Enhancing Cache Robustness in Named Data Networks

by

John Phillip Baugh

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer and Information Science)
in the University of Michigan-Dearborn
2018

Doctoral Committee:

Associate Professor Jinhua Guo, Chair
Associate Professor Bruce Elenbogen
Associate Professor Di Ma
Professor Weidong Xiang

John P. Baugh

jpbaugh@umich.edu

DEDICATION

I dedicate this work to my mother, Ellen Arlene Baugh, who was widowed when I was 13 years old. She raised me nonetheless through both joy and sorrow, paid for my undergraduate education so I could focus on my studies and not be burdened by debt - all while being slowly ravaged by rheumatoid arthritis. She is the toughest woman I've ever known, and the most sacrificial.

I dedicate this also to the memory of my father, Henry Jackson Baugh Sr., who worked hard and provided for his family, which he had no way of knowing would impact us for decades after his death.

Finally, but certainly not least, I dedicate this to God, who has sustained me, my mother, and my family through great trials in our lives. I am thankful for this, and most of all, for the Sacrifice of His Son, Jesus Christ, as a propitiation and atonement for our sin.

I love you mom, I love you dad, and I love you Lord.

ACKNOWLEDGEMENTS

I would like to give special thanks to Prof. Jinhua Guo, for being my doctoral advisor and a great friend and mentor during this long and often stressful process. His advice is priceless, and his patience is incredible. A lesser man would have certainly needed professional help after dealing with me for the last thirteen or so years (seven years for doctorate advising, and for my master's degree advising a few years before that.) Thank you so much for everything! I look forward to researching and working with you for years to come.

Also, I thank Prof. Bruce Elenbogen, Prof. Di Ma, and Prof. Weidong Xiang for their advice, example, and for agreeing to be on my doctoral committee. You have all had a great impact on my life in various ways, and I am a better teacher, scholar, and person for it.

I thank the CIS Department at the University of Michigan – Dearborn for being so helpful and kind over the years. You have helped me grow as a researcher and instructor as well.

I acknowledge the great students I've had over the years, and the friendships and bonds I've developed with several of them post-graduation. Knowing I've had an impact on someone's life for the better is a gift greater than gold.

I would like to acknowledge my family for the impact they have all had on my life – my brother, Jackson, my sister-in-law, Jenny, my nephew Phillip and his wife Elizabeth, and my niece Dana. I'd like to thank my numerous friends for being there for me and for understanding

when I couldn't go out to eat or hang out because I had to run simulations, read, write, summarize data, or do other "research things."

I also have dozens upon dozens of uncles, aunts and cousins that I also love and appreciate – you know who you are.

Finally, to all the great pets I've had in my life over the years. I've always owned dogs but have been a cat owner recently. To my current cats, Orange and Hoppy, thank you for distracting me and helping to relieve my stress… usually. To Hoppy, stop scratching the back door to death, I'll let you out when I get a chance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ALGORITHMS

LIST OF THEOREMS

ABSTRACT

Information-centric networks (ICNs) are a category of network architectures that focus on content, rather than hosts, to more effectively support the needs of today's users. One major feature of such networks is in-network storage, which is realized by the presence of content storage routers throughout the network. These content storage routers cache popular content object chunks close to the consumers who request them in order to reduce latency for those end users and to decrease overall network congestion.

Because of their prominence, network storage devices such as content storage routers will undoubtedly be major targets for malicious users. Two primary goals of attackers are to increase cache pollution and decrease hit rate by legitimate users. This would effectively reduce or eliminate the advantages of having in-network storage. Therefore, it is crucial to defend against these types of attacks.

In this thesis, we study a specific ICN architecture called Named Data Networking (NDN) and simulate several attack scenarios on different network topologies to ascertain the effectiveness of different cache replacement algorithms, such as LRU and LFU (specifically, LFU-DA.) We apply our new per-face popularity with dynamic aging (PFP-DA) scheme to the content storage routers in the network and measure both cache pollution percentages as well as hit rate experienced by legitimate consumers.

The current solutions in the literature that relate to reducing the effects of cache pollution largely focus on detection of attacker behavior. Since this behavior is very unpredictable, it is not guaranteed that any detection mechanisms will work well if the attackers employ smart attacks. Furthermore, current solutions do not consider the effects of a particularly aggressive attack against any single or small set of faces (interfaces.)

Therefore, we have developed three related algorithms, namely PFP, PFP-DA, and Parameterized PFP-DA. PFP ensures that interests that ingress over any given face do not overwhelm the calculated popularity of a content object chunk. PFP normalizes the ranks on all faces and uses the collective contributions of these faces to determine the overall popularity, which in turn determines what content stays in the cache and what is evicted. PFP-DA adds recency to the original PFP algorithm and ensures that content object chunks do not remain in the cache longer than their true, current popularity dictates. Finally, we explore PFP-β, a parameterized version of PFP-DA, in which a β parameter is provided that causes the popularity calculations to take on Zipf-like characteristics, which in turn reduces the numeric distance between top rated items, and lower rated items, favoring items with multi-face contribution over those with single-face contributions and those with contributions over very few faces.

We explore how the PFP-based schemes can reduce impact of contributions over any given face or small number of faces on an NDN content storage router. This in turn, reduces the impact that even some of the most aggressive attackers can have when they overwhelm one or a few faces, by normalizing the contributions across all contributing faces for a given content object chunk.

During attack scenarios, we conclude that PFP-DA performs better than both LRU and LFU-DA in terms of resisting the effects of cache pollution and maintaining strong hit rates. We also demonstrate that PFP-DA performs better even when no attacks are being leveraged against

the content store. This opens the door for further research both within and outside of ICN-based architectures as a means to enhance security and overall performance.

Chapter 1
Introduction

1.1 Background and Motivation

1.1.1 Challenges of the Current Internet

The architecture of the current Internet is host-based, depending heavily on location, which is enabled through use of Internet Protocol (IP) addresses. The usage and needs of the Internet of today is quite different from its inception. When the Internet was originally designed and implemented, its usage was largely for disseminating research data and sharing expensive resources, not content [1]. Security was an afterthought, as the need was far less in the earliest years of its existence. Today, users of the Internet are increasingly mobile and more interested in content, regardless of what server or source the desired content comes from.

On-demand content such as movies and television shows through popular content providers such as Netflix [2] and YouTube [3] is expected to be delivered in a timely, streaming manner. Since the Internet was not designed for such enormous amounts of content and performance, overlay networks and other technologies were created to overcome some of the inherent problems with the Internet. Content-delivery networks are one of these types of networks.

In North America alone, real-time entertainment is still a dominant player in downloads, accounting for 71.02% of all download-related fixed Internet traffic and 38.29% of all download-

related mobile Internet traffic during peak hours [4]. Content delivery (or distribution) networks (CDNs) are an application-layer solution to the limitations of the underlying Internet architecture. These CDNs provide redundant copies of data and services in geographically dispersed locations so that the content can be closer to the users that use them. The goal, of course, is to increase performance and availability. One excellent pioneering example of content delivery networks is the Akamai network [1]. This CDN was developed in the late 1990s and early 2000s to help organizations overcome the obstacles that are present in the fundamental Internet. Other CDNs such as Amazon CloudFront CDN [5] exist and provide similar services to companies and organizations for a fee.

However, despite all the changes of Internet usage over time, the fundamental architecture of the Internet itself remains the same as it was originally, namely: heavily host-centric. The application overlay solutions such as the CDN virtual networks are piling up in an attempt to meet the demands of today's users. As a result, many researchers have suggested that the Internet needs a fundamental re-design, such that it is information-centric rather than host-centric.

1.1.2 Information-Centric Networking

Information-centric networking (ICN) is one broad category of Future Internet Architectures (FIAs) that was promoted and funded by the National Science Foundation's Future Internet Architecture Project [6] that provides an alternative to the current host-centric paradigm. This new paradigm considers content (rather than hosts) as first-class citizens and the primary focus of the architecture. An ICN is aware of content and works to deliver the requested content to the user in an efficient manner, with full awareness of the underlying network, built-in security, privacy, and mobility in many ICN architectures, and most importantly to this research, in-network caching.

Several specific ICN architectures have been proposed. These include the Data-Oriented Network Architecture, or DONA [7], the Network of Information, or NetInf [8], Content-Centric Networking, or CCN [9], Named Data Networking, or NDN [10], [11], MobilityFirst [12], [13], and PSIRP/PURSUIT [14], [15]. Arguably the most popular among these is NDN [11], which is the specific ICN architecture considered in this work.

With the focus of this research being on in-network caching and related topics, it is important to briefly discuss the enabling technologies here. In-network caching takes place in so-called *content storage routers* throughout the network, which seek to accomplish what CDNs do, but without application layer overhead. Thus, in comparison to the typical content custodians (e.g., CDN servers, origin servers) that cache content at the application layer, content in ICN is cached throughout the network in the content storage routers. In-network caching is a very important characteristic of many of the different ICN architectures and is intended to perform much of what CDNs do, but at the network layer. More details on ICNs are discussed in Chapter 3.

1.2 Problem Statement and Methodology Overview

There are many interesting and unique challenges in information-centric networking related to security, including those related to privacy, access control, denial of service, content poisoning, and many others, but of the security issues being researched, cache pollution attacks remain largely unexplored [16]. This type of attack and the mitigation or prevention of its effects will undoubtedly become a topic of greater interest as the in-network caching mechanisms are researched further and built [17]. As a result, this work explores the effects of cache pollution attacks, and how they might be mitigated.

In ICN, a *cache pollution attack* is one in which an attacker seeks to decrease the effectiveness of in-network caches by causing the content storage routers to cache unpopular content objects, thus rendering the caches ineffectual. While some researchers consider cache pollution attacks to be a subset of denial of service (DoS) attacks, many others tend to categorize them separately. A few notable examples exist of research intending on reducing the effects of cache pollution attacks in ICN and will be explored in more detail in the literature review in Chapter 2.

However, none of the current research efforts, nor state-of-the-art approaches provide special protection against very aggressive attackers that overwhelm one or a few faces. This research is focused on providing protection against the ill-effects of cache pollution attacks in both single attacker scenarios and multiple attacker scenarios.

Methodologically, some key statistics drive the assessment of algorithms in this work. These include *cache pollution percentage* and *hit ratio*.

Cache pollution percentage is the percent of the cache of a content storage router that contains content objects that were requested by the attacker. For effective measurement (and related to the typical approach used by attackers), there should be little to no overlap between the content objects requested by legitimate consumers and those requested by attackers. Regular consumers in any given area tend to request the same popular content objects frequently, and less popular objects less frequently. From the study of Web caching, we know that sets of consumers generally request the more popular contents in a Zipf-like distribution [18], [19]. Attackers, on the other hand, will request content objects that are far less popular in order to replace actually popular content in the cache with their own.

Hit ratio is a measurement of the number of hits on a cache in a given storage router versus the total number of interest messages that have arrived at that same router. A hit occurs when an Interest message arrives for a content object, and that content object is cached in the router's cache. A miss occurs when an interest message arrives for a content object, and the data is not in the content store (cache.) For purposes of research, it is meaningful to only measure the hits and misses of legitimate users, and not include those experienced by attackers.

For a content storage router's cache to be effective, it is crucial that hit percentages be as high as possible, and to minimize cache pollution percentages. This research is largely focused on improving these two metrics under various conditions, in a named-data network (NDN) simulation environment.

It should be noted that it is possible that one (or more) of the current approaches (e.g., CacheShield) could be used in tandem with our PFP-DA scheme, so the research done toward this thesis need not be considered completely at odds with current approaches, but as potentially being complementary to them, and being complemented by them. PFP-DA is used to mitigate the effects of cache pollution, and improve cache robustness, while schemes such as CacheShield are primarily involved in the detection of attackers. Thus, applying different approaches to improve the overall performance is possible, and these technologies need not be treated as mutually exclusive or adversarial to one another.

1.3 Contributions

The primary contributions of this work are as follows:

We have designed and constructed a new approach to cache replacement algorithms based on fairness across a content storage router's faces. This per-face popularity (PFP) approach

normalizes the contributions from consumers on any given face, such that the overall impact any one face can have on the popularity of a given content object is no greater than the overall impact of the aggregate contributions on any other face.

We have revised the original PFP approach with a dynamic aging aspect, resulting in the PFP-DA scheme, which is more scalable and more efficient, and ensures that no particularly popular content object remains in the cache longer than its lifetime should allow. Thus, in addition to the normalized frequency aspects of the PFP algorithm, a recency characteristic is added as well.

To enable PFP-DA, modifications to the ndnSIM [20] simulator itself were necessary. This includes introduction of a data manager class (PopMan), modification of the forwarding plane, and modification of consumer types, such as enabling ConsumerCbr to behave as an attacker with the appropriate parameters.

The results from our tests discussed in detail later in this thesis indicate a clear advantage of PFP-DA over LRU and LFU-DA cache replacement algorithms. The advantages are measured by comparing pollution percentages, which we wish to minimize, and hit rates (hit percentages), which we wish to maximize. PFP-DA shows a level of resilience against the effects of cache pollution attacks that the current algorithms do not, as well as an overall improvement over LRU and LFU-DA even in scenarios where no attacker is active.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 is a literature review of the current topics of interest to Information Centric Networking (ICN), its goals, and some of the popular ICN architectures in the literature. It also

discusses the role of caching and the need to prevent cache pollution from having significant impact on content storage routers in ICN networks, in more detail.

Chapter 3 discusses the per-face popularity (PFP) and its modified counterpart, per-face popularity with dynamic aging (PFP-DA) and how content objects are maintained and removed from a content storage router's cache. Related equations and analysis of the scheme are presented as well. Furthermore, the chapter highlights some of the important data structures and algorithms related to the PFP schemes, modifications made to the simulator itself, and how the replacement policy works.

In Chapter 4, the simulation environment is discussed further, and simulation scenarios are defined. The two primary topologies used for the collected data, a simple and more advanced topology, are presented. Finally, Chapter 4 ends with the results of the experiments that were performed and how the results of PFP-DA compare with the popular LRU and LFU-DA cache replacement algorithms.

Chapter 5 considers how slight modifications of the PFP-DA popularity calculations can move us toward zero or at least far more negligible cache pollution. This is done by a small change to the actual overall popularity calculations, using a special β value to reduce the differences between the contributions for subsequent rankings for a given content object.

Finally, Chapter 6 covers conclusions made, and the opportunities for future research to further improve the performance and effectiveness of PFP-DA.

Chapter 2
Literature Review

2.1 Information Centric Networking

The current architecture of the Internet was originally developed in a very different world. The Internet was designed for sharing expensive resources, between cooperating, trustworthy, and relatively few hosts. The organizations that shared these resources primarily consisted of academic and governmental entities. In addition to sharing these expensive resources, the ability to add new individuals to the network easily, and even connect new networks to the Internet were primary goals.

While the Internet has made possible incredible technologies, communication, and interconnectivity, the architecture itself is showing its age. The primary usage of the Internet has changed dramatically since its inception. The primary goals of its users are no longer the sharing of expensive resources and easy addition of networks and individual devices in a trustworthy environment (although these are certainly supported); rather, users of today's Internet are interested in massive content upload and download, mobility, security, privacy, and other such features that were never even considerations of the early Internet architects.

Because of the growing focus on content, rather than on hosts, many researchers in information-centric networks (ICN) have suggested that there be a clean slate redesign of the Internet architecture itself. This new architectural paradigm (ICN) focuses on the information, and

the requesters of the information (typically consisting of consumers, also called subscribers) instead of focusing on hosts or locations, and the providers of the information (consisting of publishers, producers, or other content custodians.)

In the current host-centric architecture, if a user wishes to acquire some content, the user must know not only the content or service, but also that content or service's location, often in the form of a URL (or URI.) But the current needs of Internet users are not naturally satisfied by the network. It is obvious that users can retrieve content in today's Internet from a variety of locations (such as CDNs, described earlier.) However, the "black boxing" or obfuscation of the location and complexity of content retrieval is handled by application-level solutions and patches rather than consisting of first-class solutions provided by the network-layer itself. This structure does not take advantage of network layer optimizations.

Thus, information-centric networking fundamentally changes the way information is found and delivered, in terms of how the information is associated with the location at which it is stored. ICN decouples the information from the location(s) at which it is stored. Because this is so incredibly different, the clean-slate approach is seen as the best approach to the future Internet. Any other approaches to a Future Internet would take the form of application-layer patches on top of the fundamentally aged Internet with its inadequate protocols. This is already done extensively by many overlay networks.

There are multiple challenges in information-centric network research, including naming, name resolution, data routing, mobility, security, efficient data delivery, and many others [17].

2.1.1 Fundamental Design Considerations in ICN Architectures

When designing a new architecture, especially one for something as large as the Internet, it becomes more and more important to determine fundamental design goals. While different ICN architectures may focus on different sets of design goals, many of the more important ones will be discussed here.

Naming

Because the content in an information-centric network is the focus (rather than hosts), a way by which we identify that content becomes crucial. The process of providing such identifiers for content defines an architecture's *naming scheme*. Naming is a large challenge in an ICN, because instead of just providing "names" (IP addresses, for example) for devices within a network, an effective ICN architecture must develop a scheme that is both efficient and effective to distinguish between different information objects (data, content, services, etc.) The namespace, which is the number of available names, must be enormous in such an architecture. In [21] the authors estimate the namespace must be between 10^{13} and 10^{15}.

A primary consideration regarding naming is whether the content object name will be flat or hierarchical. A *flat name* is one that is not hierarchically or otherwise linked to other names. However, one benefit of flat names is that they can be made to be more easily self-certifying. This reduces or removes the need for a third-party authentication source, increasing efficiency and could also reduce network traffic overall. A *hierarchical name*, on the other hand, is a name that is related to other names or entities and classified in a hierarchy. An example of a human-readable hierarchical name would be a URI/URL-like name, such as umdearborn/audio/bachconcerto.m4a. However, it is important to note that hierarchical names need not necessarily be human-readable.

Name Resolution and Data Routing

Because they are related (often coupled tightly in some architectures), the topics of name resolution and data routing should be discussed together. *Name resolution* is the process by which a name is resolved to a location (or locations.) Although in an information-centric network, the application layer does not generally need to know the location of a requested content item, the network itself does. At some point, content must be located and transmitted back to the user, or if the content cannot be found, the user must be informed of this fact. In other words the content name must be matched to a content source.

Data routing is the process of producing a path through which the data travels from the producer or content custodian (provider of information) and the consumer or subscriber (requester of information.) How the name is resolved can dramatically affect how the data is routed.

There are two primary approaches in which the name resolution and data routing characteristics of an ICN architecture are associated with one another. One of these ways is a *coupled approach*. In a coupled approach, a request for information is sent to the publisher of the content or a content storage source containing the requested content (typically through multiple hops across various routers.) Then, when the publisher responds with the content itself, the content travels the same path as the initial request, just in reverse order.

