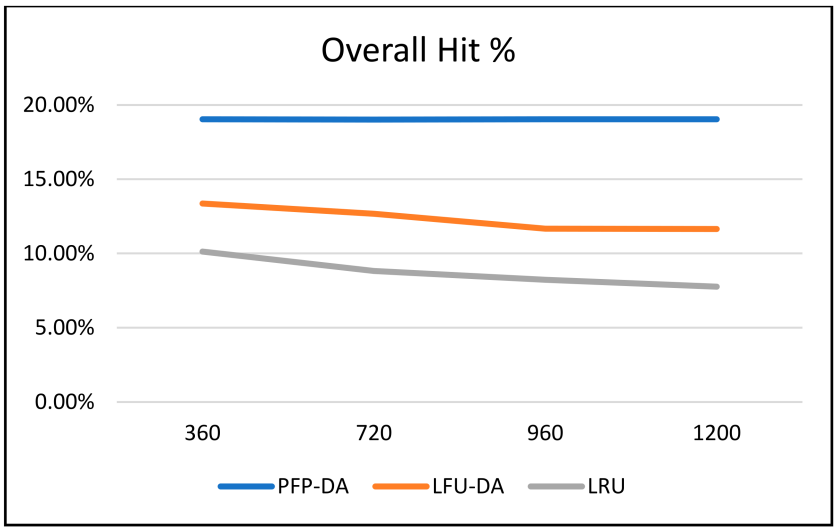

Figure 4. ST with 100 cache size, single attacker, and varying attacker request frequency—overall hit %.

Let us consider the Advanced Topology (AT) test results with a single attacker and varying request frequencies.

Again, with the Advanced Topology in Tables 7 and 8, we see that PFP-DA dominates the other two algorithms, especially in terms of hit rate. For routers R0, R2, R3, and R4, the hit rates are quite low, as before. This is largely due to them not being connected directly to any, or in the cases of R2 and R4, not many consumers. From a network design perspective, the routers with very low hit rates in such simulations might be good candidates to be regular non-storage routers since their contributions are mostly negligible for all three algorithms.

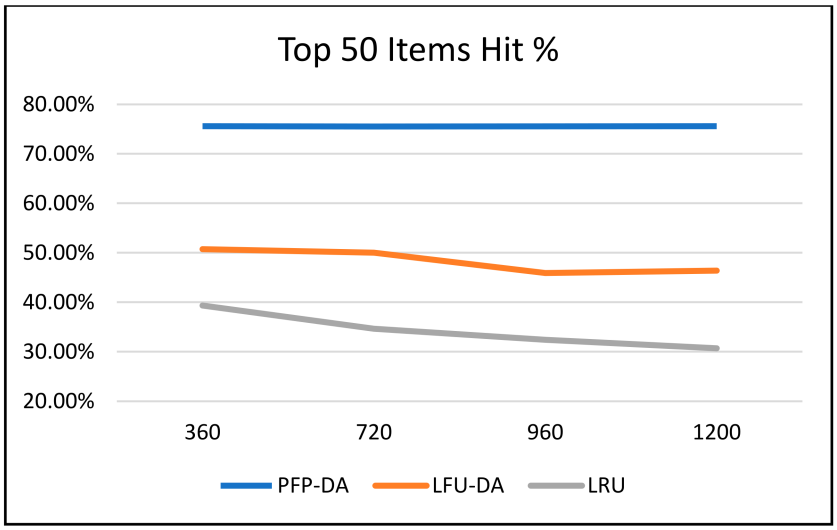

Figure 5. ST with 100 cache size, single attacker, and varying attacker request frequency—top 50 items hit %.

Table 7. AT with size 100 cache, single attacker, varying request frequency—pollution %.