In a *decoupled approach*, on the other hand, the name resolution service doesn't determine the route between subscriber and publisher in a manner which causes the data to travel the reverse path of the request.

Mobility

In the early Internet, devices and systems were relatively fixed. They didn't move very frequently. A lot has changed since these early devices were deployed. Mobility should arguably

be a high priority requirement of any architecture that will ultimately replace the current Internet architecture. In fact, the traffic generated by mobile devices is expected to surpass the traffic generated of PC traffic by 2021, with smartphones alone accounting for 33% of total Internet traffic, and PCs to account for only 25% [22]. The addressing scheme of the Internet reflects the fact that it was designed in consideration of fixed hosts, not mobile hosts.

Although mobile nodes can typically easily switch networks, changing their IP addresses in the process, this type of approach does not provide continuous connectivity during periods of mobility. This is an increasingly desirable feature.

Mobile IP ([23]) is not a very efficient solution. It is not a fundamental part of IP, but rather a patch. Worse than this, it creates so-called *triangular routing* in which packets have to be sent from the content source to a home agent, and then to the mobile device. Similar situations occur when the mobile device generates data, the data may have to be transferred to a home network agent first and then to the requesting entity. This is obviously very inefficient.

The ICN approach to mobility is built upon the publish/subscribe communication model ([24].) Entities offering information or providing a service publish advertisements for the information to the network. Entities that desire particular information subscribe to the information. A special entity called a *broker* is responsible for matching the subscriptions with publications. As the authors of [25] note, this is different than a traditional understanding of publish/subscribe. In such a traditional scheme, the publishing function involves transmitting the data to the end users/subscribers. In ICN, publishing just involves advertisement of availability of content.

It may not be clear why this aids in mobility for network entities. A fundamental decoupling occurs between the subscribers and the publishers, both in terms of *space* (physical location) and *time* (when the content is requested.) The publishers in such a model take a more

passive roll, and the subscribers have more control. The network causes the publishers and subscribers to be relatively agnostic of one another.

Security

Another high priority consideration in ICN architectures is that of security [17] , [26]. The host-based Internet that we use today was originally developed in an environment that depended on trust. Trustworthy individuals sharing research and data across the early Internet didn't have to worry as much about issues of confidentiality and integrity of data or origin authenticity. The Internet was relatively small and made up of individuals and entities that could trust one another. It was also designed in such a way that any traffic injected into it, with the right information in packets, will get forwarded. This design lends itself to denial of service (DoS) attacks and other problems.

The requirements of the early Internet are quite different from the requirements we have today. Security is of utmost importance. The security protocols and mechanisms that exist today are primarily just patches that attempt to fill the gaps in security of the underlying network design.

ICN architectures are largely focused on subscriber-oriented data transmission. In other words, there is no data flow unless the data is requested. This is an underlying network characteristic that makes it much more difficult to perpetrate certain malicious attacks. Furthermore, ICN architectures that use self-certifying names can aid the filtering of junk or malicious data.

2.1.2 Popular ICN Architectures

Although ICN can be viewed as a set of goals and principles, many different solutions have been proposed to accomplish these goals. In this section, we discuss a few of the most popular ICN architectures.

Data-Oriented Network Architecture (DONA)

The Data-Oriented Network Architecture (DONA) was originally proposed in [7] and is one of the first fully defined ICN architectures. The naming scheme in DONA is flat instead of hierarchical. This means that unlike other naming schemes, such as URLs, which have a hierarchical structure (e.g. umdearborn.edu/courses/somecourse), DONA assigns names that are truly flat and are not related to their location. The information in DONA can move about the network, change networks, change content servers, and it will still keep the same name. This strongly achieves the desired ICN goal of having decoupled location and information name.

In DONA, the names are dependent upon principals. A principal is essentially a trusted entity that uses a public key cryptographic system and has the ability to authorize hosts to publish (provide) data or services. A name in DONA is made up of a cryptographic hash (called P) of the principal's public key, and a label (called L), chosen by the principal for the named information. The principal is also responsible for ensuring that the label is unique within its affected authorization. Thus, a particular named item in DONA takes the form P:L.

When a client retrieves data, this means that the client will receive a triplet of the form <data section, principal's public key, principal's signature> and possibly other related metadata. This format of the data allows the clients to immediately determine the integrity of the data they receive.

In other words, when a client requests data with the name P:L, it receives the aforementioned triplet (<Data, Pk, Sig>), and can use the public key to ensure that it hashes correctly to P. This helps to ensure the origin integrity (authenticity) of the data. Further, the key can be used to ensure that it did in fact generate the signature for the particular data in the triplet. This provides data integrity, to ensure that the data was not tampered with between its publisher and the subscriber (origin and destination.) Due to these characteristics of the named data triplet in DONA, the names are considered to be self-certifying. In other words, no additional parties are needed to perform verification on the received data.

The recipient of the data can simply apply the aforementioned checks and determine immediately if the data is genuinely what was requested, and that the provider of the data is authorized/approved by the principal. As an additional note, another interesting aspect of names in DONA is that, while primarily focused on information and services, the names could apply to essentially any entity.

The flat names in DONA would not be very user-readable. So that brings up many challenges and issues, such as how users will remember these long and difficult names. The answer to that particular question is that users will not be expected to remember the full names, but will, similarly to URLs in a DNS system, directly interact with a user-friendly name associated to each content (service, etc.) item. One major difference however, between DNS and DONA, is that the names in DONA will not be attached to a particular location. Furthermore, DONA does not provide a reverse-lookup mechanism.

DONA is built around the goal of providing close-by copies of requested information and reducing and recovering from failures. In every autonomous system (AS) in DONA, there will be at least one logical Resolution Handler (RH.) These network entities will provide a DNS-like

service to the DONA system. The RHs themselves are organized in a hierarchical fashion. When one RH is above another on the hierarchy, it is said that the higher RH is the *parent* of the RH lower in the hierarchy.

To accomplish name resolution, DONA uses two message packets, FIND and REGISTER. When a publisher/provider wishes to offer some data or service, it issues a REGISTER(P:L), where P:L is the name based on the scheme described earlier. Each RH maintains a registration table that has information in it about next hop (to the next RH) and distance to the available copy (or copies) of the information being requested through a FIND(P:L) message.

The REGISTER messages must be authenticated by an RH. The RH issues a challenge to the client, and it must be able to sign with the principal P's private key (proving that it is the principal), or provide a certificate or other proof of having been authenticated by the principal. It is important to note that REGISTER messages can expire, since they have a TTL (time to live.) Additionally, a publisher/provider of content can issue an UNREGISTER message to indicate that they are no longer providing a particular piece of content.

The technique used by Resolution Handlers is LPM (longest prefix match.) Thus, if the registration table of a particular RH contains an entry for P:* (any data under the hash of principal P), then that is considered a match (on P.) However, if the RH has P:L as an entry in its registration table, it obviously serves the information with the name P:L (or forwards the FIND to the next hop), since this is the longest prefix match. If neither P:* nor P:L are found in a registration table, then that RH is said to not have information on the data item being searched for. If the data item is not found, then the RH forwards the FIND to its parent.

Additionally, the special form FIND(*:L) indicates that the subscriber is willing to accept information with a matching label from any publisher/provider. The format of a FIND packet is as follows, replicated from the original work of the authors of [7]:

<table>
<tr><td colspan="3">IP header</td></tr>
<tr><td>Type</td><td>Name(P:L) – 40 bytes</td><td>Next Header Type</td></tr>
<tr><td colspan="3">Transport Protocol Header</td></tr>
</table>

Figure 1: Format of FIND packet in DONA

The FIND messages continue to be forwarded up the RH hierarchy until they find a match, or until the message reaches a Tier-1 Autonomous System. At this point, if the P:L is not found, then a not-found error is generated and sent back to the issuer of the FIND message.

Also, RHs are not involved in the transport of the data resulting from a FIND message back to the requester. The actual data transfer can be handled by standard IP network routing and forwarding protocols.

Caching can also be done in DONA using the Resolution Handlers. The RHs decide what information objects to cache. Once they do so, they take an incoming FIND request and replace the source (the subscriber) IP address with its own to ensure that after the FIND request is forwarded, the content requested will definitely be sent back to the RH. The RH will know to forward the data further to the original subscriber, but the rest of the network will think that the RH *is* the original subscriber. This is an appropriate approach, especially for the decoupled approach with DONA. Note that DONA can be either coupled or decoupled. With the source replacement technique in FIND messages, the information requested is guaranteed to be delivered through the requesting RH to be cached. Then if subsequent requests for the information are made, the RH can deliver the information instead of moving the request (FIND) up the RH hierarchy.

Mobility is naturally supported since it is a subscriber-driven architecture. Mobile entities simply issue a FIND message from whatever their current location is, and the Resolution Handler infrastructure provides them with the closest copy of the information. Even mobile publishers are supported. They would simply have to unregister content from one network location and re-register in another. There is overhead with this process, but it is straightforward and clean. As noted earlier, security is also supported within the naming choice of DONA. The flat names are self-certifying.

Network of Information (NetInf)

The ICN architecture, Network of Information, known also as NetInf [8] describes information (content) as Information Objects (IOs), which are the center of the architecture, relegating hosts to a secondary position. The actual bit and byte level of the content is referred to as Data Objects (DOs), which are of less concern or interest to the user than are the IOs. However it is important to note that DOs can be further broken down into *chunks*. It is a significant feature of NetInf that there is a decoupling not only between host and content, but also between data object and information object. An IO can be viewed as an aggregation of data objects. The data objects, depending on the situation and type, could be swapped out, whereas the IO identifier would remain (i.e., persistent identifier.) The authors in [8] do not give full implementation details, but rather desirable characteristics and directions for development of a Network of Information (NetInf.)

NetInf and related research expands upon the notions of content as a centralized citizen in the network from [10] and the importance of audio/visual, web, and e-mail and further promotes the importance of real-time streaming, VoIP, and other live, time-sensitive technologies.

Information Objects contain metadata, which can be used to determine if a particular IO satisfies subscriber interest, or if it is (and how it is) related to other information objects. Metadata can take many forms, such as the use of RDF (Resource Description Framework) tags, which are already popular in Semantic Web technology.

Security is a very important topic and fundamental NetInf primitive requirement [27]. The flat names in NetInf are self-certifying and are designed such that they enable both data integrity and origin/owner integrity (identification and authentication.) The content identifiers (IDs) persist even when the owner or location may change.

The name resolution in NetInf is used to resolve the content name to a location (or locations) at a high-conceptual level as the familiar host-based DNS works. However, it should be noted that DNS requires a hierarchical namespace, so a flat namespace cannot use such a system exactly like DNS.

NetInf distinguishes between *global resolution* which works at a global scale, much like the current host-based Internet does, and *local resolution*, which allows for resolving and locating local content available in near proximity or local scope (e.g., in the same building or locale), and in intermittent scenarios (e.g., mobile ad-hoc networks.)

DHTs (distributed hash tables) are one of the most popular approaches to performing the name resolution. DHTs are quite effective, especially in terms of local resolution. However, the authors note that scalability becomes an issue largely due to the nature of autonomous systems (ASs.) The authors note a preference for flat names due to their ability to be self-certifying, but also realize the desirable characteristics of a hierarchical system in that names can be aggregated.

Therefore, an effective name resolution (NR) system would provide a trade-off between scalability and name persistence.

NetInf utilizes a *multilevel distributed hash table* (*MDHT*) approach [28]. Regular DHTs are typically quite good for intra-AS name resolution but can experience problems due to the notoriously poor cooperation between various autonomous systems [8]. This MDHT system allows multiple DHT areas (scopes) to exist. The content is registered at three levels, from most local to more global: Access Node (AN), Point of Presence (POP), and finally Autonomous System (AS.) For example, an *access node* (*AN*) DHT exists and can run its own DHT algorithm, so intra-area routing is done using this algorithm. When performing inter-area communication, a node must be found that can communicate with a higher level. For example, an AN that can communicate with a POP node, or a POP node that can communicate with an AS node. It should be noted that the content itself is registered at all three levels (AN, POP, and AS.)

If content is not found at the AS level in a particular autonomous system, then another level entity, known as the Resolution Exchange (REX) system attempts to find the content. The REX system itself is distributed among the top-level ASes.

The authors indicate that they are involved and encourage research in name-based alternative implementations. This is due to research indicating that costs of communication in dynamic networks cannot grow more slowly than linearly [29], [30].

Content-Centric Networking/Named Data Networking

Content-centric networking (CCN) [9], [10] is another one of the very few full-fledged, early information-centric networking architectures. The goals of CCN are very similar to those of

DONA and other ICN architectures. The original CCN researchers sought to develop a scalable, secure, and high-performance network architecture based upon named data.

Building upon the earliest CCN architecture, named data networking (NDN) extends on Jacobson's work and is one of five initiatives funded by the National Science Foundation under its Future Internet Architecture Program (see [11]).

Taking a different approach to naming than DONA, NDN uses a hierarchical naming scheme, where names can be like URLs. For example, the name of a given audio clip of a lecture in computer science produced by a professor at University of Michigan – Dearborn might take the form umdearborn/content/cis/audio/lec1cis427.mpg. Although this looks like a URL, this type of name is not the same as a name interpreted by DNS. In fact, it need not even be human-readable. The reason that NDN researchers have chosen a hierarchical naming scheme is because they see it as being beneficial to aggregation.

Technically, in named data networking, the content or service publishers can choose the naming scheme for specific applications since a precise naming architecture is not clearly defined in NDN. This leaves a very large namespace available to publishers of content to choose from. It is likely that if NDN is more fully adopted and implemented that publishers will have standards or conventions, but NDN itself puts very few hard requirements on the name structure itself.

NDN uses longest-prefix matching to resolve name requests. It even allows for the possibility of request content that hasn't been produced yet (some content is produced on the fly.) The publisher would create the name deterministically within the context of its naming scheme and then send the information back to the requester.

The authors in [11] suggest that this can be done either by using a deterministic algorithm that would allow both producer and consumer to arrive at the same name, or by using Interest

selectors along with the longest prefix matching to retrieve the desired data. The authors suggest that a simple set of selectors can be used along with partially known names to retrieve the data.

In NDN, there are two kinds of messages: Interest messages and Data messages. Interest messages are used by subscribers to indicate their interest in a particular piece of content. In response, a provider (publisher) of the content responds with a stream of Data messages.

The format of the Interest and Data messages is as follows:

Interest Packet
Name
Selectors (preferences, filters, etc.)
Nonce
Guiders (scope, Interest lifetime, etc.)

Data Packet
Name
Meta Information
Content
Signature

Figure 2: Format of Interest and Data Packets in NDN

Notice that both the Interest and Data packets contain the name of the information being requested, or delivered, respectively. All messages are forwarded by Content Routers (CRs.) These special routers maintain three data structures that are crucial to routing within the NDN architecture:

- FIB (Forwarding Information Base)
- PIT (Pending Interest Table)
- CS (Content Store)

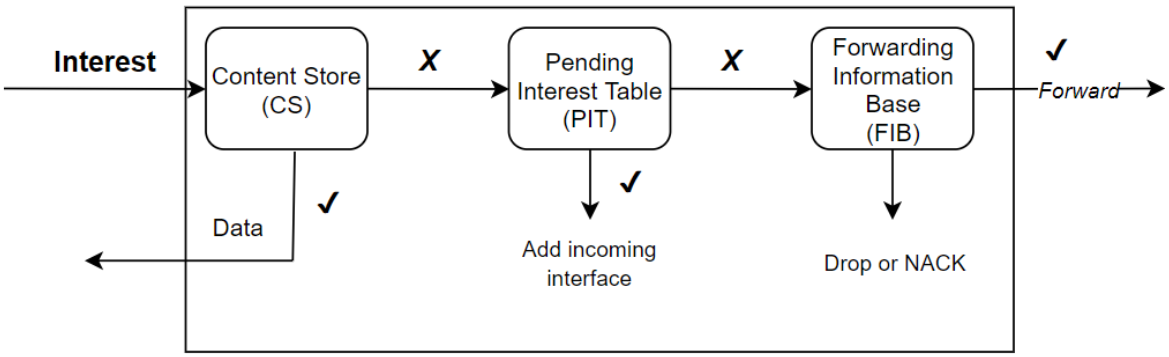

Figure 3: Forwarding in NDN (Adapted from [11])

Forwarding can be visualized as seen in Figure 3, with an X indicating a miss on a particular data structure, and a ✓ indicating a hit. The forwarding information base (FIB) is like a forwarding table in traditional host-based IP networks. However, the FIB maps the content names to output interfaces used to forward Interest messages. The FIB has name-prefix entries and can map a name to multiple output interfaces. To populate the FIB, routers advertise name prefixes (e.g., umdearborn.edu) so that the network will know that the CR has that particular content object.

As the name suggests, the pending interest table (PIT) maintains entries for Interest messages that have been forwarded but have not yet been satisfied. The name of the data being requested as well as the incoming and outgoing interfaces are all recorded in the PIT. When an Interest message arrives, the content router's content store (CS) is checked to see if the data exists. If it does, the data is returned on the incoming interface.

The content store (CS) is where a router will cache data that it determines might be popular or was requested recently, might be requested frequently, etc. If the data is not found in the content store, the next check is the PIT. If it is found in the PIT, which as I noted lists all the *pending* Interests, the PIT records only the *incoming* interface for that information that is being requested. Why is the Interest message not forwarded on? Because the pending Interest indicates that when

(or if) the data is found, a Data message containing the information will find its way back to the content router. Then, the Data will be forwarded on *all* the incoming interfaces, because these all indicate requests for the same content. Forwarding the Interest message further would have been a waste of resources.

If neither the CS nor the PIT contain the data that is being requested, then that means that the Interest is a fresh request, and must be forwarded, so the incoming and outgoing faces are recorded in the PIT until the request is satisfied.

The Data message follows the path back (similarly to the way a path is taken in a symmetric routing protocol), and the Interest entries in the PIT that have no other outstanding incoming interfaces are removed from the PIT after the forwarding of the Data on the incoming face has been performed.

Like in DONA, mobility is supported naturally in NDN. Instead of FIND messages, the mobile subscriber simply issues new Interest messages from its current location. If an Interest were generated in one network, just before entering a second network, the first CR common to both networks would suppress the Interest packets from going any further through the network. This is essentially the same case as if the mobile node in network B having moved from network A were simply a completely different node making a request for the same content. Note that an attempted delivery of data will occur to the original location as well as the new location of the mobile node. If a publisher moves, the forwarding information bases (FIBs) that point to that publisher must be updated. Thus, the publisher must advertise the name prefixes that it can handle.

In terms of security, NDN does support human-readable names being associated with the actual information object names. All Data messages include signatures based on the name and other information that can be used to identify and certify the signer or certification authority. Thus,

all nodes, including mobile, static, individual, group, or even content routers can verify that the name and packet are legitimate and belong together.

MobilityFirst

MobilityFirst [12], [13] is an information-centric Future Internet Architecture project that focuses on security (specifically, trustworthiness) and as the name suggests, mobility, as its central design goals. The fundamental support of mobility in the architecture means that mobile devices, information, services, and other network entities must be able to seamlessly transition between various networks. The recognition of the traditional IP architecture as being fragile when dealing with mobile and wireless networks and the ever-growing number of mobile computing devices were fundamental inspiration for the MobilityFirst project.

Names in MobilityFirst are GUIDs (globally unique identifiers.) These GUIDs are assigned by a GNS (global naming service.) Names are used to identify a multitude of different entities, known as *principals* in MobilityFirst such as interfaces, devices, services, human users, content, or another collection of GUIDs. The GUIDs are self-certifying, flat, 160-bit strings. The names can be one-way hashes of information fragments or entire information objects, one-way hashes of public keys, or other entities. The self-certification property of the globally unique identifiers means that any principal can authenticate other principals without using a third-party certification authority.

One strength of MobilityFirst in terms of naming, is its ability to support both named data-based information and service delivery as well as traditional host-to-host communication (however, this is done by GUIDs, not directly by anything like an IP address.) Further, besides a GUID, it is possible (and useful) for principals to be assigned a human-readable name. The GNS (Global Naming Service) can translate the human-readable names into globally unique IDs.

It is also important to note that MobilityFirst uses NAs (network addresses.) These network addresses essentially correspond to autonomous systems (ASes), but can also be used for a more fine-grained network such as a subnet, or larger entities such as data centers or Internet service providers (ISPs.)

The Global Name Resolution Service (GNRS) is the service that translates the GUIDs into network addresses. As with most of the ICNs, we identify a requester of content or service as the subscriber, and the provider of the content or service as the publisher. When a publisher wants to offer some sort of content or service, it asks the GNS (global naming service) for a GUID and then registers its network address, along with its name (GUID) with the GNRS.

When a subscriber requests some content or a service, it makes use of a GET message by sending it to its local CR (Content Router.) The GET message contains the GUID of the requested content or service, as well as the GUID of the subscriber. Since the content router (CR) forwards the GET message if it doesn't have the requested content or service to other CRs or other network entities. Before it can do this, however, it must request the GNRS for a route.

It is interesting to note that the data routing is a hybrid approach, using both IP routing and name-based routing. The GNRS is responsible for mapping addresses to GUIDs, so that traditional IP routing can be used to forward the packets. But the initial request for information, and all end host/end node communication is done by requesting a name. So, the end host interaction with the network and content is based almost exclusively on interacting with names (GUIDs) rather than addresses. The GNRS takes care of the mapping process.

The name resolution and data routing processes are decoupled. This is due to the fact that the GET and content messages/fragments are routed based on their own destination GUIDs. The

path that the GET message took need not be the path that the delivery of the data or service takes on the way to the requester/subscriber.

As the name suggests, mobility is a high priority in MobilityFirst. The GNRS is primarily responsible for handling host mobility within the network. Security is also supported based on the GUIDs being mapped to human-readable names. This is quite similar to the way that the name-information object mapping occurs in NDN.

PSIRP/PURSUIT

As the name suggests, PURSUIT uses a Publish/Subscribe paradigm instead of a Send/Receive paradigm. The authors identify three participants in PURSUIT: publishers, subscribers, and a network of brokers. Publishers are the entities which produce, own and/or are responsible for distributing content. Subscribers are the entities interested in the content – they are essentially, the consumers of the content. The Information Brokers in the network are responsible for matching publishers with subscribers, for routing the content, and for initiating the movement of data through the network so that subscribers receive the published content. In PURSUIT, the specific broker that is involved in the matching process between publisher and subscriber is called the *rendezvous point* or just *RV*. Also, this point may be referred to as a *rendezvous node*, or *RN*. This allows for effective decoupling of space/time constraints between the subscriber and publisher.

In terms of naming, the fundamental PSIRP architecture (upon which PURSUIT builds) uses flat names, but organizes these identifiers into scopes, allowing for some aggregation into hierarchical structures. Scopes can be physical structures (e.g., campus network, corporate network) or logical structures (e.g., Facebook friends, Twitter followers.) The identifiers are like those used in DONA, in that there is a Sid (Scope identifier) and a Rid (Rendezvous identifier)

used to uniquely identify an item. This binary identification is common in DONA-like naming schemes (e.g., P:L). PSIRP is a clean-slate approach to the Internet. In other words, it aims to replace the current Internet architecture without relying on current protocols.

There are three foundational functions that are executed in PURSUIT: rendezvous function, topology and routing function, and forwarding function. The rendezvous function matches subscribers' interests with specific content from publishers. The so-called topology and routing module monitors the network topology and constructs and updates the routing tables, effectively creating delivery paths for the content. And of course, the forwarding module is responsible for forwarding information on the appropriate interfaces based on the paths established by the routing function. Routers are also responsible, in many cases, for caching popular content within a domain.

2.2 Review of Security and Privacy in Information-Centric Networks

While this paper focuses on cache pollution attacks, which will be covered later, other security and privacy issues in ICNs are noteworthy and will be discussed here. The authors of [17] provide a more comprehensive discussion of many different security, privacy, and access control issues that are of importance in ICN. I will only cover a few topics here.

2.2.1 Denial of Service Attacks

Denial of Service (DoS) and Distributed Denial of Service (DDoS) form a pattern of attack in which an attacker wishes to reduce the availability of resources. DoS attacks focus on flooding a network or a specific server with requests in order to make it difficult or impossible for legitimate users to access the assets in which they are interested.

Some researchers categorize interest flooding, content poisoning, cache pollution and other attacks as DDoS techniques [31]. Interest flooding consists of an attacker sending interests into the network for content that is unlikely to be cached in in-network storage devices. This is in order to reduce the ability of these in-network storage devices to handle the interests, causing packet drop and network congestion. In NDN, the attack is largely focused on filling the PITs with the attacker's interests. If the content being requested is fake (valid prefix, invalid suffix), then the producers will simply disregard the requests, but the intermediate Pending Interest Tables are left with the pending interests until they are purged due to expiration.

The categories of countermeasure used against various DoS attacks in ICNs include rate limiting on faces, monitoring of PIT size growth, statistical modeling approaches to detect abnormal traffic patterns, and other detection mechanisms designed to punish attackers.

2.2.2 Content Poisoning Attacks

As discussed earlier, some researchers consider content poisoning to be a subset of DoS attacks. However, it is worth treating separately. Content poisoning attacks seek to fill in-network storage devices with invalid or malicious content. This type of attack requires that an attacker control at least one intermediate router so that it can inject invalid content into the network. The content has a valid name, but fake or malicious content or a fake signature. Some ICN architectures are more resilient to this type of attack than others, especially those architectures that use self-certifying names.

A DNS cache poisoning attack occurs when an attacker inserts bad data into a DNS server's cache so that they can control server responses for those names. Interest and data packets in NDN are routed directly, and not converted to addresses, so it would seem the problem is solved.

However, this is incorrect. The authors of [32] note that NDN cache poisoning can be achieved using a combination of route hijacking and content poisoning. If carefully planned, an attacker can position themselves in a network to answer interests with invalid (and often, quite large) content.

In terms of countermeasures, many approaches exist. Digital signing and verification of each packet is one especially popular approach. For example, SCID (self-certifying interest/data packet) [32] are one such solution. These types of signatures allow the consumers to immediately determine if content was poisoned or not.

Another approach is exclusion-based feedback [33]. This approach takes advantage of a feature in NDN called selectors. If a client independently determines content with certain data to be invalid or poisoned, it can exclude this content and report to the network storage device (content storage router) that it excluded that content. The content storage router can then use these exclusions, collected from all consumers on all faces, to reduce the rank of a content object. Although a potentially effective approach, malicious consumers could send false exclusion data to the router to reduce the rank of normal content.

2.2.3 Privacy

Privacy is an area of discussion that has been largely untouched in ICN architectures [26]. Many architectures naturally leak content and location information, names, payloads, signatures, and other such information. To truly introduce privacy into ICN-based architectures would require introduction of primitives into the network layer with no overlays.

If an overlay was used, this would cause leakage of current IP problems into ICN – duct-taping a fundamental problem that should be solved and optimized at the network layer. However,

the authors of [26] argue that an upper-layer service might be required to provide privacy. They even suggest that trying to provide privacy in a name-based architecture like NDN is a fool's errand.

The authors of [34] cover privacy as it relates to caching. They suggest that much information can be gleaned simply by an attacker requesting content from a content storage router. If the content was cached, then the attacker can assume that someone with which the attacker shares a face must have requested the content also. This simple type of attack would be classified as a timing attack.

They go further in establishing mitigation for two types of traffic: interactive (such as streaming communication) and content distribution. One of the tactics involves artificial delay. This too comes with problems and has a potential of reducing cache effectiveness. Other approaches such as random cache to create an anonymity set of content. This too comes with trade-offs, and efficiency considerations.

Few papers exist on privacy in ICN. Therefore, privacy remains largely an open issue in ICN-based architectures.

2.3 Caching and Current Approaches to Mitigating Cache Pollution Effects

In a normal situation within ICN, content storage routers (an enabling technology for in-network caching) should cache the most popular content objects. These cached items typically are requested following a Zipf-like distribution [18]. The goal of caching the most popular items is of

course to reduce the latency and network load associated with requests and data transfers for specific content objects.

The goal then, of an attacker leveraging a cache pollution attack against an information-centric network is typically to reduce the effectiveness of the cache by pushing popular contents out. An attacker could do this in many situations by simply requesting less popular content frequently (often with far more frequency than legitimate consumers' typical request frequency.) So, the goals of an attacker in a more granular view is to increase link utilization and to decrease cache hits (and likewise, increase cache misses.)

2.3.1 Cache Pollution

Cache pollution is frequently divided into two categories: *locality disruption* and *false locality* [17]. Locality refers to the phenomenon in which the same content set are accessed with higher frequency than other contents [35]. The name is derived from the fact that a particular *locale* (location or area of influence) is likely to have certain content sets that are more popular than others, and thus more frequently accessed.

The locality disruption attack seeks to disrupt the area served by a particular cache (its locale.) The requested contents may come from the entire content space and assume no special knowledge on the part of the attacker. The entire purpose is to *churn* the cache. In other words, the purpose is to move popular items around in the cache and ultimately cause some of the lower ranked of the most popular items to be pushed out of the cache. This in turn, increases cache pollution and decreases hit rate.

During a false locality attack, the attacker attempts to change the popularity distribution of a given cache by requesting particularly unpopular contents. This type of attack, rather than

requesting randomly selected content objects from the universe of content or a very large subset, involves the attacker continuously requesting a smaller number of items in an attempt to push even the most popular content objects out of the cache.

2.3.2 Current Related Research

Much of the research that has been done to prevent cache pollution has involved CCN and NDN. One such example is the work of Xie et al. with *CacheShield* in [36]. This particular work is intended to work in most large generic networks, but the research specifically involves CCN. The CacheShield approach is a proactive approach that attempts to thwart locality-disruption pollution attacks by exploiting the ability to distinguish between the characteristics of normal requests and malicious requests. The authors note that normal requests follow Zipf-like distribution. As the name suggests, this distribution is based on Zipf's law which, when applied to content caching, states that the relative probability of a request for the *i*th most popular content object is proportional to $1/i$. Malicious requests, on the other hand, tend to follow uniform distributions. CacheShield tracks the cached content objects, and even the requested objects' names. Although the names take up some space, they take up, typically far less space than the actual content objects. Therefore, when a content item is requested but not cached, its name is recorded (if this is the first request), or the count of requests for that name is incremented by one (if the content has been requested before.) Ultimately, if the content count reaches a certain threshold, then the content will be cached. A major weakness of this approach is that the attacker might be able to determine the threshold, t, and then simply request objects past that threshold, causing their (previously unpopular) content to be cached.

Conti et al. [37] describe a somewhat more realistic threat model. This work differentiates between two different attack strategies, called *broad-selection* and *smart-selection*. Broad-selection, used by the authors of CacheShield [36] is a strategy in which the attackers would issue interest messages for all existing content with equal probability. However, as Conti et al. [37] point out, this is not always realistic in an attack scenario. Further, the authors' findings show that this attack is not effective with LFU (least-frequently used) cache replacement algorithms being used. So, they enhance the threat model to test their own work using so-called smart-selection. With smart-selection, the adversary focuses on a small subset of the universe of content. Specifically, smart-selection is used to focus on the least-requested content in a Zipf distribution (the tail of the distribution.) The authors vary the number of content objects used in the attack, from 0.5 to 1.0 (50% to 100%) of the entire content domain. This would of course mean that 100% would mimic the uniform attack scenario in [36].

This work of Conti et al. [37] uses a machine-learning based approach. Their algorithm has two phases: *learning* and *attack detection*. The learning phase results in a calculation of an attack detection threshold τ. The attack detection phase also uses a computed value δ_m, which for a given measurement *m*, provides a measurement of the variability to determine normal behavior. If at some point δ_m surpasses τ, then an attack is detected. This work is an improvement on CacheShield, but still suffers from several weaknesses. For example, it detects the existence of an attack but does not track which messages caused the attack or given any origin information. Further, it assumes that normal behavior follows Zipf-like distribution, and attacks always follow uniform distribution. The authors argue that an attacker could simulate a Zipf-like distribution, of course, which would render this approach less effective. The authors also assume that a content store has the capacity to store 1% of the entire content domain.

Karami et al. in [38] describe a cache replacement technique that helps to both detect and mitigate both false locality and locality disruption attacks. This technique is based on the attributes of content and their relationship to attack vs honest consumer content. The authors use the Adaptive Neuro-Fuzzy Inference System (ANFIS) approach that has been used in linguistics studies to find nonlinear relationships between inputs and outputs. This approach, as the name suggests, uses a combination of both ANN (artificial neural networks) and fuzzy systems. The neural networks are used to automatically extract fuzzy logic rules from the numeric data. The membership functions are also adjusted in an adaptive manner through this process. There are three steps in the technique, namely I/O pattern extraction, then accuracy verification of the ANFIS, and finally integration into the cache replacement policy. The approach does show promise in increasing cache robustness in mitigating cache pollution attacks. Although the approach does show improvement over CacheShield, the scalability is somewhat questionable due to the amount of memory and computational overhead required.

All these approaches, as [17] points out, and as seen above, suffer from various deficiencies. The problems range from ineffectiveness against smarter attack scenarios to extremely high computational overhead at routers. Therefore, even with the work that has been done, the exploration of cache pollution attacks and their effects, and ways to overcome these effects remain an open research area with room for significant improvement [16], [17].

Chapter 3
Per-Face Popularity Scheme and Cache Replacement Algorithms

3.1 Overview of Per-Face Popularity Concepts

Previous approaches to cache pollution prevention are often highly susceptible to increases in attack rate, and focus largely on detection mechanisms, which may be difficult if the attackers are smart. Further, particularly aggressive attackers on a single, or even small percentage of faces can potentially churn the cache quite substantially. Due to the limitations of previous approaches in addressing cache pollution attacks in ICN, we have developed a scheme that prevents Interest messages arriving over any one face from affecting the cache more than those arriving over any other face. Our scheme is called the *per-face popularity* (PFP) scheme.

The goal of our scheme is to mitigate the effect that an attacker can have on a cache and ultimately limit the percentage of the cache that can be polluted, and more importantly, increase the overall hit rate of the most popular items. Although legitimate users can help hoist a given content object into the cache if the items have borderline popularity, our scheme can significantly reduce this impact.

Additionally, if the attacker(s) request content that is outside the set of typical content on the higher frequency end of the Zipf-like distribution, our scheme shows significant protection against even very aggressive false locality attacks from one or even several attackers against a given content storage router.

When designing a PFP-based solution, one technique would be to give each face its own cache. Although this is implementable, it is not feasible given performance requirements and the desire to make the algorithm(s) scalable. Instead, the PFP scheme puts a limit on the amount of influence each face can have on which items remain in the cache.

Our scheme uses a popularity manager (PopMan) class that maintains lists of content request frequencies from different faces. Ultimately, the items that are maintained in the content store (cache) are those that are most popular overall, given equally weighted influence from consumers whose request ingress across all faces of the content storage router. *Overall popularity* refers to what items are most popular based on the Interest messages entering a router across all faces, without any one face having more influence than any other face.

In many current schemes such as the basic least-frequently used (LFU), least-frequently used with dynamic aging (LFU-DA) or least-recently used (LRU) [39], any single face or small set of faces can receive far more Interests than others and this can overwhelm the overall popularity for a given content object or objects. This is particularly effective against LFU-based schemes, since they are concerned with frequency of requests, but also affects LRU-based schemes, since the likelihood of most recently requested items being attacker-requested items also increases.

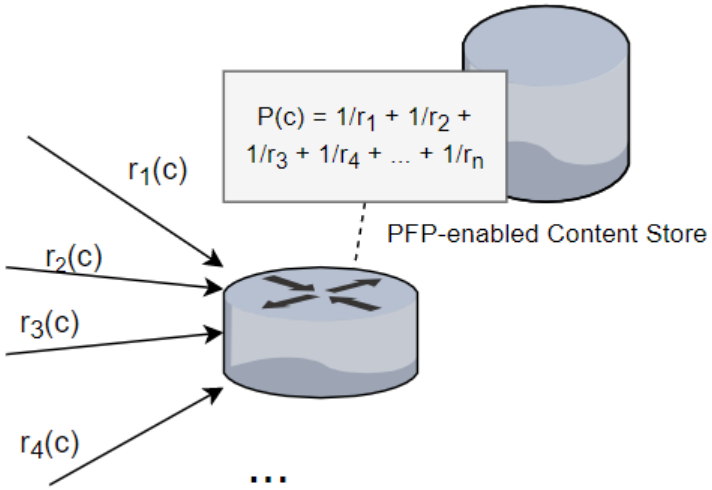

Figure 4: Normalizing Popularity Contributions in PFP

In our PFP scheme, we normalize the contributions from each face so that Interests arriving over any one face have no more effect on the popularity than those arriving over any other face. This process is visualized in a simplified model in Figure 4. As a result, the items that remain in the cache are those that have legitimate popularity, represented fairly across all faces. It is this normalized overall popularity that is used to determine overall content store ranking and ultimately, what items are evicted and what items remain.