		Pollution %				
Algorithm	**Attacker Speed**	**R0**	**R1**	**R2**	**R3**	**R4**
PFP-DA	360	11.00%	8.00%	11.00%	11.00%	3.00%
LFU-DA	360	11.00%	12.00%	11.00%	12.00%	11.00%
LRU	360	11.00%	14.00%	14.00%	12.00%	10.00%
PFP-DA	720	18.00%	8.00%	23.00%	16.00%	8.00%
LFU-DA	720	17.00%	22.00%	17.00%	22.00%	14.00%
LRU	720	18.00%	18.00%	17.00%	19.00%	17.00%
PFP-DA	960	21.00%	8.00%	13.00%	18.00%	7.00%
LFU-DA	960	22.00%	27.00%	25.00%	25.00%	19.00%
LRU	960	15.00%	28.00%	19.00%	16.00%	16.00%
PFP-DA	1200	26.00%	8.00%	26.00%	24.00%	7.00%
LFU-DA	1200	27.00%	29.00%	25.00%	27.00%	23.00%
LRU	1200	16.00%	11.64%	20.00%	19.00%	14.00%

Table 8. AT with size 100 cache, single attacker, and varying request frequency—hit %.

		Hit %					R1 Special Hit %	
Algorithm	**Attacker Speed**	**R0**	**R1**	**R2**	**R3**	**R4**	**Top 100**	**Top 50**
PFP-DA	360	0.63%	16.60%	5.22%	1.58%	2.57%	66.75%	80.49%
LFU-DA	360	0.04%	12.58%	2.75%	0.38%	2.18%	50.25%	62.00%
LRU	360	0.00%	7.46%	1.23%	0.06%	1.23%	28.18%	34.55%
PFP-DA	720	0.55%	16.78%	4.97%	1.56%	2.52%	67.17%	81.07%
LFU-DA	720	0.01%	10.78%	2.19%	0.58%	2.35%	42.61%	53.34%
LRU	720	0.00%	6.89%	1.20%	0.05%	1.19%	26.05%	31.98%
PFP-DA	960	0.58%	16.84%	4.98%	1.51%	2.44%	67.18%	81.02%
LFU-DA	960	0.02%	10.80%	2.01%	0.29%	1.82%	42.54%	52.83%
LRU	960	0.00%	6.58%	1.20%	0.06%	1.15%	24.94%	30.67%
PFP-DA	1200	0.58%	16.87%	4.88%	1.49%	2.48%	66.98%	80.65%
LFU-DA	1200	0.15%	11.64%	2.45%	0.33%	2.32%	46.22%	58.57%
LRU	1200	0.00%	6.41%	1.18%	0.06%	1.14%	24.36%	29.95%

We once again turn our attention to router R1. Running PFP-DA, R1 maintains a 16.6% to 16.9% hit rate throughout all attack frequencies. LFU-DA also appears to be reasonably resistant to attack, even increasing the overall hit rate at 1200 attack frequencies. This seems odd, but it largely stems from natural fluctuations between launch scenarios.

There remains no question, however, as to which algorithm performs best. PFP-DA maintains the highest percentage of overall hit rates. Considering the top 50 items, PFP-DA has anywhere from a 20 to 30% better hit rate than LFU-DA.

5.3. Varying Number of Attackers

To further understand how these three algorithms work under different conditions, we turn our attention to varying the number of attackers (or, more correctly, the number of faces under attack that are dominated by the attackers). In the scenarios used for these tests, the cache size is 100 (1%), the attackers' request frequency is 720 requests/second, and the regular consumers request 120 requests/second. Arguably, it is reasonably difficult for attackers to overwhelm many faces on a content storage router, but we wanted to test out the algorithms' performance under stressful conditions. If any attack variation has a significant impact on PFP-DA, it will affect a larger percentage of attackers. PFP-DA would not benefit from normalizing contributions directly, as it did with the single attacker in varying frequency scenarios.

The data in Table 9 for the Simple Topology give us insight into the typical performance of algorithms, including when there are no attacks occurring. The zero-attacker scenario gives a baseline for each of the caching algorithms under consideration. For ST, since there are ten total consumers, one attacker is 10%, three attackers are 30%, and so on. We tested variations from zero attackers all the way up to seven attackers, which are 70%.