3.2 Equations, Analysis, and Theorems

3.2.1 Popularity Calculation

The popularity of items coming across any given face is based upon the frequency of Interest messages arriving on that face. When the Interests arrive, they affect the frequency used in calculating the popularity of any given item. This popularity is then used to rank items within

a list of most popular items, and then the rank, *r*, is presented by each face and used in the overall popularity contribution, thusly:

$$PFP\ Contribution = \frac{1}{r}$$

Equation 1: Contribution from a Face

If a content object is not ranked on a particular face, then its contribution is 0.

The overall popularity of a content object C, across all faces, is the sum of the popularity contributions from all the faces. That is:

$$P(C) = \sum_{i=1}^{n} \frac{1}{r_i} = \frac{1}{r_1} + \frac{1}{r_2} + \cdots + \frac{1}{r_{n-1}} + \frac{1}{r_n}$$

Equation 2: Overall Popularity, *P(C)* of a Content Object, *C*

where r_i is the ranking of a given content object C from the i^{th} face of *n* faces.

3.2.2 Analysis

Given a content router with *n* faces, if cache pollution attacks are launched against *k* faces on a router using the PFP scheme, the fraction of the cache that can be polluted (with no contributions from non-attacker faces) is on average about $\frac{k}{n}$.

To launch a pollution attack, attackers may control one or a small set of faces and send large amounts of Interests through those faces. However, with the PFP scheme, we normalize the contributions from each face so that Interests arriving over any one face have no more effect on the popularity than those arriving over any other face. As a result, the percentage of cache that attackers could pollute is approximately proportional to the faces they are able to have total control against.

As we shall see with the experimental results, even with a relatively high pollution percentage, the hit rate of legitimate consumers can still be quite high. In many situations (e.g., when compared to LRU), PFP-DA has a higher pollution percentage, but also a much higher hit rate.

3.2.3 Theorems

Under certain assumptions, it is possible to establish an upper bound on the amount of cache that an attacker can pollution, resulting in theorems related to the cache pollution percentage. We assume that consumers are all requesting various content items, such that the total number of content items being requested across the universe of contents is substantial enough to equal the size of the cache, n. In other words, all the popular content is being requested by the various consumers. In practice, the behavior observed might be different and the cache pollution percentages can vary widely. However, given the aforementioned assumptions, we can establish the subsequent theorems.

Theorem 1: Single Face Attack Scenario

Given a content router with n faces, if a cache pollution attack is launched against a single receiving face of that router on which the PFP scheme is used, the fraction of cache that can be polluted is no more than $1/n$.

Proof:

Consider that an attacker controls a single face. If there are a total of n faces, then that means the logical quota given to each face is t/n where t is the total cache capacity and n is the number of faces.

Let m be $1/n$ of the total cache capacity. That is,

$$m = \frac{1}{n} * t = \frac{t}{n}$$

Equation 3: Logical Quota of Cache Capacity

To prove the theorem, we need only show that no more than m unpopular items (selected by the attacker) can be cached with only one face receiving interests entirely from the attacker. The top m ranked content objects (or object chunks) of each face have a popularity of 1, 1/2, 1/3, …, $1/m$, respectively. So there are $m * n = t$ content objects with a minimum popularity of at least $1/m$.

Therefore, in order for a content chunk to be cached, it must have an overall popularity of at least $1/m$. If an attacker can influence only a single face (and other faces are receiving interests for the top n popular objects from legitimate consumers), then he can only cause m unpopular objects to have a popularity of at least $1/m$. Therefore, the attacker can only cache m items maximally.

Thus, it is demonstrated that Theorem 1 holds.

Theorem 2: Coordinated Attacks Launched Through Two Faces

Given the same assumptions as described earlier in this section for Theorem 1, we establish a theorem related to an attack against $k = 2$ faces. Given a content router with n races, if cache pollution attacks are launched against $k = 2$ faces, on that router on which the PFP scheme is used, the fraction of cache pollution is no more than

$$\frac{k^2}{n} = \frac{2^2}{n} = \frac{4}{n}$$

Equation 4: Cache Pollution Ratio For Two Attacker Scenario

Proof:

If an attacker is coordinating a more sophisticated attack and has complete control of the Interests entering two of the faces of a router, given no contribution from faces not under the attacker's control, the fraction of cache pollution is no more than

$$\frac{4}{n}$$

Again, for a content item to be cached, it must have an overall popularity of at least $1/m$. First, we show the maximum number of items that can be cached is no more than $\frac{4m}{t} = \frac{4}{n}$. This is the upper bound.

An item needs to be ranked at least $2m$ by at least one of the attacked faces. Otherwise the highest overall popularity score for any item with ranking no better than $2m$ is

$$\frac{1}{2m+1} + \frac{1}{2m+1} = \frac{2}{2m+1} < \frac{1}{m}$$

Thus, it would not be cached.

Therefore, at most $2m$ items from each face can be cached, thus with control of two faces, as most $4m$ items total can be cached by the attacker.

Next, we demonstrate it is possible to have $4m$ items with an overall popularity of at least $1/m$. The top m ranked content items on each face have a popularity of at least $1/m$. In additional, there

is a way to have an overall popularity of at least $1/m$ items ranked between m+1 and $2m$. For example, items ranked both by face 1 and face 2 (the attackers), as follows.

Ranking from i_1	Ranking from i_2	Overall popularity
m + 1	3m – 1	1/(m+1) + 1/(3m-1) = 4m/(m+1)(3m-1) > 1/m
m + 2	3m – 2	1/(m+2) + 1/(3m-2) = 4m/(m+2)(3m-2) > 1/m
m + 3	3m – 3	1/(m+3) + 1/(3m-3) = 4m/(m+3)(3m-3) > 1/m
…	…	…
m + (m-1) = 2m – 1	3m – (m-1) = 2m + 1	1/(2m-1) + 1/(2m+1) = 4m/(2m-1)(2m+1) > 1/m
1/2m	1/2m	1/2m + 1/2m = 2/2m = 1/m

Table 1: Coordinated Attacks Through Two Faces

Thus, as we can see from the data in Table 1, items ranked between m + 1 and $2m$ by face 1 are cached, and those ranked between m + 1 and $2m$ by face 2 are cached. But this is the maximal amount an attacker with the Interests entering two faces on a router can control.

Therefore, Theorem 2 holds.

3.3 Per-Face Popularity (PFP) Cache Replacement Algorithm

The original system that we developed was the regular per-face popularity (PFP) replacement scheme. This scheme uses several interactions among various components, including custom components entirely of our design, as well as the ndnSIM [20], [40] system, which is a network simulator extension built on top of the extremely well-known ns-3 [41].

3.3.1 The Popularity Manager (PopMan)

A significant component of the overall PFP design is the popularity manager (PopMan.) It contains several fields of useful data that are utilized by the overall system in calculating what items should be evicted due to lowest popularity. One crucial field is a vector of

FaceContentFrequency objects. The FaceContentFrequency class and its composition relationship with the ContentFrequency class is modeled below.

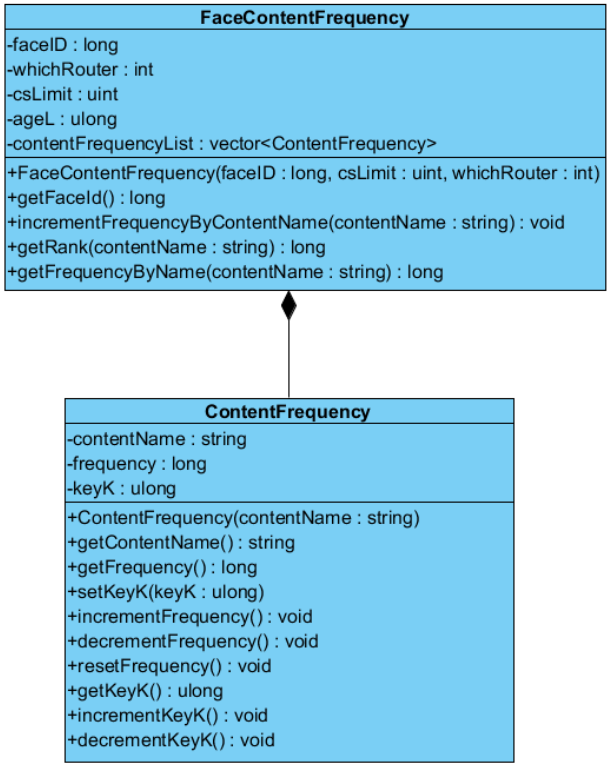

Figure 5: UML Class Diagram of the FaceContentFrequency and ContentFrequency Classes

As we can see in **Error! Reference source not found.**, the class diagram indicates a composition relationship between the FaceContentFrequency and the ContentFrequency classes. Obviously, this diagram does not cover the entire PFP design, but shows a little about its subcomponents so that we may understand it better.

From a bottom-up description, the ContentFrequency objects are used to encapsulate a content object name with a frequency (the keyK field and associated methods are used with the PFP-DA, and not with regular PFP.) Although this class is general enough to be used in a regular

popularity scenario without any per-face considerations, it is used by the PFP system by its composing class, FaceContentFrequency. Namely, a vector of the ContentFrequency objects is maintained within each FaceContentFrequency object. This vector is used so that each face can keep a list of the names of all content objects requested on it.

The popularity manager (PopMan) class (not indicated in **Error! Reference source not found.**) maintains a vector of FaceContentFrequency objects itself. Thus, the PopMan can manage and determine the rankings exposed by the FaceContentFrequency objects' methods. This allows the PopMan to utilize this data in its getOverallPopularity method, which directly applies Equation 2.

```
Input:  contentName as string
Output: overall popularity as a double

overallPopularity := 0.0
facePopularity := 0.0
rank := 0

for every ci ∈ faceContentFreqList do
  rank := ci.getRank()
  if(rank != -1)
    facePopularity := 1.0 / rank
    overallPopularity := overallPopularity + facePopularity
  end if
end for

return overallPopularity
```

Algorithm 1: Popularity Manager's Get Overall Popularity Pseudocode

Furthermore, the PopMan maintains a list of all the iterators to the content that live in the actual content storage router's cache. It then uses the getOverallPopularity method in its getContentToEvict method, which is used directly by the eviction policy (more on this later.)

3.3.2 Simulator Modifications

As a matter of necessity, we modified the ndnSIM simulator itself for various purposes, including collection of the required hit and miss data from legitimate users. The PopMan class was placed within the ns-3 and ndnSIM source to easily obtain and provide information to the modified classes therein.

Of particular interest is our modification of the Forwarder class, which is the powerhouse of the Network Forwarding Daemon (NFD) [42]. Because ndnSIM does not currently expose any methods to obtain actual Interest hit or miss data, this was implemented as a modification of the Forwarder class to provide necessary bespoke functionality. Additionally, ndnSIM does not provide any reasonable way of subclassing the Forwarder class and setting the subclassed Forwarder as the new forwarder to use in the NFD. Therefore, modifying the actual class was the only straightforward way of accomplishing our goals.

The data collection done within the Forwarder class takes place primarily in its onIncomingInterest, onContentStoreHit, and onContentStoreMiss methods. These methods were provided by ndnSIM originally but have been modified in order to collect data about the interests and their hit and miss values.

The onIncomingInterest method of the Forwarder class calls the recordInterestStats method of our popularity manager, PopMan. This is used by the PFP scheme to determine what content should be evicted. This is done as follows.

```
Input:  inFace of type nfd::Face, representing a specific face over which the
interest was made
Input:  interest of type const ns3::ndn::Interest&, representing the interest
object that triggered this method call

if(this == nullptr or we're not using PFP)
  return
end if

faceID := inFace.getId()
nameOfContent := getShortName(interest)
ind := getIndexOfFace(faceID)

if(ind == -1)
  construct faceCF(faceID, csLimit, whichRouter)
  faceCF.incrementFrequencyByContentName(nameOfContent)
  faceContentFreqList.push_back(faceCF)
else
  faceContentFreqList.at(ind).incrementFrequencyByContentName(nameOfContent)
end if
```

Algorithm 2: Popularity Manager's Record Interest Stats Pseudocode

Every node's Forwarder object will automatically maintain a popularity manager object variable. However, if the node is not a content storage router (e.g., a regular router, consumer, producer) then, the object pointer will be null.

Only in the case of the storage router having a popularity manager object set will the pointer not be null. In Algorithm 2, we can see that the first part ensures that whatever router we're on is a content storage router, and that it is using the PFP algorithm.

After obtaining the face, name of the content, and index of the face from the faceContentFreqList, the algorithm either creates a new FaceContentFrequency object (if the ind is -1, indicating that it was not found by getIndexOfFace) when the face is new, and otherwise uses the found FaceContentFrequency object and increments its associated frequency.

The Forwarder class's onContentStoreHit and onContentStoreMiss each call the PopMan class's recordHitMissInfo method. This method is for data collection and is not particularly useful for the actual PFP scheme itself, so it can be considered part of the metrics collection, but is still

of important note here. They only do so if the content is not one of the attacker's content. In a deployable implementation, it would obviously render cache pollution attacks trivial if we knew *a priori* which content requests were from an attacker. However, for data collection purposes, it's important that we only count the hits and misses that originate from legitimate consumers.

The fundamental recordHitMissInfo algorithm is shown below in pseudocode.

```
Input:  interest of type const ns3::ndn::Interest&, used to indicate the interest
Input:  isHit of type bool, to distinguish between a hit (true) and a miss (false)

if(this == nullptr)
  return
end if

nameOfContent := getShortName(interest)
seqNumOfContent := empty
component := interest.getName().at(1)

if(component.isSequenceNumber())
  seqNumOfContent := to_string(component.toSequenceNumber())
end if

index := getIndexOfCHMF(nameOfContent)

if(index != -1)
  construct chmf(nameOfContent, seqNumOfContent)
  if(isHit)
    chmf.incrementHitFrequency()
  else
    chmf.incrementMissFrequency()
  end if

  contentHMFreqList.push_back(chmf)
else
  if(isHit)
    contentHMFreqList.at(index).incrementHitFrequency()
  else
    contentHMFreqList.at(index).incrementMissFrequency()
end if
```

Algorithm 3: Popularity Manager's Record Hit and Miss Info Pseudocode

Algorithm 3 has some similar features that we've discussed before. Initially, it ensures that the popularity manager object pointer is not null. This time, however, it does not ensure we are

using PFP. This is because we want to record hit and miss data regardless of whether PFP is enabled or not.

Afterwards, we set the content name, sequence number, and component variables. It is important to note that in ndnSIM, different data is stored as interests. For example, handshakes, acknowledgements, and other setup information can travel as Interest messages. However, the different content objects in ndnSIM are identified by different sequence numbers. Therefore, only actual content object interests have a sequence number. Therefore, we test the component to ensure it is a sequence number, and then obtain it if it is (as a string.) We attempt to find the name of the content in our list of ContentHMFrequency objects (content hit/miss frequency objects.) These objects are used to maintain hit and miss information for our data collection. If an object corresponding to the interest exists in our list, positive integer is returned. If not, a -1 is returned, indicating this is the first time we've seen this content interest.

If the returned value is -1, we create and push back a new ContentHMFrequency object with the appropriate hit or miss incrementation, and if it's not -1, we use the index of the found object, and just increment its hit or miss frequency.

The data that is collected regarding interests is ultimately used to determine what content item should be evicted from the actual content store when eviction is required.

3.3.3 Policy Subclassing

Continuing our discussion, the next important component of the PFP scheme is the custom Policy class. The UML diagram below shows the fields and methods related to our discussion, even though others are inherited (and left empty.)

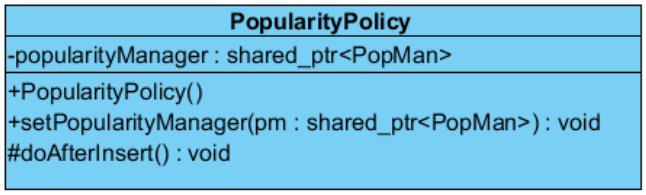

Figure 6: UML Class Diagram of the PopularityPolicy class

The PopularityPolicy class is a subclass of the Policy class, provided by ndnSIM. This class is responsible for the decisions that are made regarding cache replacement for the content store. Fortunately, unlike the Forwarder class, the Policy class can be subclassed and then the subclass can be set as the ndnSIM system's policy to indicate the custom replacement policy that is to be used.

The PopularityPolicy class maintains a shared smart pointer, popularityManager, to the popularity manager object. The constructor sets the managed popularityManager pointer to nullptr.

Arguably, the most interesting of the methods is doAfterInsert, which is a pure virtual (abstract) method in the Policy class. Of the inherited pure virtual methods (doAfterRefresh, doBeforeErase, doBeforeUse and evictEntries being the others), this is the only one that our scheme fully implements, while the others are simply left as stub methods.

This method, doAfterInsert is called by the system after an insertion is made into the cache of a given router to which the policy object is applied. An interesting aspect of ndnSIM is that it automatically caches everything by default. Even if someone wishes to implement a cache insertion algorithm (rather than a cache replacement algorithm), they must emulate this behavior by immediately evicting the content object that was cached. Therefore, from a purely technical

perspective, all the caching algorithms are cache replacement algorithms, even if some implementations imply they are implementing a caching (cache insertion) algorithm.

The system removes an item from the content store (cache) when emitSignal is called. This (macro) method has two parameters passed to it: the signal to emit, and the associated data to perform its duty. Specifically, for evicting content objects, the signal beforeEvict is passed in to emitSignal, along with the iterator that is to be removed from the content store.

Because of the work done in the PopMan class, only minimal work is done in the PopularityPolicy class in doAfterInsert, as follows.

```
Input:  it of type nfd::cs::iterator, the iterator representing the item in the
content store that was just inserted

dataIt := it->getData()
dataName := dataIt.getName().toUri()
popularityManager->addIterator(it)

if(getCs()->size() > getLimit())
  iterator iteratorToRemove := popularityManager->getContentToEvict()

  popularityManager->removeIterator(iteratorToRemove)

  this->emitSignal(beforeEvict, iteratorToRemove)
end if
```

Algorithm 4: PopularityPolicy's doAfterInsert Pseudocode

The code in Algorithm 4 is quite straightforward for the most part. As an item is added to the content store (the default behavior, done automatically when a Data message enters the router across any face), the name is obtained and the popularityManager pointer adds the iterator to its object's list of iterators.

The size of the content store, which is the number of items currently in the cache itself, is compared to the limit, and if it exceeds the limit, the cache replacement routine is initiated. If the

size is not greater than the limit, then there is room in the content store and no need to initiate the cache replacement algorithm. This is done primarily by calling the popularity manager object's getContentToEvict method through its iterator.

After the iterator is obtained, the popularity manager object removes the iterator from its own list, which reflects the content of the actual cache. Finally, the signal to remove the given content object from the content store is emitted. A clear separation of concerns is established to decrease coupling and increase cohesion in the software, as the popularity manager does the majority of the work related to determining what to evict, and the policy simply uses this information to do the actual eviction.

3.4 Per-Face Popularity with Dynamic Aging (PFP-DA) Cache Replacement Algorithm

A somewhat subtle issue with the original PFP scheme that reduces scalability is in its maintaining of names and frequencies of all items for which interests have been expressed and doing so for each face. The problem with this is that the entire universe of contents is massive and maintaining even the names of all contents (or all of what is requested of it) is simply not scalable. Many items in the universe of content might be requested only once and never have a chance of entering the actual cache, yet a potentially enormous list[1] of content object names is maintained for each face of the router.

[1] The phrase "list" in this context is used as a general term for a collection, and does not necessarily refer to an implementation of the ADT List

Therefore, using techniques that have been applied in other algorithms, such as LFU-DA [39], these name lists can be dramatically reduced, improving scalability without the PFP algorithm being reduced in any substantial way.

Considering some background information first, the least frequently used with dynamic aging algorithm (LFU-DA) takes the least frequently used (LFU) algorithm and adds an aging component to the items to ensure that no items remain in the cache longer than they should. The fundamental LFU algorithm maintains items in its cache based on the frequency of request. Quite simply, the most frequently accessed item remains at the "top of the list", and less popular items below it, with the least popular item(s) in the cache being pushed out as necessary to make room for new popular items.

However, a flaw with the basic LFU algorithm is this – it doesn't take *recency* into consideration. For example, a content object C*j* might be cached at some time and might be extremely popular – receiving thousands upon thousands of requests. Therefore, every time a request is made, the frequency counter of C*j* is incremented. Later, this item may not be nearly as popular, perhaps rarely or never receiving requests. But, because it was so incredibly popular at some time in the past, it continues to occupy space in the cache even though it is not currently popular.

The solution used in LFU-DA is to use a *dynamic aging* aspect to maintaining items in the cache. While some LFU-A (LFU with aging) algorithm exist, they require special parameters to be passed in *a priori*, and to be manually tuned, which is undesirable. The dynamic aging aspect allows the system itself to tune the system and manage the aging. One specific variation of the LFU-DA algorithm found in [39], and uses the following criteria.

- K_i, the key value of content object i
- L, which is the running age factor, that starts at 0 and is updated every time an item in the cache is replaced. For an item f that is being replaced, K_f is the priority key of that item. When the item is replaced, $L := K_f$.
- F_i is the frequency (number of requests) for the item

We calculate the key for an object being brought into the cache as follows:

$$K_i = F_i + L$$

Equation 5: LFU-DA Key Value Calculation

Instead of being kept in a cache by means of the frequency alone, the key value is used instead. Since this key value reflects a larger value than the frequency itself (typically), it helps to ensure a natural churn of the cache with recency considered.

Drawing upon the LFU-DA approach, PFP-DA (Per-Face Popularity with Dynamic Aging) uses the keys (as mentioned earlier) to organize the popularity rankings themselves. We do not apply the key technique directly to the content store itself. Instead, the key technique controls what remains in the popularity ranking lists of content names and what is evicted from them.

Instead of having a hypothetically unbound list of content names for each face that could be as large as the entire content space, we limit the size of each of these popularity-ranking lists to the size of the cache. For example, if the content store of the router can hold 100 items, then each face maintains a list of up to 100 items each. Therefore, one could even say that there are two planes of cache replacement algorithms at play here. One is for the "cache" of content names for each face, which adds item names as they are received until the list is full and then applies to

replacement routine; the other is the actual content storage router cache, which uses the PFP algorithm. Combined, these form the PFP-DA algorithm.

Chapter 4
Experimental Results

4.1 Simulation Environment and Set-up

As mentioned previously in this dissertation, the network simulation environment that was chosen was ndnSIM (specifically, version 2) [20], an extension built on top of the extremely popular ns-3 simulator [41]. This choice was made largely due to the extensive use of ndnSIM for NDN-related research throughout the research community.

Since ns-3, and as a result, ndnSIM do not work on Windows platforms yet, a Linux-based machine was utilized. Specifically, the machine has a System76 Gazelle laptop, with Ubuntu 16.04.1 LTS (64-bit) as the operating system. The processor consists of a quad-core 6MB cache 3.5 GHz i7-6700HQ. 16 GB of DDR4 RAM are installed as well.

The ndnSIM simulator consists of many components extending the use of the ns-3 simulator. Although a full coverage of the entire simulation system could easily fill several books, a few of the significant classes and features are discussed here. Note that some of these have already been discussed to some extent, and I will avoid undue repetition in this section.

4.1.1 Consumers and Producers

Using the ns-3 Application class as its base, ndnSIM created the ns3::ndn::App class and from that, has created several node roles. Two of the most important are the Consumer and

the Producer classes. As the name suggests, objects of classes in the Consumer hierarchy request content by issuing Interest messages. Producers of course, generate the content to fulfill request.

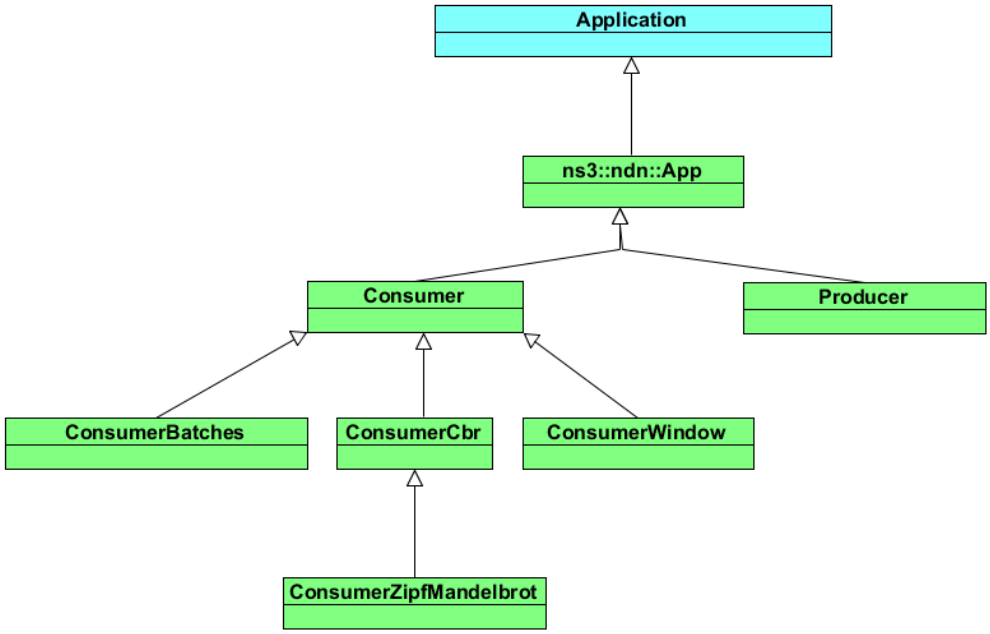

Figure 7: UML Domain-Level Class Diagram of App hierarchy

In Figure 7, the Application class is from ns-3. ndnSIM creates its own hierarchy by extending Application with the ns3::ndn::App class. The other classes in this hierarchy are also in the ns3::ndn namespace as well, but the scope resolution and namespace names have been removed in order to save space and avoid redundancy.

The descendants of Consumer are particularly noteworthy, especially ConsumerZipfMandelbrot. As the name suggests, this type of consumer is used to request content objects using a Zipf-like distribution. Various attributes can be applied to the consumers to control how they work. One such parameter in the ConsumerZipfMandelbrot is indicated by the character "s", which represents the α value [43], [44]. While Zipf's law implies a normalizing constant, Ω

is used in a distribution of items, where item i is ranked $\frac{\Omega}{i}$, a Zipf-like distribution results in items ranked by $\frac{\Omega}{i^{\alpha}}$. Content requests over the Web have been recorded as typically falling in a Zipf-like distribution with α ranging between 0.64 and 0.83. Therefore, ndnSIM researchers have typically set their α value to 0.6, 0.7, 0.8, or 0.9. For consistency in my research, I have recorded the data with Zipf's α value set to 0.8. And, although the formal experimentation has been done with 0.8, informal simulation runs have ranged from 0.6 to 0.9 with extremely similar results to the data collected in this work.

4.1.2 Adversarial Model

In addition to the major modifications mentioned earlier in this work, I added an additional attribute to the ConsumerCbr to support the adversarial model. Although an entirely new subclass could have been created, I chose to add a simple attribute to indicate whether the consumer was an attacker or not. If it was, then it would look for items to request from a special file, attack.txt, in the simulation directory in ndnSIM.

Because the attack is launched with a small subset of the overall content space, and the attacker attempts to move unpopular items into the cache from this set, it could be best classified as a false locality attack. The false locality attack is generally considered to be the more effective attack when compared to the locality disruption attack [45].

We assume a rather powerful attacker model, in that the attackers know the popularity distribution of the content space. Therefore, as mentioned, the false locality attack becomes possible. We assume that when a face is under attack, that the attacker(s) completely control that face. In other words, no non-attacker content is being requested through interests on that face.

Furthermore, in some of our experiments described later, we assume that attackers can control more than one face. The frequency with which the attackers send interests varies by the experiments being performed. These experiments typically range from six times the normal consumer at a rate of 720 (normal consumers request at 120 interests/second) to eight times the normal consumer at a rate of 960.

In the simulation scenarios that were run with multiple attackers, the coordination among attackers can be seen resulting from the shared set of items used in the attack. These are the items described above that come from the attack.txt file used in the simulation. Therefore, the theoretical maximum damage against a cache would be the size of the content object list used by the attackers. However, in practice, multiple variables affect this situation. For example, if the attacker content list was larger than the items that could be held in the content store, then the size of the content store would be the theoretical maximum.

Also, the range of unpopular content objects being requested can affect this situation as well. For example, if the attackers request items that are somewhat popular, then legitimate consumers as well as attackers contribute to the popularity of these items. This would have far less impact on the hit rate, since some of the items that may make their way into the cache would be requested by legitimate consumers.

Therefore, to give our attackers the best possible chance of causing the most damage possible, we have given them all benefits related to the simulation. The attackers are assumed to know the least popular contents, so that there would be little to no legitimate consumer contribution to their popularity. The attackers also have substantially faster request frequencies than typical users. Our tests do not assume any detection mechanisms, so that only the robustness created by the cache replacement schemes and associated algorithms affect the cache churn.

4.1.3 Simulation Scenarios

An important aspect of ndnSIM and ns-3 simulations is the concept of a *scenario*. A scenario is the actual client application of ndnSIM that sets up the simulation, and ultimately launches it. Features such as the frequency of consumer and attacker requests, the size of the content store, the policies to be used, and many others are typically set in the scenario. Additionally, the number of simulated seconds is set in a scenario as well.

Note that this does not necessarily correspond to actual seconds. In fact, it typically takes a significant amount more real time to simulate any given simulated time than what the simulated time implies. Because ndnSIM is a discrete event simulator, the real time it takes to run an application can vary significantly depending on the complexity of the data structures and algorithms employed.

For the experimentation, I employed three different scenarios:

- PFP-DA
- LFU-DA
- LRU

These scenarios correspond to the three algorithms for comparison. The PFP-DA is of course our cache replacement scheme, Per-Face Popularity with Dynamic Aging. The LRU (least recently used) is considered one of the most basic cache replacement algorithms and provides a look at how an algorithm that doesn't consider request frequency at all works. LFU-DA (least frequently used with dynamic aging) is an efficient LFU implementation that serves as a good comparison against our PFP-DA given that it does take frequency into consideration and should,

theoretically, protect against cache pollution and especially keep the most popular items in the cache.

4.2 Topologies

A multitude of topologies exist, including several custom topologies that could be tested. To test the focus of our PFP-DA against other algorithms, it was conducive to data gathering to devise a very simple single router scenario, as well as a more advanced scenario to see how the algorithms affect intermediate routers, particularly those that have no consumers directly attached to them. The data collected from the core routers and the edge routers in the slightly more advanced scenario can tell us a lot about the algorithms' effectiveness depending on the situation. Some information about where content storage routers can also be gleaned from the data as well.

To obtain the clearest picture possible given resource constraints, the topologies were kept relatively simple, but the scenarios applied a multitude of variations, including:

- Cache size as a percentage of the content space
- Request frequency (speed) of a single attacker across a single face
- Number of attackers requesting at same frequency across multiple faces

Each of these is important for determining the overall effect that each algorithm has on the cache pollution percentage and the hit rate. We wanted to know which, or if, varying these aspects of the scenarios would have a significant impact on any given algorithm. The impact (or lack thereof) gives us insight into the resistance of each algorithm against attack, as well as general performance – scenarios when there are no attackers.

As a reminder of how the Interests and Data messages work in NDN, an Interest message is sent into the network. If a (close) content storage router has the requested content cached in its content store, it responds with the associated Data message (containing the content object being requested.) Frequencies, recency, or other statistics are recorded as necessary depending on the algorithm being used.

However, if an Interest message cannot be satisfied by a given content router, the router inserts the Interest information into its PIT (Pending Interest Table.) If other Interest messages arrive for the same content object, and if the Data message has not arrived to satisfy the request yet, these subsequent Interest messages are stored in the PIT as well. Then, when the Data message finally arrives, it will satisfy the requests made from all different consumers on different faces.

An important side effect of the way this system works is that Interests for the same object are not forwarded further into the network if Interests for that object exist in the PIT. This is called *interest aggregation*. This prevents unnecessary congestion, but also means that since the Interests aren't forwarded, the cache replacement algorithms on routers beyond the routers connected directly to consumers (both legitimate consumers and attackers) will likely not be pushed to their limits to obtain metrics. Therefore, the routers connected to the most consumers are of particular interest.

Our findings were exciting and, in some cases, a bit unexpected. We will discuss more thoroughly the results later in this chapter. Needless to say, the PFP-DA algorithm produced interesting results both when under attack, and when no attackers were requesting content objects.

4.2.1 Simple Topology

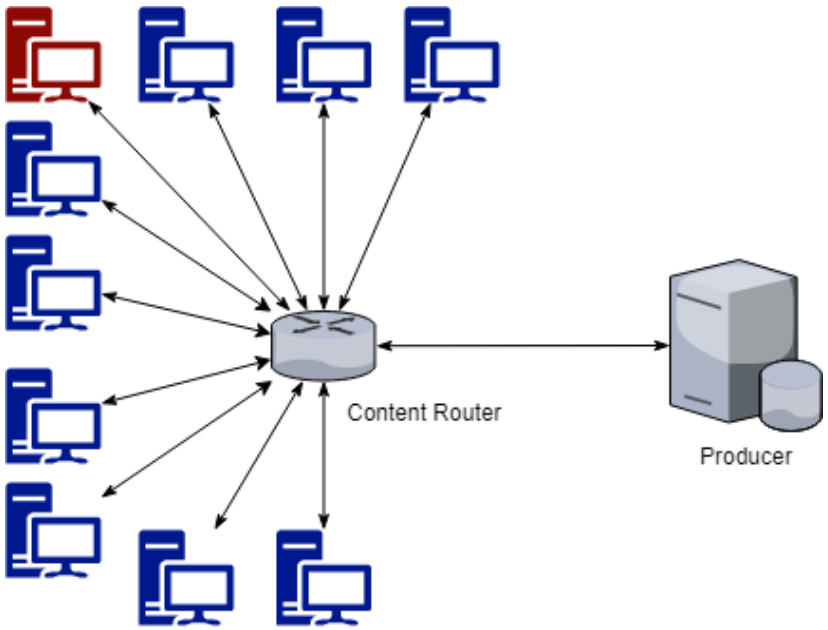

Figure 8: Simple Topology with Single Router and Ten Consumers

The network diagram of Figure 8 shows the simple topology (ST) consisting of ten consumers connected to a single content router, which is then connected to a content producer. Although this specific diagram indicates a single red consumer (an attacker), this is only true of the single-attacker scenarios used in testing. Larger number of attackers were used in other tests to see how strong the PFP-DA (and other algorithms) are against multi-face attacks.

Another important point to make is that each of these consumers could technically represent an aggregate set of incoming interests, and not just a single consumer. So, whether there are one or one hundred consumers sending requests over the same face, the situation is essentially the same. To simplify the discussion and the understanding of the scenarios, we view the relationship between consumer and face to be a one-to-one relationship.

We note that for the PFP-β tests later in this dissertation, a modified version of the simple topology, called the larger simple topology (LST) was used. This is identical to the ST, but with 20 consumers (including attackers) attached to the router instead of 10.

4.2.2 Advanced Topology

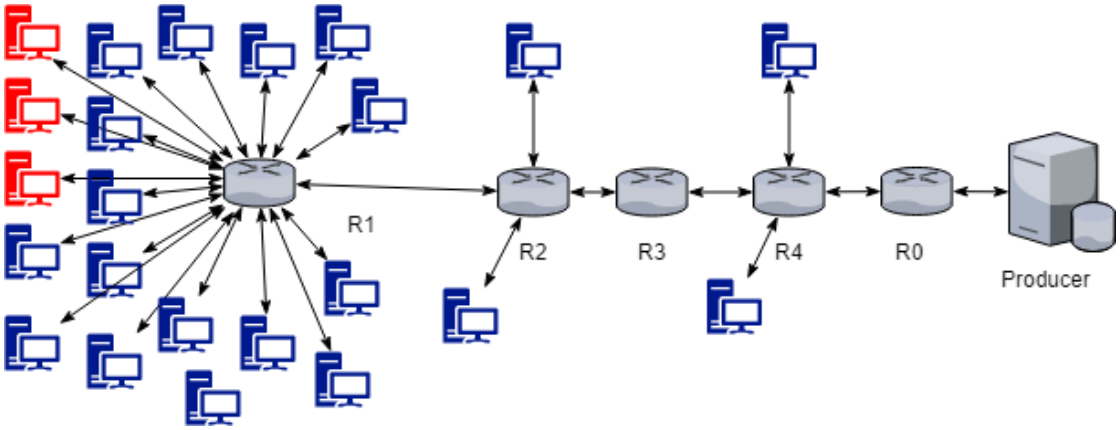

Figure 9: Advanced Topology with Five Routers, Twenty-Three Consumers, and a Producer

The topology modeled in Figure 9 (AT) is similar to the so-called *merging scenario* used in Xie's *CacheShield* test cases [36], with ours being slightly more complex with more consumers and routers. Again, although the three consumers with red color indicate three attackers leveraging a cache pollution attack against router R1, this is only one variation of the attacks that were launched.

We remain focused in our testing on a single router taking most of the Interest messages and which is attached to the most consumers. However, in this scenario, unlike the simple topology, we sprinkle a few consumers down the path toward the producer, and leave some routers as content storage routers, but not directly attached to any consumers (e.g., R3, R0.)

This advanced topology can give us interesting insights into the effects of the various cache replacement algorithms at each of the intermediate routers, as well as the edge routers. It can also help us decide what routers might benefit from being given caches, and which do not provide as much benefit.