Table 9. ST with size 100 cache, 720 attack frequency, and varying number of attackers.

Algorithm	# Attackers	% Attackers	Pollution %	Hit %	Top 100	Top 50
PFP-DA	0	0.00%	0.00%	20.69%	69.67%	82.96%
LFU-DA	0	0.00%	0.00%	14.82%	47.55%	55.59%
LRU	0	0.00%	0.00%	12.17%	38.59%	46.40%
PFP-DA	1	10.00%	15.00%	19.02%	62.80%	75.53%
LFU-DA	1	10.00%	36.00%	12.67%	41.17%	50.00%
LRU	1	10.00%	36.00%	8.82%	28.72%	34.64%
PFP-DA	3	30.00%	41.00%	16.29%	54.60%	66.76%
LFU-DA	3	30.00%	65.00%	9.16%	29.78%	35.31%
LRU	3	30.00%	64.00%	5.86%	19.34%	23.50%
PFP-DA	5	50.00%	68.00%	12.67%	43.89%	53.83%
LFU-DA	5	50.00%	74.00%	3.17%	9.62%	9.77%
LRU	5	50.00%	74.00%	3.86%	12.95%	15.82%
PFP-DA	7	70.00%	83.00%	7.97%	27.88%	34.80%
LFU-DA	7	70.00%	85.00%	0.70%	1.83%	1.67%
LRU	7	70.00%	88.00%	2.07%	6.91%	8.57%

In Figure 6, the pollution percentages for LFU-DA and LRU remain very similar, with PFP-DA outperforming both from about the 10% up to the seven-attacker (70%) scenarios. At around the higher percentage of attackers, the gap closes substantially as PFP-DA is overcome with cache pollution.

Important to note, however, is that the quality of content remaining in the cache is not captured with the overall cache pollution percentage data. The hit rates give us a much better picture of this in Figures 7 and 8. Both the overall and top 50 item hit rates take on very similar shapes, graphically. LRU, largely by coincidence, overtakes LFU-DA around the 50% attacker mark. This is largely because LFU-DA is based on request frequency, while LRU is not. Therefore, LRU depends entirely on recency and is affected only due to attacker content being more recently accessed as the number of attackers increases. Thus,

the LRU data are arguably affected indirectly, whereas LFU-DA and PFP-DA are primarily affected by frequency, which means they are directly affected by incoming frequencies.

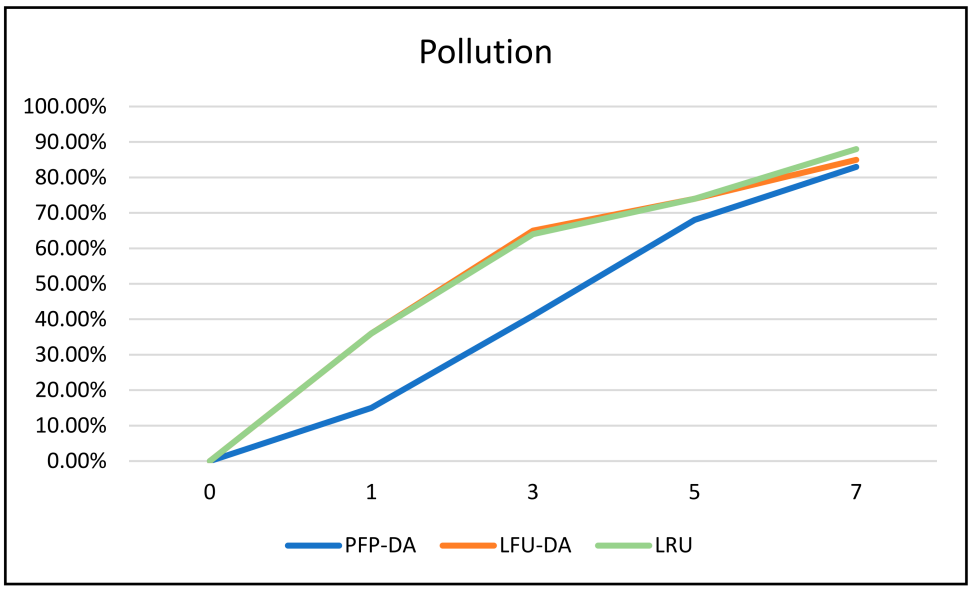

Figure 6. ST with 100 cache size, 720 attacker frequency, and varying number of attackers—pollution %.