The routers connected directly to many consumers (in our case, R1) will provide the most meaningful data. This is due to many factors but is especially true because of interest aggregation (described earlier.)

4.3 Results

In this section, we display and discuss the experimental results. As mentioned before, we test the topologies running the three different algorithms against various cache sizes, various attacker request frequencies (while the legitimate consumer request frequencies remain the same), and various number of attackers. Note that the baseline metrics consist of a single attacker, 720 frequency of the attacker vs 120 frequency of a legitimate consumer, and 100 cache size (1%).

In each subsection, we examine the results of modifying the single variable in that subsection and explore the implications of the data collected to determine how each algorithm performs under the different circumstances.

The content space used in these tests consists of 10,000 content objects, with sequence numbers 1 through 10,000.

4.3.1 Varying Cache Size

In this set of tests, there was one attacker for both the simple and advanced topologies, requested at a frequency of 720 requests per second, with regular consumers requesting at 120 requests per second. The only item that was varied was that of the cache size, ranging from 100 (1%) to 500 (5%) out of the overall content space size of 10,000. The number of items requested by an attacker is equal to the cache size to make it possible for the attacker to try to completely overwhelm the cache (e.g., if all consumers attached to a router were attackers and the cache size was 500, the attackers could completely fill the cache with unpopular items.)

Algorithm	Cache Size	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%

Table 2: ST with Varying Cache Size, Single Attacker at 720 Frequency

The Pollution % metric indicates the percent of the cache at the router occupied by attacker content. The Hit % is the overall hit percentage (hit rate) over all requests. To determine if the algorithms are effective at maintaining the most popular items in the caches, we look at the top 100 and top 50 items, according to the sequence number, which reflects the Zipf-like distribution. In other words, how good are the algorithms at maintaining the truly popular contents in the caches?

The PFP-DA cache pollution percentage holds well against the single attacker across all cache sizes, decreasing the percentage of pollution from 15.00% with a cache size of 100, to 12.40% with a cache size of 500. The other two algorithms, LFU-DA and LRU all experience pollution increases. With LFU-DA, this is expected since the frequency of the attacker is reasonably aggressive compared to the nine legitimate consumers. LFU-DA experiences a slight drop with a 500 limit of its cache. With LRU, which is not based on frequency, the attacker's frequency likely increases the probability that during a snapshot of the cache contents, that attacker content will populate the cache during the frequent and continuous churn.

Even more striking, arguably, is with the hit percentages. All three algorithms experience increases in hit percentage as the cache size increases. This is to be expected, since the caches can accommodate a larger portion of the content space and therefore this increases the probability that an item being requested will be in the cache.

The overall percentage of PFP-DA across all cache sizes is better than both LFU-DA and LRU. Because LFU-DA does attempt to maintain the most popular items in the cache, it performs better than LRU in each of the different cache size variation tests. However, PFP-DA holds a 6.35% to 9.05% better overall hit percentage than the LFU-DA algorithm.

The difference of these algorithms becomes even more evident when we examine the top 100 and top 50 hit percentages. Again, while all three algorithms experience top 100 and top 50 item hit percentage improvement as the cache size increases, for top 100, PFP-DA maintains a 21.63% lead on LFU-DA for 100 content items (1%), and then a 27.19% lead for 200 content items (2%), 25.09% lead for 300 content items cache (3%), and 20.07% for 500 content items cache (5%.) For the top 50 items, PFP-DA maintains a range of 17.93% (for 500 content items cache,

or 5% of the content space) to 28.80% (for 200 content items cache, or 2% of the content space) lead over LFU-DA. This is substantial, under all conditions.

Now, we observe the results from the advanced topology for pollution and hit rates.

			Pollution %				
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%

Table 3: AT with Varying Cache Size, Single Attacker, 720 Frequency - Pollution %

				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%

Table 4: AT with Varying Cache Size, Single Attacker, 720 Frequency - Hit %

For this topology, there are five routers to consider. Therefore, the tables have been split to accommodate the data.

Table 3 and

Table 4 demonstrate that overall cache pollution percentage does not always correlate with a higher hit rate. For example, LRU has lower percentage rates on router R0 for all cache sizes from 200 up to 500. However, when we observe the hit percentage of LRU at this router is 0% across all router sizes. And while the other algorithms don't have particularly high percentages, they at least register with some. PFP-DA, as low as the R0 percentage is, dominates LFU-DA.

We observe similar pollution percentage situations on routers R2 and R3, where LRU has less cache pollution percentage but substantially lower hit rate than both LFU-DA and PFP-DA. What causes this seeming contradiction? Although lowering the cache pollution percentage is a goal, it is important to keep two aspects of this data in mind: 1.) the data is derived from a snapshot in time at a given point in the simulation (i.e., the end of the simulation), and 2.) the overall percentage of pollution says nothing of *which* items remain in the cache. LRU likely has mostly less popular items in the cache than both LFU-DA and PFP-DA.

This is confirmed by the hit percentages across all routers and all cache sizes. PFP-DA always performs better than LFU-DA, and LFU-DA always performs better than LRU using the hit rate metrics.

Since R1 is the router with the most consumers attached to it, we can glean the most useful information about the algorithms' performance metrics. Again, like in the simple topology tests, PFP-DA maintains a substantial lead on LFU-DA, and even more significant lead over LRU. This is particularly obvious, once again, when we observe the hit rates of the top 100 and top 50 items on router R1. PFP-DA demonstrates resilience against the attack and maintains a substantially

higher percentage of popular content objects, especially from the top 100 and top 50, than do either LFU-DA or LRU.

4.3.2 Varying Attacker Request Frequencies (Single Attacker)

To determine the effect of varying the attacker request frequencies on the cache pollution and hit rate, we maintained one attacker, with a size 100 (1%) cache, and consumers with a request frequency of 120. The attacker request frequency (attacker speed) was varied to determine the abilities of the three algorithms when being attacked by an aggressive 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%

Table 5: ST with Size 100 Cache, Single Attacker with Varying Request Frequency

With the simple topology, we observe in

Table 5 that the pollution percentage of the cache running PFP-DA remains essentially unaffected by the change in attacker speed. The LFU-DA and LRU, on the other hand, are directly impacted by this increased aggression against their content stores and grow quickly.

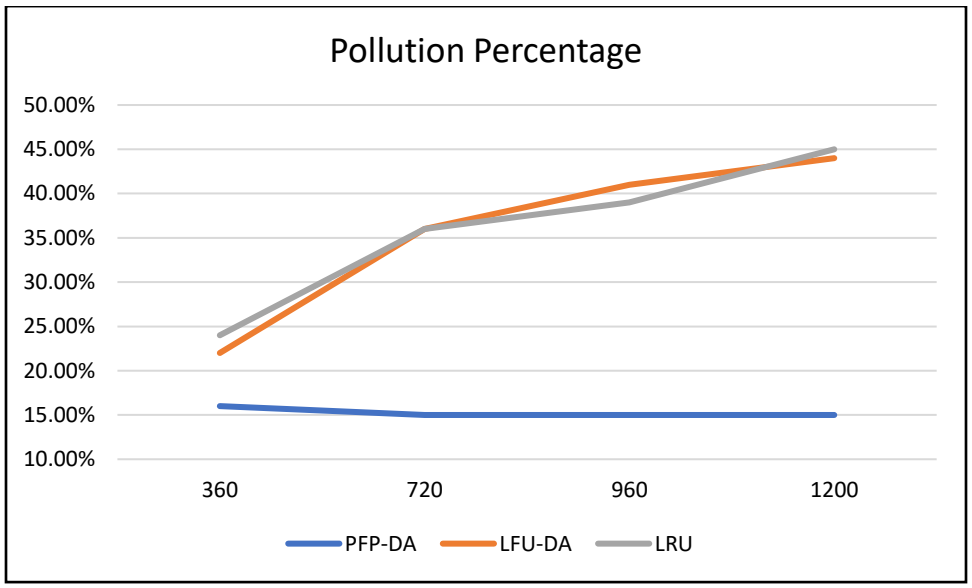

Figure 10: ST with 100 Cache Size, Single Attacker, Varying Attacker Request Frequency Pollution %

Graphically, we can see in Figure 10 that varying the attacker's request frequency has virtually no impact on the cache running PFP-DA. However, we can also see that LFU-DA and LRU are both impacted significantly by the attacker's frequency, with their pollution percentages increasing steadily as the attacker's frequency of requests increases from 360 to 1200.

The phenomenon exhibited by PFP-DA is clearly related to its normalization of the per-face contributions to the overall popularity, thus affecting the overall cache. Since only one attacker (or more correctly, one face of attackers) is present, the attack frequency on that face has no impact on the contribution of that face. All other faces (all other consumers) have equal contributions, even though their request frequencies are substantially less.

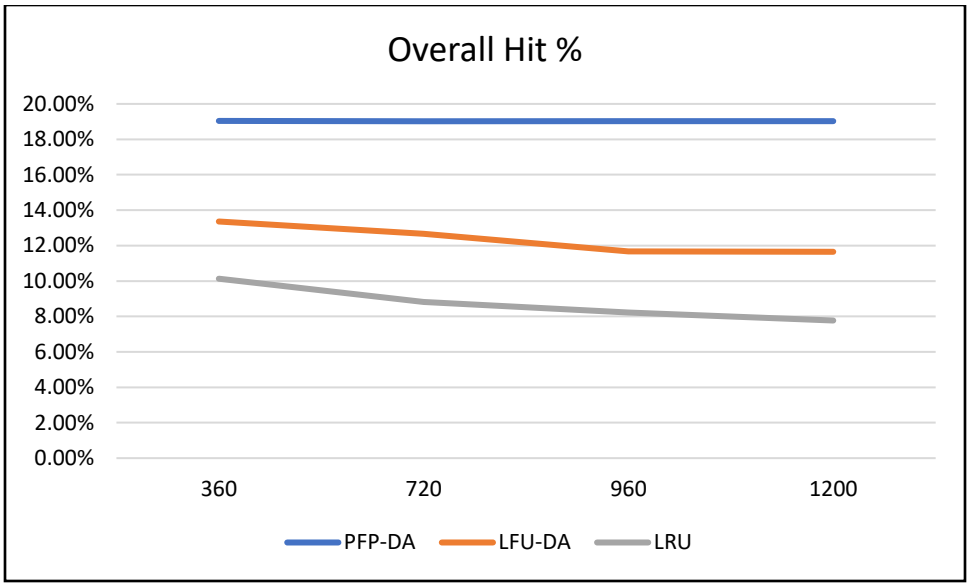

Figure 11: ST with 100 Cache Size, Single Attacker, Varying Attacker Request Frequency Overall Hit %

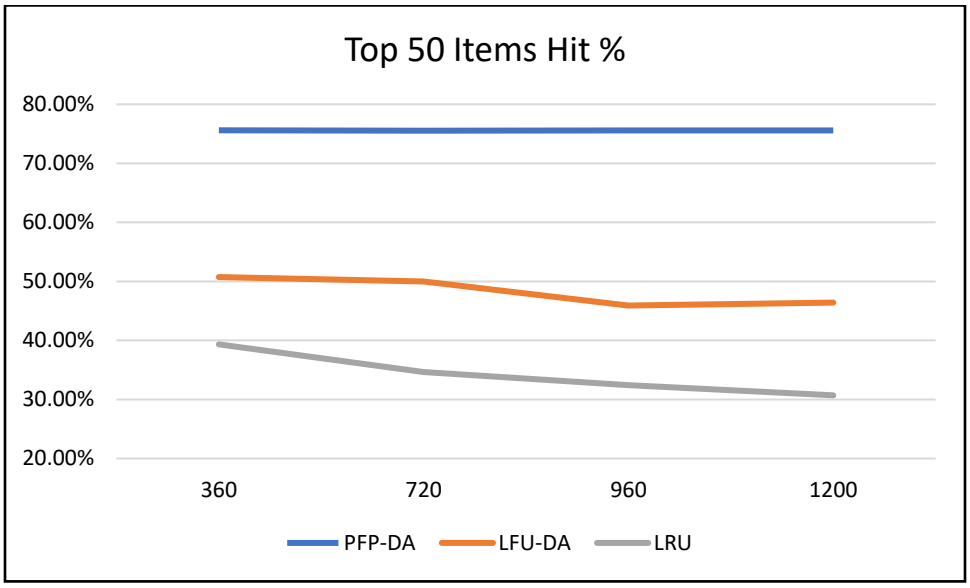

Figure 12: ST with 100 Cache Size, Single Attacker, Varying Attacker Request Frequency Top 50 Items Hit %

Both Figure 11 and Figure 12 clearly demonstrate PFP-DA's superiority across all attack speeds in terms of hit rate (hit %.) LFU-DA does a reasonably good job of maintaining the hit rate, since it is frequency-based, but it is affected somewhat by the increase in attacker frequency.

However, LRU clearly does not protect against the attacker's requests for unpopular contents, as its hit rate decreases steadily as the attacker's request frequency increases.

Interestingly, not only does PFP-DA do the best job at maintaining its hit rate as the attacker's frequency increases, it also starts with a substantially better hit rate the LFU-DA. PFP-DA starts with around 5% better overall hit rate than LFU-DA, and 25% better hit rate the top 50 items. The resistance to the effects of the cache pollution attacks in these scenarios is significant, and the results clearly demonstrate PFP-DA's dominance with a simple topology and single attacker.

Let us now consider the AT (Advanced Topology) tests with a single attacker and variation of only the request frequency.

		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 6: AT with Size 100 Cache, Single Attacker with Varying Request Frequency - Pollution %

For the pollution percentage on the advanced topology, we focus on R1, as it is the router that is directly attached to the most consumers. We see that under these conditions, the R1 running

PFP-DA experiences the least cache pollution (8% throughout all attacker frequency tests.) However, the values vary greatly for both LFU-DA and LRU. LFU-DA experiences an overrun of pollution as the frequency increases, but LRU appears to be all over the place. However, even looking at all other routers, LRU has better (lower) cache pollution percentages than LFU-DA, and even in most cases, PFP-DA.

At any given time, the LRU cache will have whatever items were most recently requested. Although it appears to be naturally resistant against cache pollution, it is equally "resistant" against maintaining popular items in the cache. This is clear from the hit rate statistics from earlier, as well as below.

		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%

Table 7: AT with Size 100 Cache, Single Attacker with Varying Request Frequency - Hit %

Again, in

Table 7, we observe what we've seen before. The hit rates are quite low for the routers R0, R2, R3, and R4. This is largely due to them not being connected directly to any, or in the cases of R2 and R4, not many consumers. From the perspective of network design, the routers with very low hit rates could just be regular non-storage routers since their contributions are mostly negligible for all three algorithms.

Therefore, we once again turn our attention to the router R1. Running PFP-DA, R1 maintains around 16.6 – 16.9% hit rate throughout all attack frequencies. LFU-DA appears to have reasonably good resistance to the attack, even increasing its overall hit rate when attacked with 1200 frequency from the attacker. Due to the minimal variation, this indicates that the algorithm is maintaining the percentage, and the variation itself is due to natural fluctuations between scenario launches.

However, there is still no question – PFP-DA maintains the highest percentage of overall hit rate. When we consider the top 50 items, PFP-DA has anywhere from 20-30% better hit rate than LFU-DA. For the top 50, we can also see that LFU-DA, especially from the change of frequencies of 360 to 720, drops in hit rate substantially. In this case, this appears to be more than just simple variation between scenario launches, although it certainly contains some variation due to this phenomenon. LFU-DA cannot fully resist the flooding of requests from the attacker.

4.3.3 Varying Number of Attackers

To further our understanding of how the three algorithms we are testing work under different conditions, we turn our attention to varying the number of attackers. In the scenarios used for these tests, the cache size is held at 100 (1%), the attackers' request frequency is 720 requests per second, with regular consumers requesting at 120 requests per second.

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%

Table 8: ST with Size 100 Cache, 720 Attack Frequency, with Varying Number of Attackers

The data in **Error! Reference source not found.** for the Simple Topology gives us insight into the typical performance of the algorithms, including when there are no attacks occurring. In the tests involving no attackers, the pollution, as expected, is 0%.

For the ST, the number of attackers is out of the total number of consumers, which is 10. Therefore 1 attacker is 10%, 3 attackers is 30%, and so on.

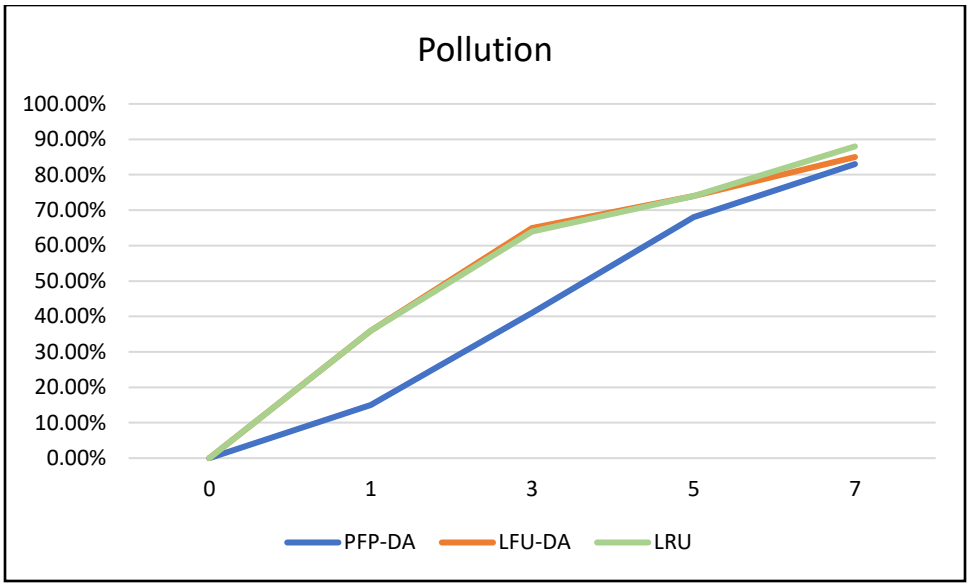

Figure 13: ST with 100 Cache Size, 720 Attacker Frequency, Varying Number of Attackers Pollution %

Graphically, we can observe that all three algorithms are susceptible to being overwhelmed when the number of attackers is great enough. However, we observe that LFU-DA and LRU don't have substantial differences between them in this set of tests. In terms of pollution, PFP-DA performs substantially better from the single attacker scenario with 10% compromised nodes, up to about the 7 attacker (70% compromised nodes) scenarios.

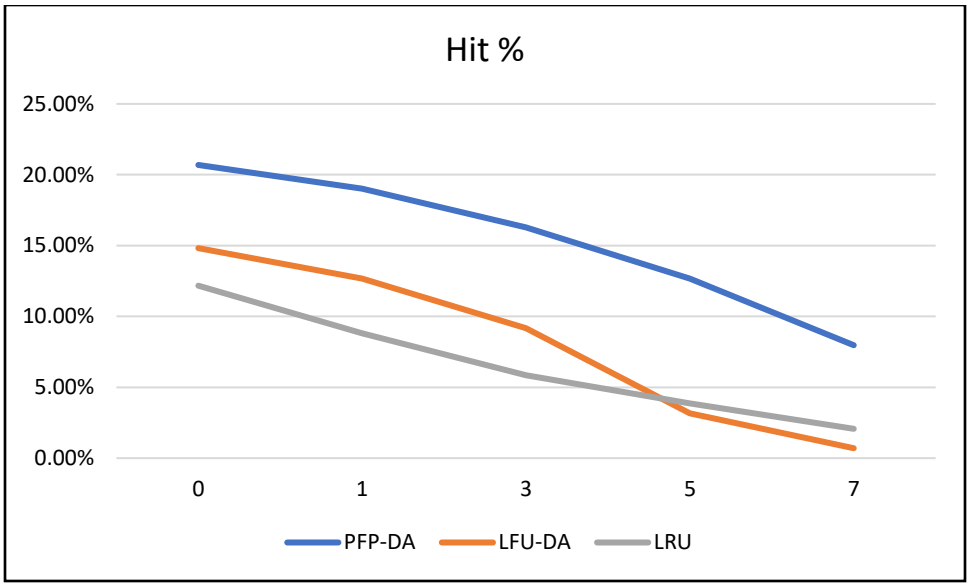

Figure 14: ST with 100 Cache Size, 720 Attacker Frequency, Varying Number of Attackers Hit %

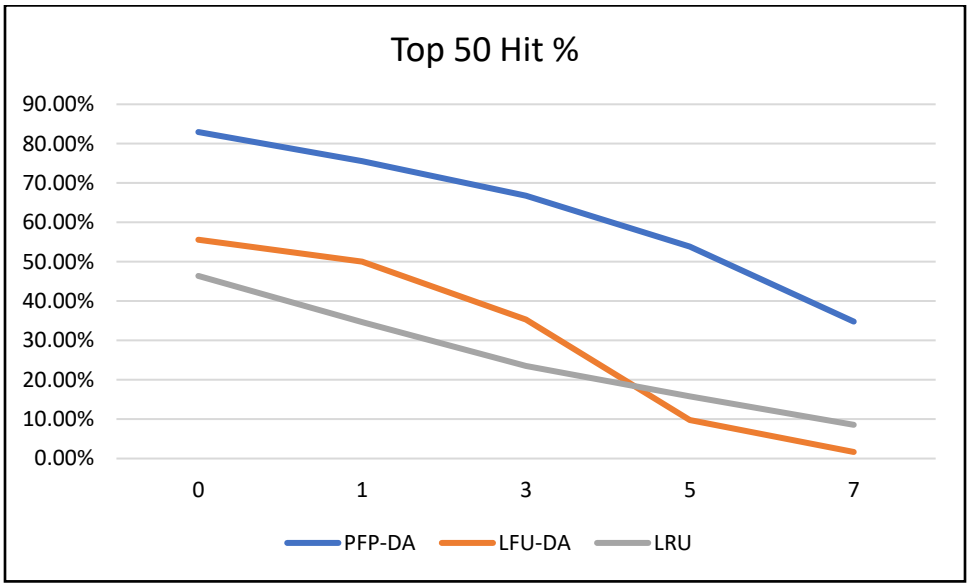

Figure 15: ST with 100 Cache Size, 720 Attacker Frequency, Varying Number of Attackers Top 50 Hit %

As expected, in both Figure 14 and Figure 15, all three of the algorithms show hit rates that fall as the number of attackers increase. LFU-DA appears to plummet as 5 attackers (50%) overwhelm the algorithm. Although PFP-DA clearly suffers as the number of faces being attacked

increases, the drop is steady. LFU-DA experiences becomes overwhelmed because the sum of the frequencies of the attackers, regardless of the face on which the attacks arrive, matter. Since LFU-DA has no normalization of the per-face contributions, the sum of multiple attackers across multiple faces, or a single attacker with a very aggressive request frequency would all cause a massive overwhelming of the algorithm. For example, at the 5 attacker (50%) mark, LFU-DA feels the entire brunt of the attackers, with 720 * 5 = 3,600 requests per second being for attacker contents. This is six times as much as the requests from all the legitimate consumers, that is, 5 * 120 = 600 requests per second. This results in over 85% of the total requests received coming from attackers, even though the attackers make up only 50% of the total consumers.

PFP-DA, on the other hand, experiences a more near-linear decline with respect to the number of attackers. This is due to the normalization that takes place upon the contribution of each of the faces, which scales the contributions so that the overall contribution of requests on any given face does not substantially diminish the contributions of requests on any other face.

Before continuing with a different topology, we discuss the per-item hit rate briefly, in the scenario with no attackers. These results were very impressive as they demonstrate the effectiveness of PFP-DA at expressing the true popularity of items, normalized across all faces. The results from the recent attacker-variation graphs and tables clearly show PFP-DA is superior to both LFU-DA and LRU by about 6% better hit rate overall, 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.

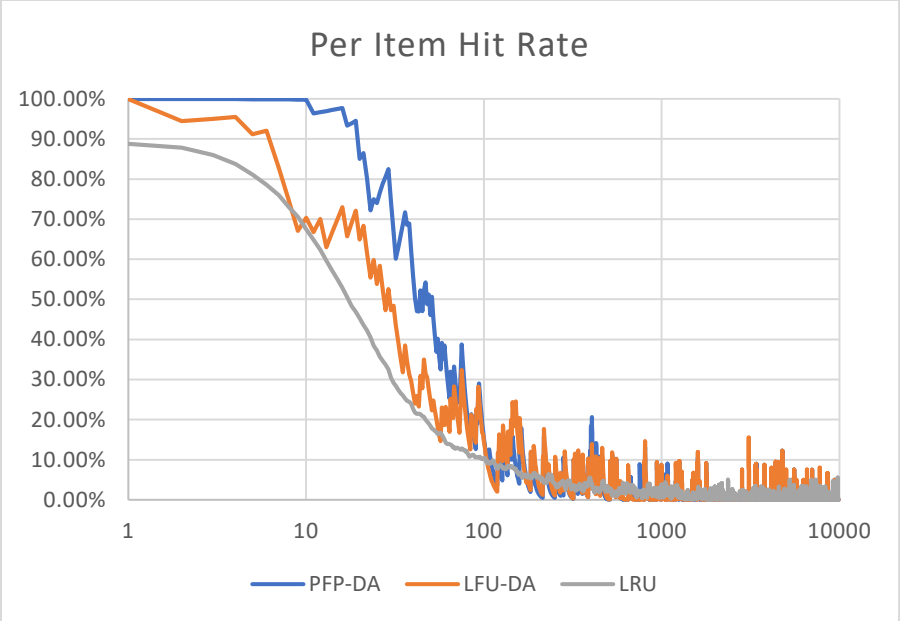

Figure 16: Per-Item Hit Rate, Zero Attackers, Simple Topology

The results are displayed graphically in Figure 16, which charts the moving average with an x-axis logarithmic scale. Clearly, PFP-DA outperforms LFU-DA and LRU for the top content objects, maintaining a very strong hit rate until around the 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.

Now, we move our attention to the Advanced Topology (AT). For the AT, we consider the total number of attackers as being out of 22 total consumers, but since all of them are connected to R1, we are mostly concerned with this percentage. Therefore, the calculated percentage is based on the number of total consumers attached to the router R1.

			Pollution %				
Algorithm	**#Attackers**	**% Attackers on R1**	**R0**	**R1**	**R2**	**R3**	**R4**
PFP-DA	**0**	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
LFU-DA	**0**	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
LRU	**0**	**0.00%**	0.00%	0.00%	0.00%	0.00%	0.00%
PFP-DA	**1**	**5.26%**	18.00%	8.00%	23.00%	16.00%	8.00%
LFU-DA	**1**	**5.26%**	17.00%	22.00%	17.00%	22.00%	14.00%
LRU	**1**	**5.26%**	18.00%	18.00%	17.00%	19.00%	17.00%
PFP-DA	**3**	**15.79%**	39.00%	30.00%	25.00%	25.00%	14.00%
LFU-DA	**3**	**15.79%**	39.00%	46.00%	37.00%	42.00%	34.00%
LRU	**3**	**15.79%**	14.00%	32.00%	16.00%	17.00%	14.00%
PFP-DA	**5**	**26.32%**	50.00%	44.00%	27.00%	30.00%	16.00%
LFU-DA	**5**	**26.32%**	46.00%	67.00%	46.00%	47.00%	46.00%
LRU	**5**	**26.32%**	25.00%	52.00%	28.00%	24.00%	23.00%
PFP-DA	**10**	**52.63%**	66.00%	77.00%	17.00%	32.00%	25.00%
LFU-DA	**10**	**52.63%**	67.00%	78.00%	63.00%	67.00%	61.00%
LRU	**10**	**52.63%**	22.00%	73.00%	32.00%	32.00%	25.00%

Table 9: AT with Size 100 Cache, 720 Attack Frequency, with Varying Number of Attackers – Pollution %

			Hit %					R1 Special Hit %	
Algorithm	**#Attackers**	**% Attackers on R1**	**R0**	**R1**	**R2**	**R3**	**R4**	**top 100**	**top 50**
PFP-DA	**0**	**0.00%**	0.46%	17.52%	4.21%	1.29%	2.61%	69.71%	84.75%
LFU-DA	**0**	**0.00%**	0.06%	13.15%	1.95%	0.44%	2.41%	51.08%	64.39%
LRU	**0**	**0.00%**	0.00%	8.30%	1.28%	0.07%	1.21%	31.20%	38.16%
PFP-DA	**1**	**5.26%**	0.55%	16.78%	4.97%	1.56%	2.52%	67.17%	81.07%
LFU-DA	**1**	**5.26%**	0.01%	10.78%	2.19%	0.58%	2.35%	42.61%	53.34%
LRU	**1**	**5.26%**	0.00%	6.89%	1.20%	0.05%	1.19%	26.05%	31.98%
PFP-DA	**3**	**15.79%**	0.54%	16.13%	4.37%	0.99%	2.83%	64.87%	79.22%
LFU-DA	**3**	**15.79%**	0.00%	9.25%	2.04%	0.76%	2.20%	36.44%	45.02%
LRU	**3**	**15.79%**	0.00%	6.54%	1.42%	0.08%	1.27%	24.67%	30.50%
PFP-DA	**5**	**26.32%**	0.53%	14.74%	4.70%	1.67%	3.22%	60.95%	75.69%
LFU-DA	**5**	**26.32%**	0.04%	7.62%	3.08%	0.83%	2.18%	31.38%	40.66%
LRU	**5**	**26.32%**	0.00%	4.65%	1.57%	0.08%	1.33%	17.95%	22.38%
PFP-DA	**10**	**52.63%**	0.36%	10.22%	6.94%	1.96%	4.08%	42.28%	55.36%
LFU-DA	**10**	**52.63%**	0.00%	7.07%	1.97%	1.32%	2.44%	27.18%	34.60%
LRU	**10**	**52.63%**	0.00%	3.14%	2.19%	0.16%	2.10%	11.29%	13.87%

Table 10: AT with Size 100 Cache, 720 Attack Frequency, with Varying Number of Attackers – Hit %

The results in Table 9 show the effect of increasing the number of attackers in the AT test scenarios. Not surprisingly, we again see that 0% attackers yields 0% cache pollution. Once we increase this to 1 attacker, we have our base scenario that we've seen before. The cache pollution for all three algorithms increases on R1 as the number of attackers increases. Until the 10-attacker scenario, PFP-DA holds a substantially lower percentage than in the LFU-DA and LRU. For other routers, this is less predictable. For example, R0 indicates a very low cache pollution percentage for LRU when compared with both LFU-DA and PFP-DA. PFP-DA, overall has the worst cache pollution percentage on R0. However, none of the pollution percentages give us a qualitative look at the cache.

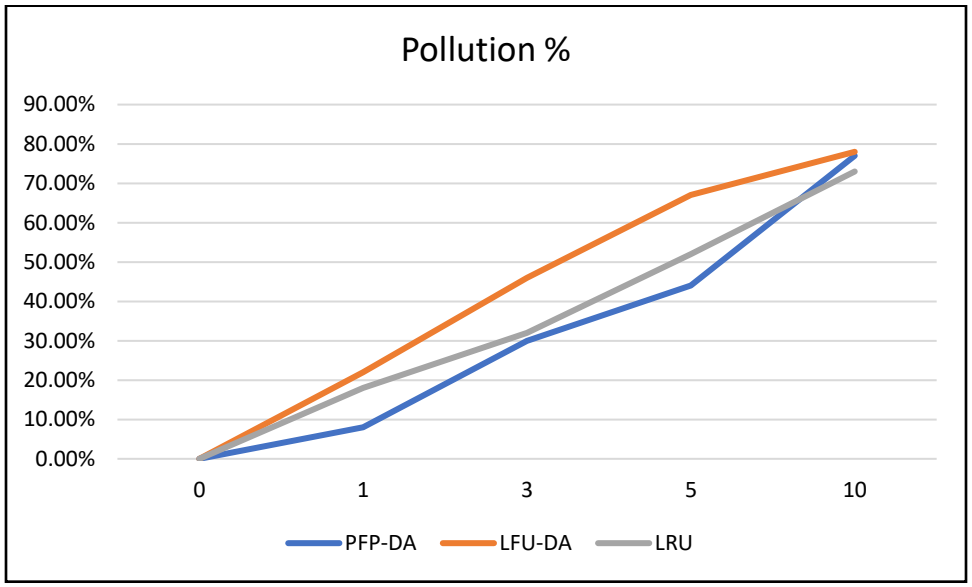

Figure 17: AT with 100 Cache Size, 720 Attacker Frequency, Varying Number of Attackers Pollution %

As we've seen before, the hit rate is what really gives us insight into the qualitative aspect of the items that are being cached. Thus, in Table 10 we see that PFP-DA dominates every router in every test set. Again, R1 is the most important router since it is the one with the most consumers (including all attackers.)

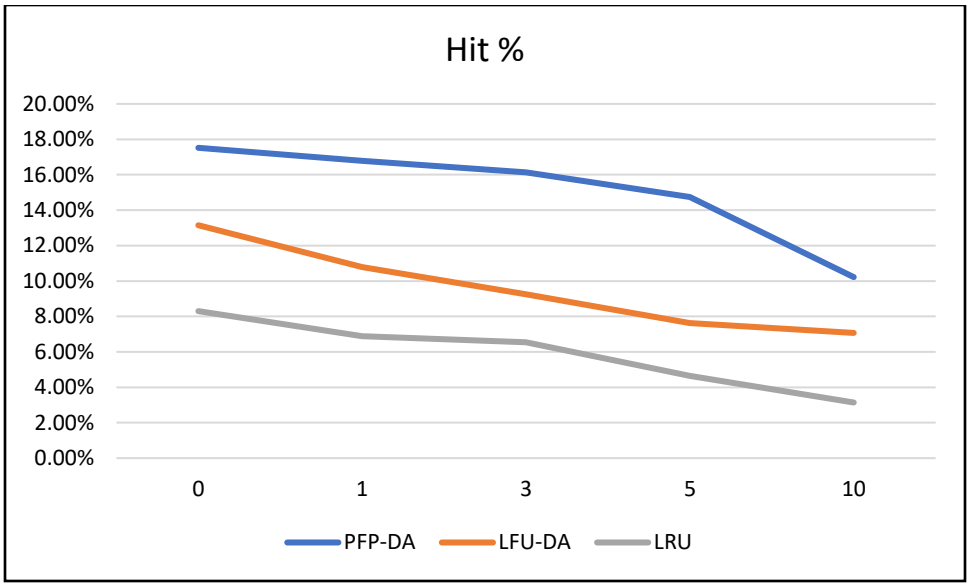

Figure 18: AT with 100 Cache Size, 720 Attacker Frequency, Varying Number of Attackers Overall Hit %

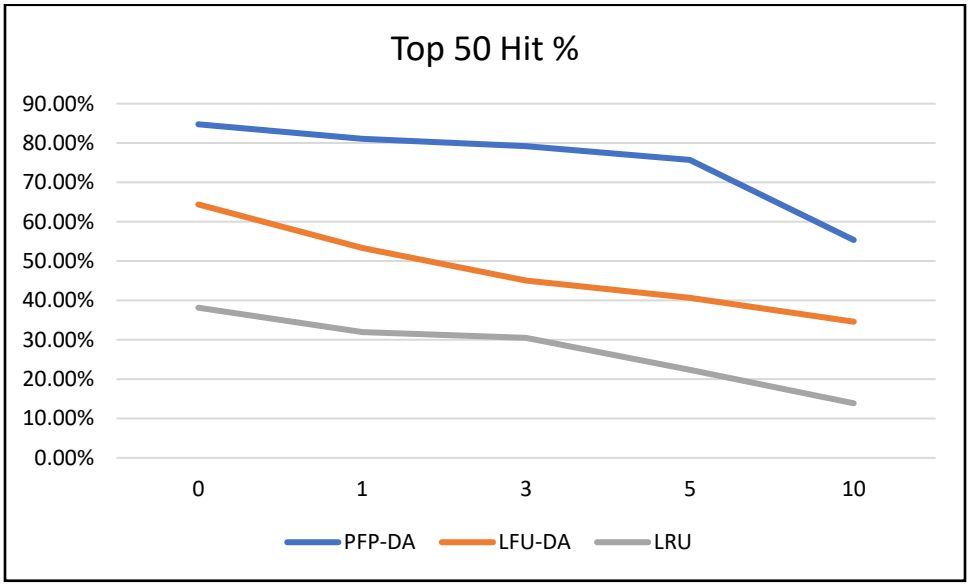

Figure 19: AT with 100 Cache Size, 720 Attacker Frequency, Varying Number of Attackers Top 50 Hit %

Graphically, the difference of hit rates is most obvious. Both Figure 18 and Figure 19 show a tremendous advantage when PFP-DA is compared against LFU-DA and LRU. We see that in both the overall and top 50 hit rates that PFP-DA seems to hold very steadily until approximately

50% (10) of the total consumers are attackers. At this point, PFP-DA performs less impressively against the LFU-DA and LRU, but still maintains a significantly higher hat rate.

4.3.4 Varying Attacker Request Frequencies (Three Attackers) on Advanced Topology

We performed some additional tests to gain more insight into the attacks and their effects. For this series, we used a combination of a constant number of attackers greater than one (3), while also varying attacker request frequencies.