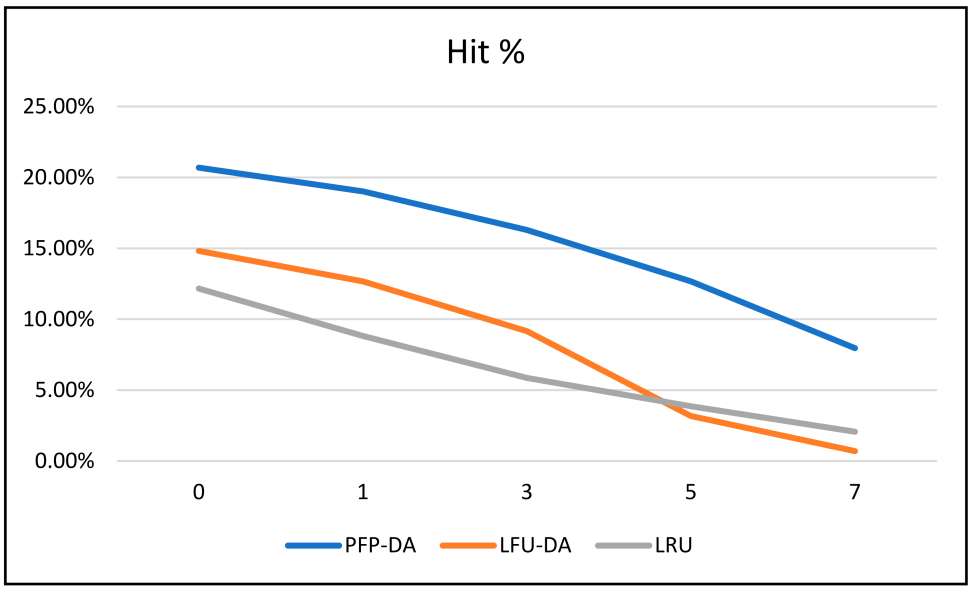

Figure 7. ST with 100 cache size, 720 attacker frequency, and varying number of attackers—hit %.

The difference between PFP-DA and both LRU and LFU-DA is remarkable. Even as the number of attackers increases, and PFP-DA is affected, it still performs substantially better than LRU and LFU-DA. We note that PFP-DA has a better hit rate (overall) by 5–9% during the increase in attacks. Even more interesting is that PFP-DA still maintains an almost 35% hit rate for the top 50 items, even as LFU-DA is entirely overwhelmed.

Perhaps the most remarkable scenario is the 0% (no attacker) scenario. This clearly shows PFP-DA is superior to both LFU-DA and LRU by about a 6% better overall hit rate and around 30% for the top 50 items. This demonstrates that PFP-DA is the best cache replacement algorithm overall, even during periods where no attack is being launched. The explanation for this is that because of the normalization across the multiple faces, the items that remain in the cache give a truer and fair indication of what is popular, not just

popular across one or two faces with higher frequencies of request. Depending on the traffic arriving at a given router, the popularity of the content arriving at each of the faces is not necessarily homogenous and, in fact, might vary significantly based on the origin of the requests.

Clearly, PFP-DA outperforms LFU-DA and LRU for the top content objects, maintaining a very strong hit rate until around item 100, where the results are less clear. Our previous results show that PFP-DA performs better with maintaining the most popular items in the cache, and this figure reinforces that PFP-DA is better, even with no active attack.

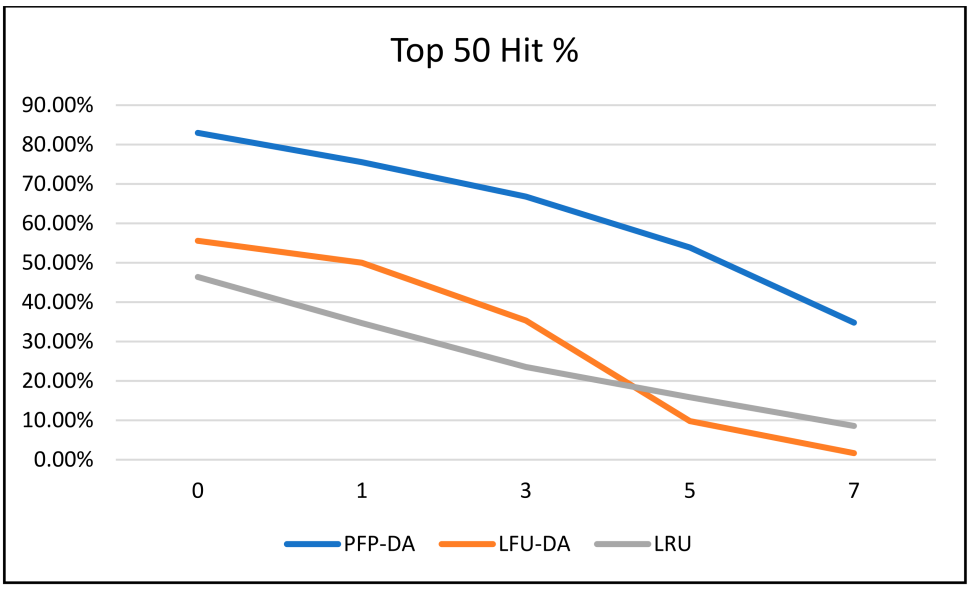

Figure 8. ST with 100 cache size, 720 attacker frequency, and varying number of attackers—top 50 items hit %.

5.4. PFP-β Results

As discussed earlier, PFP-β (PFP-Beta) takes a β parameter, which acts as the exponent to the denominator for each of the contributions from faces during the overall popularity calculation. Equation (4) is repeated below and expanded to demonstrate the use of β.

$$P(C,\ \beta) = \sum_{\substack{i=1 \\ 0 \le \beta \le 1}}^{n} \frac{1}{{r_i}^{\beta}} = \frac{1}{{r_1}^{\beta}} + \frac{1}{{r_2}^{\beta}} + \ldots + \frac{1}{{r_{n-1}}^{\beta}} + \frac{1}{{r_n}^{\beta}}$$

The lower the β, the smaller the numeric difference between the per-face contribution for each item and the top item. This causes content objects that have interests received from multiple faces to be substantially favored. The mechanism for this behavior is described in more detail in Section 3.3.

The motivation for this variation on PFP-DA is to see if we can further lower cache pollution and increase the hit rate. Through our initial tests, we have made some very interesting discoveries.

Firstly, we can observe the effects (or lack thereof) of varying the attack speed of the single attacker. The pollution is 0%, which is impressive. The hit rates are substantially better than the single attacker scenarios we tested for PFP-DA (see Table 6). Of course, with PFP-DA, the regular Simple Topology was used, with 10 nodes attacking the router, whereas our PFP- β was tested with the Larger Simple Topology (LST), which has 20. However, we will find that even with two attackers (10%) against the LST, considering the coordination and the consistency of the percentage, the pollution will remain at 0%, with almost identical hit rates.

Therefore, as shown in Table 10, we can conclude that if we consider the PFP-DA tests from earlier to be a special case of PFP- β where β = 1.0, that the smaller β = 0.05 for essentially the same variations performs approximately 16% better in terms of cache pollution, approximately 3.7% better for overall hit rate, about 25% better for the top 100 items (with Zipf-like distribution), and about 23% better for the top 50 items (98.83% vs. 75.53%, for a rate of 720). Astonishingly, the top 50 items experience just under a 99% hit rate with our β = 0.05. Although 75.53% was incredibly good and totally outperformed LFU-DA (50% top 50 hit rate), our β = 0.05 tests far surpass our previous PFP-DA tests in terms of small attacker percentage with varying attacker frequency.

Table 10. PFP- β with single attacker, cache size of 100, beta of 0.05, and varying attack frequency.

Attacker Speed	Pollution %	Hit %	Top 100	Top 50
720	0.00%	22.76%	87.98%	98.83%
840	0.00%	22.81%	87.97%	98.83%
960	0.00%	22.84%	87.98%	98.83%

Secondly, we tested variations of the β value, keeping the number of attackers at 2 with attack frequencies of 720 each and a cache size of 100. As we see in Table 11, increasing the β value increases the pollution percentage and decreases the hit rates in these scenarios. *Ceteris paribus*, as β moves toward the regular PFP-DA (β = 1.0), the performance drops.

Table 11. Two attackers, frequency of 720, cache size of 100, and varying beta values.

Beta	Pollution %	Hit %	Top 100	Top 50
0.10	0.00%	22.55%	89.56%	97.91%
0.30	3.00%	22.09%	87.89%	97.79%
0.50	10.00%	20.51%	81.84%	94.81%
0.70	13.00%	19.13%	76.34%	90.23%

Thirdly, we held the number of attackers at 2 with attack frequencies of 720 each, a constant β = 0.1, and varied the cache size. As more spaces become available, the pollution percentage increases extremely slightly, see Table 12. However, it is negligible, even for a cache size of 500 (5% of the content space), as only a few items, accounting for 0.4% of the entire cache, can make it into the content store's cache.

Table 12. Two attackers, frequency of 720, β value of 0.1, and varying cache size.

Cache Size	Pollution %	Hit %	Top 100	Top 50
200	0.00%	29.65%	99.05%	99.69%
300	0.33%	33.98%	99.53%	99.73%
500	0.40%	40.49%	99.60%	99.76%

The overall hit rate increases significantly, as would be expected since more items can reside in the cache overall. The top 100 and top 50 hit rates increase slightly, but they are so close to 100% already that there is little room for improvement.

In Table 13, we see where PFP-β falls short. Although the performance of PFP-β is better than that of PFP-DA for small percentages of attackers, PFP-β performs worse than PFP-DA when the number of attackers increases. In other words, it appears that lower β values are resistant to cache pollution under a small percentage of attackers (less than 15%, for example), but higher β values (such as the original PFP-DA, technically holding β = 1.0) allow for more resistance against larger percentages of attackers.

Table 13. Attacker frequency of 720, β value of 0.1, cache size of 100, and varying number of attackers.

# Attackers	% Attacker	Pollution %	Hit %	Top 100	Top 50
1	5%	0.00%	22.76%	87.98%	98.83%
2	10%	0.00%	22.55%	89.60%	97.96%
3	15%	8.00%	21.00%	81.99%	95.86%
4	20%	26.00%	19.25%	81.59%	95.45%
5	25%	52.00%	15.43%	68.14%	87.85%
6	30%	63.00%	13.10%	60.38%	80.39%

6. Summary, Conclusions, and Future Work

Throughout this work, we have explored the effects of normalizing contributions toward caching influence across faces, yielding our Per-Face Popularity, or PFP schemes. The original PFP scheme showed promise but was not scalable due to the overly large (unbounded) lists that needed to be maintained of content object rankings for each face. To mitigate this issue and to also add a sense of recency to our PFP scheme, we developed PFP-DA. PFP-DA employs a dynamic aging characteristic to ensure that content items that were ranked for a given face do not remain there longer than they should due to their extreme popularity at one time. Many of our tests involved PFP-DA. The results clearly demonstrate that PFP-DA performs better than LFU-DA and LRU in nearly all circumstances that were tested. This was especially true in terms of the ability of PFP-DA to maintain the most popular items in the cache, thus yielding much higher hit rates.

To further explore the PFP-DA scheme, a β parameter was added, which, when sufficiently small, reduces the numeric distance between the rank contribution of the top-ranked item and any subsequent item. We can consider our regular PFP-DA as a special case of PFP-β with β = 1.0. The PFP-β tests yielded interesting results. For smaller values of β, the cache pollution tended to be less (even 0%), and hit rates tended to be greater (nearly 100% for the top 100 and top 50 items), but only if the percentage of attackers was relatively small.

In conclusion, we have created a new approach to increasing cache robustness by means of normalizing influence on caching decisions using a per-face fairness technique called Per-Face Popularity, or PFP. We have improved the original algorithm substantially and created many additional opportunities for future research as well.

For our future research endeavors related to this work, we will explore the applicability of our PFP techniques to current caching technologies, such as those used by CDNs. Even though our original work is geared toward content-centric approaches, specifically NDN, it is possible that our research can yield more immediate benefit to the industry, even if used in application overlays.

Further comparisons with other caching algorithms, such as the Name Popularity Algorithm [25] and CaDaCa [26], are needed. It is possible that the PFP-based algorithms could enhance or be enhanced with these additional caching techniques in a hybrid strategy.

Furthermore, we will continue to refine our techniques and attempt to arrive at precise mathematical or statistical models to better predict and describe the behavior of our schemes.

Author Contributions: Conceptualization, J.B. and J.G.; Coding, J.B.; Supervision, J.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Zip files of the code and raw data are available at https://github.com/profjpbaugh/jpb-jg-pfp-2023.git (accessed on 27 July 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CCN	Content-Centric Networking
CDN	Content Distribution (or Delivery) Networks
ICN	Information-Centric Networking
LFU-DA	Least Frequently Used with Dynamic Aging
LRU	Least Recently Used
NDN	Named Data Networking
PFP	Per-Face Popularity
PFP-DA	Per-Face Popularity with Dynamic Aging
PFP-β	Per-Face Popularity with Beta Parameterization

References

1. Ahlgren, B.; Dannewitz, C.; Imbrenda, C.; Kutscher, D.; Ohlman, B. A Survey of Information-Centric Networking. *IEEE Commun. Mag.* **2012**, *50*, 26–36. [CrossRef]
2. Xylomenos, G.; Ververids, C.N.; Siris, V.A.; Fotiou, N.; Tsilopoulos, C.; Vasilakos, X.; Katsaros, K.V.; Polyzos, G.C. A Survey of Information-Centric Networking Research. *Commun. Surv. Tutor.* **2014**, *16*, 1024–1049. [CrossRef]
3. Somayeh Kianpisheh, T.T. A Survey on In-Network Computing: Programmable Data Plane and Technology Specific Applications. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 701–761. [CrossRef]
4. Koponen, T.; Chawla, M.; Chun, B.-G.; Ermolinskiy, A.; Kim, K.H.; Shenker, S.; Stoica, I. A Data-Oriented (and Beyond) Network Architecture. In Proceedings of the SIGCOMM '07, Kyoto, Japan, 27–31 August 2007.
5. Ahlgren, B.; Marchisio, M.; D'Ambrosio, M.; Marsh, I.; Dannewitz, C.; Ohlman, B.; Pentikousis, K.; Strandberg, O.; Remarz, R.; Vercellone, V. Design Considerations for a Network of Information. In Proceedings of the ACM ReArch '08, Madrid, Spain, 9–12 December 2008.
6. D'Ambrosio, M.; Dannewitz, C.; Karl, H. MDHT: A hierarchical name resolution service for information-centric networks. In Proceedings of the ACM SIGCOMM, Toronto, ON, Canada, 19 August 2011.
7. Content Centric Networking Project. Available online: www.ccnx.org (accessed on 23 July 2023).
8. Jacobson, V.; Smetters, D.; Thornton, J.; Plass, M.; Briggs, N.; Braynard, R. Networking Named Content. In Proceedings of the ACM CoNEXT '09, Rome, Italy, 1–4 December 2009.
9. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Claffy, K.C.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named Data Networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. [CrossRef]
10. Nygren, E.; Sitaraman, R.; Sun, J. The Akamai Network: A Platform for High-Performance Internet Applications. *ACM SIGOPS Oper. Syst. Rev.* **2010**, *44*, 2–19. [CrossRef]
11. Zha, Y.; Cui, P.; Hu, Y.; Xue, L.; Lan, J.; Wang, Y. An NDN Cache-Optimization Strategy Based on Dynamic Popularity and Replacement Value. *Electronics* **2022**, *11*, 3014. [CrossRef]
12. Alubady, R.; Salman, M.; Mohamed, A.S. A review of modern caching strategies in named data network: Overview, classification, and research directions. *Telecommun. Syst.* **2023**, 1–46. [CrossRef]
13. Liu, Z.; Jin, X.; Li, Y.; Zhang, L. NDN-Based Coded Caching Strategy for Satelite Networks. *Electronics* **2023**, *12*, 3756. [CrossRef]
14. Tourani, R.; Misra, S.; Mick, T.; Panway, G. Security, Privacy, and Access Control in Information-Centric Networking: A Survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 566–600. [CrossRef]
15. Gao, Y.; Deng, L.; Kuzmanovic, A.; Chen, Y. Internet Cache Pollution Attacks and Countermeasures. In Proceedings of the 2006 IEEE International Conference on Network Protocols, Santa Barbara, CA, USA, 12–15 November 2006.
16. Xie, M.; Widjaja, I.; Wang, H. Enhancing Cache Robustness for Content-Centric Networking. In Proceedings of the IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012.
17. Conti, M.; Gasti, P.; Teoli, M. A lightweight mechanism for detection of cache pollution attacks in Named Data Networking. *Comput. Netw.* **2013**, *57*, 3178–3191. [CrossRef]
18. Karami, A.; Guerrero-Zapata, M. An ANFIS-based cache replacement method for mitigating cache pollution attacks in Named Data Networking. *Elsevier J. Comput. Netw.* **2015**, *80*, 51–65. [CrossRef]
19. Park, H.; Widjaja, I.; Lee, H. Detection of Cache Pollution Attacks Using Randomness Checks. In Proceedings of the ICC 2012—Communication and Information Systems Security Symposium, Ottawa, ON, Canada, 10–15 June 2012.
20. Arlitt, M.; Cherkasova, L.; Dilley, J.; Friedrich, R.; Jin, T. *Evaluating Content Management Techniques for Web Proxy Caches*; Hewlett Packard: Palo Alto, CA, USA, 1999.
21. Afanasyev, A.; Moiseenko, I.; Zhang, L. RndnSIM: NDN Simulator for NS-3; NDN Technical Report NDN-0005; Named Data Networking. 2012. Available online: https://named-data.net/publications/techreports/trndnsim/ (accessed on 23 July 2023).
22. Mastorakis, S.; Afanasyev, A.; Moiseenko, I.; Zhang, L. ndnSIM 3: An Updated NDN Simulator for NS-3; NDN Technical Report NDN-0028 (Revision 2); Named Data Networking. 2016. Available online: https://named-data.net/publications/techreports/ndn-0028-2-ndnsim-v2/ (accessed on 23 July 2023).

23. NS-3 Discrete-Event Network Simulator. Available online: https://www.nsnam.org/ (accessed on 23 July 2023).
24. WAF. Available online: https://waf.io/ (accessed on 24 July 2023).
25. Silva, A.; Araujo, I.; Linder, N.; Klautau, A. Name Popularity Algorithm: A Cache Replacement Strategy for NDN Networks. *J. Commun. Inf. Syst.* **2019**, *34*, 206–214. [CrossRef]
26. Herouala, A.T.; Ziani, B.; Kerrache, C.A.; Tahari, A.e.K.; Lagraa, N.; Mastorakis, S. CaDaCa: A new caching strategy in NDN using data categorization. *Multimed. Syst.* **2022**, *29*, 2935–2950. [CrossRef]