		Pollution %				
Algorithm	**Attacker Speed**	**R0**	**R1**	**R2**	**R3**	**R4**
PFP-DA	720	39.00%	30.00%	25.00%	25.00%	14.00%
LFU-DA	720	39.00%	46.00%	37.00%	42.00%	34.00%
LRU	720	14.00%	32.00%	16.00%	17.00%	14.00%
PFP-DA	840	40.00%	29.00%	26.00%	28.00%	14.00%
LFU-DA	840	41.00%	39.00%	44.00%	44.00%	39.00%
LRU	840	17.00%	33.00%	16.00%	14.00%	13.00%
PFP-DA	960	42.00%	26.00%	26.00%	28.00%	16.00%
LFU-DA	960	39.00%	35.00%	42.00%	45.00%	38.00%
LRU	960	19.00%	29.00%	19.00%	15.00%	16.00%

Table 11: AT with Size 100 Cache, Varying Attack Frequency, with 3 Attackers – Pollution %

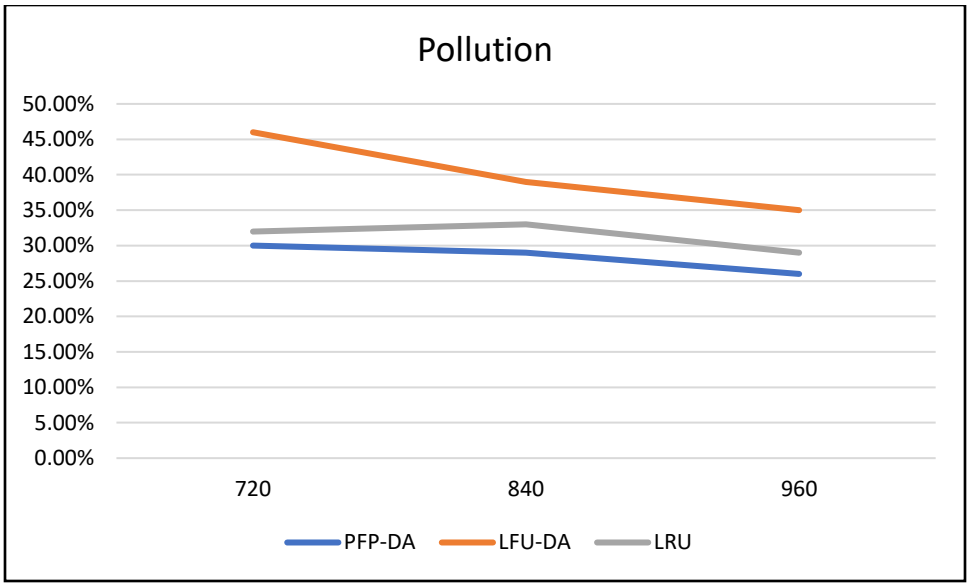

Figure 20: AT with Size 100 Cache, Varying Attack Frequency, with 3 Attackers – Pollution %

In terms of the pollution percentage seen in Table 11 and Figure 20, all three algorithms seem to remain quite steady as the attacker speed is increased on all the routers. On R1, we see that PFP-DA does maintain the lowest pollution percentage, but it is not by a substantial amount when compared with LRU. However, it is significant against LFU-DA, which matters more in this type of comparison since it is frequency-based.

				Hit %			R1 Special Hit %	
Algorithm	**Attacker Speed**	**R0**	**R1**	**R2**	**R3**	**R4**	**top 100**	**top 50**
PFP-DA	720	0.54%	16.13%	4.37%	0.99%	2.83%	64.87%	79.22%
LFU-DA	720	0.00%	9.25%	2.04%	0.76%	2.20%	36.44%	45.02%
LRU	720	0.00%	6.54%	1.42%	0.08%	1.27%	24.67%	30.50%
PFP-DA	840	0.58%	16.38%	4.31%	0.87%	2.82%	65.65%	79.83%
LFU-DA	840	0.00%	9.77%	1.48%	0.28%	1.67%	38.21%	48.03%
LRU	840	0.00%	6.28%	1.46%	0.08%	1.26%	23.80%	29.37%
PFP-DA	960	0.66%	16.44%	4.34%	0.88%	2.78%	65.81%	79.92%
LFU-DA	960	0.02%	9.27%	2.32%	0.64%	1.87%	36.13%	44.33%
LRU	960	0.00%	6.19%	1.45%	0.08%	1.27%	23.46%	28.98%

Table 12: AT with Size 100 Cache, Varying Attack Frequency, with 3 Attackers – Hit %

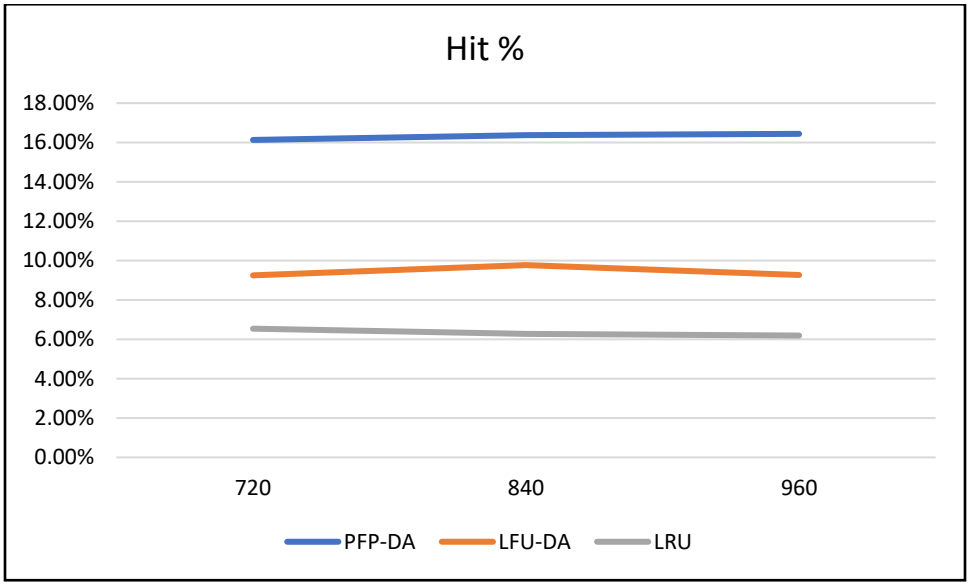

Figure 21: AT with Size 100 Cache, Varying Attack Frequency, with 3 Attackers – Hit %

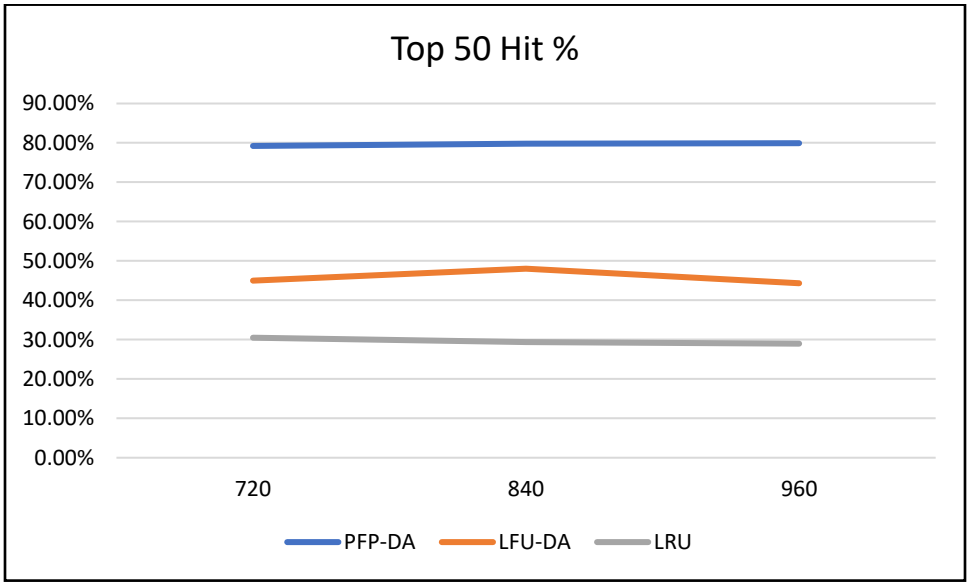

Figure 22: AT with Size 100 Cache, Varying Attack Frequency, with 3 Attackers – Top 50 Hit %

The combination of more attackers and increasing attacker request frequency as seen in

Table 12 as well as in Figure 21 and Figure 22 seems to have very little effect on the hit rates for any of the algorithms as far as considering the attacks across the three frequencies. However, from the very beginning, PFP-DA shows superiority against the other algorithms. This is especially clear on R1, and with the top 100 and top 50 hit rates. PFP-DA maintains a better hit rate by over 30% against LFU-DA throughout each test set.

Chapter 5
Toward Zero Cache Pollution

5.1 Parameterized Exponentiation of Rankings in Popularity Contributions: PFP-β

Let us consider Equation 2 again.

$$P(C) = \sum_{i=1}^{n} \frac{1}{r_i} = \frac{1}{r_1} + \frac{1}{r_2} + \cdots + \frac{1}{r_{n-1}} + \frac{1}{r_n}$$

This equation calculates the overall popularity of a content object *C*. If we consider the contributions from each of the faces, recall that we use the r_i of the content object *C* from each given face *i*. The r_i represents the rank of the given content object *C* in face *i*'s popularity ranking list. For example, if face 1 has *C* ranked third in its popularity ranking list, then its contribution to P(*C*) is $\frac{1}{3}$. The fourth ranked item would contribute $\frac{1}{4}$, and so on.

From each face, therefore, contributions resembling that of a Zipf distribution are observed. Thus, as the rankings move farther away from the top item, the contribution of an individual ranking becomes less. To reduce the difference of rank contributions of the top item and subsequent items, let us consider modifying the overall popularity equation to raise each rank contribution value to β, a value ranging from 0 to 1. Although α could be used, this would likely cause confusion with the Zipf-like distribution used as a parameter in ndnSIM and in the discussion of consumer request patterns.

So, our equation now becomes:

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

Equation 6: Parameterized Exponentiated Overall Popularity

Here, the lower the β, the smaller the difference between the per-face contribution for each item. With small β values, there is a reduction in the difference between any given ranked item and the top ranked item, and contributions from multiple faces are favored as they contribute larger amounts, overall.

To ensure zero or very low pollution percentage, we could set β to be very small (e.g., 0.1.) Thus, the PFP contribution differences between the top ranked items and lower ranked items becomes small. A low ranked item may still be cached if it receives contributions from multiple faces. On the other hand, even the top ranked items of a single attacker may not be cached due to the contributions for those items having come from a single or very few faces.

To obtain some initial data about what the addition of this parameter might have on preventing cache pollution and increasing hit rate, we ran some preliminary tests. The tests were performed with a single attacker on the Simple Topology (ST), with the attacker requesting at a frequency of 720 requests per second, with a cache size of 100 (1%) and only varying the beta value. Furthermore, we only performed these tests on PFP-DA since this is the only algorithm in which β has meaning.

Note that our original Equation 5 essentially is a special case of Equation 6 with β set to 1. In other words, we ran a set of truly Zipf-like contributions. Values of β less than 1 mark a Zipf-like distribution of contributions.

Beta Value	Pollution %	Hit %	top 100	top 50
1.00	14.00%	19.60%	65.04%	78.44%
0.90	15.00%	19.90%	66.07%	79.44%
0.70	14.00%	20.43%	68.03%	81.38%
0.50	12.00%	21.43%	71.22%	84.51%
0.30	8.00%	22.71%	75.42%	88.76%
0.10	4.00%	23.30%	76.96%	89.88%
0.05	4.00%	23.32%	77.01%	89.92%

Table 13: Early Results of Varying β Value

Although this modification of PFP is preliminary, it indicated impressive early results and implied that lower pollution and higher hit rate are possible. Table 13 shows at the top, the β value equal to 1.00, which is the case used throughout most of the research for this dissertation, as PFP-DA can be seen now as a special case of PFP-β. Although the pollution and hit rate are both good, they generally improve as the β value is lowered. The most drastic change in pollution is noticed from 0.50 to 0.10 and 0.05. The pollution lowers dramatically from 12.00% to 4.00%, and the hit rate also increases, both in terms of the overall hit rate, but also the top 100 and top 50.

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

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%

Table 14: PFP- β with single attacker, cache size of 100, beta of 0.05, varying attack frequency

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. Of course, with PFP-DA, the regular Simple Topology was used, with 10 nodes attacked to 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 percentage, the pollution will remain at 0%, with almost identical hit rates.

Therefore, we can conclude that if we consider the PFP-DA tests from earlier to be a special case of PFP- β where $\beta = 1.0$, that the smaller $\beta = 0.05$ 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 (by Zipf-like distribution), and about 23% better for top 50 items (98.83% vs 75.53%, for a rate of 720.) Astonishingly, the top 50 items experience just under 99% hit rate with our $\beta = 0.05$. Although 75.53% was incredibly good, and totally outperformed LFU-DA (50% top 50 hit rate), our $\beta = 0.05$ tests far surpass our previous PFP-DA tests in terms of small attacker percentage with varying attacker frequency.

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%

Table 15: Two attackers, frequency of 720, cache size of 100, varying beta value

Secondly, we tested variations of β value, keeping the number of attackers at 2 with attack frequencies of 720 each, a cache size of 100. As we see in Table 15, 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.

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%

Table 16: Two attackers, frequency of 720, β value of 0.1, varying cache size

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. 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 in to the content store's cache.

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, there is little room for improvement.

# 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%

Table 17: Attacker frequency of 720, β value of 0.1, cache size of 100, varying number of attackers

In Table 17 we see where the PFP-β falls short. Although the performance of PFP-β is better than 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 $\beta = 1.0$) allow for more resistance against larger percentages of attackers.

Chapter 6
Conclusions and Future Work

6.1 Summary and Conclusions

The needs of users of the Internet have changed greatly since its inception. Originally, features such as e-mail, sharing research data and other static content, and the sharing of expensive resources among mutually trusting parties were the primary goals. However, modern usage focuses far more on the content itself rather than the origin. Security, privacy, mobility, streaming data, large file transfer, and many other aspects of the current Internet usage are commonplace.

Due to the disparity between the current Internet architecture at the network level and the needs of modern users, many overlays and patches have been applied to the aging architecture. As a result, many researchers have begun to consider the future of the Internet and concluded that its architecture should undergo a complete redesign, resulting in the category of future Internet architectures known as Information-Centric Networks (ICN). Several architectures have been proposed, one of the most important being Named Data Networking (NDN) [10], due largely to its wide early support and growing research community.

Among the many goals of NDN (and most ICN architectures), in-network storage stands out as a major feature. Unlike the current solution deployed as an application overlay (CDN), in-network storage takes advantage of optimizations at the network layer and spreads more storage

across larger regions in so-called content storage routers, which maintain a content store (cache) of popular content objects (more correctly, content object chunks.)

Because of its prominence within ICN architectures, the in-network storage devices and their caches will undoubtedly be major targets for attackers. One attack that is leveraged against the in-network storage capabilities of NDN to reduce its effectiveness is the cache pollution attack, which involves attackers trying to get their (unpopular) contents cached instead of legitimately popular contents. The ultimate goal of these attacks is to pollute the caches and reduce hit rate.

In this thesis, I have discussed several current schemes to mitigate the effects of content pollution, and to improve cache robustness. While these schemes [36], [37], [38] are effective in some circumstances and show promise, they do not attempt demonstrate any specialized mitigation measures against particularly aggressive attackers overwhelming one or a small percentage of faces on a router with Interests for unpopular contents. Additionally, current cache replacement algorithms (LRU, LFU, LFU-DA) are quite susceptible to aggressive attacks, which ultimately pollute the cache, and moreover reduce the hit rate.

Our research has focused largely on reducing the impact of a given attack against a face or small number of faces. Each face has an associated ranked list of popular items on that face, and their rankings contribute to the overall popularity calculation that is calculated at the router. This scheme is called per-face popularity (PFP.) To ensure recency is also considered, and that no content object remains in the cache too long beyond its genuine popularity, dynamic aging was added to the algorithm, resulting in per-face popularity with dynamic aging (PFP-DA.) PFP-DA also improves scalability by reduction in size and bounding of each face's content ranking lists.

Our tests in both very simple and slightly more advanced topologies result in lower cache pollution in many circumstances, and more importantly, higher hit rate, especially for the most popular items.

Furthermore, we found that PFP-DA performs better even when no attacks are being leveraged against the content store. Due to the power law distribution of Interest requests, many items are requested only a single time. Unlike LRU and LFU, which cache every request item, PFP-DA will not likely cache any of these single-request items since they usually receive only a small contribution (due to low ranking) from a single face.

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 $\beta = 1.0$. 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 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 the caching decisions, by 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.

6.2 Future Work

This thesis and the associated research demonstrate dramatic improvement over current cache replacement algorithms. Although substantial promise has been shown, a more powerful testbed is required to effectively and efficiently test larger topologies, and more variations.

Realistically, it may be quite some time until the current IP protocols are replaced. Until then, making CDNs and related technologies as efficient as possible is an important goal. Applying PFP-DA to current technology could improve the state-of-the-art and that which is already deployed.

Furthermore, PFP-DA shows great promise, but it is certainly not as good as it can be. As mentioned earlier, we strive toward zero cache pollution (or as little as possible.) Our PFP-β tests have indicated that for lower percentages of attackers, zero or near-zero cache pollution can be achieved, with very high hit rates. However, more tests with the β value applied in the popularity calculations are needed to test its limits under more circumstances.

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. We look forward to investigating these research opportunities.

References

[1] E. Nygren, R. Sitaraman and J. Sun, "The Akamai Network: A Platform for High-Performance Internet Applications," 2010.

[2] V. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner and Z.-L. Zhang, "Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery," *INFOCOM,* 2012.

[3] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo and S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," in *IEEE 31st International Conference on Distributed Computing Systems*, Minneapolis, MN, 2011.

[4] "Sandvine Global Internet Phenomena Report - 2016," https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf.

[5] Amazon, "Amazon CloudFront CDN," [Online]. Available: http://aws.amazon.com/cloudfront/.

[6] "NSF Future Internet Architecture Project," [Online]. Available: http://www.nets-fia.net/. [Accessed 21 March 2018].

[7] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker and I. Stoica, "A Data-Oriented (and Beyond) Network Architecture," in *SIGCOMM '07*, Kyoto, Japan, 2007.

[8] B. Ahlgren, M. Marchisio, M. D'Ambrosio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Remarz and V. Vercellone, "Design Considerations for a Network of Information," *ACM ReArch '08,* 2008.

[9] "Content Centric Networking Project," [Online]. Available: www.ccnx.org.

[10] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs and R. L. Braynard, "Networking Named Content," in *CoNEXT '09*, Rome, Italy, 2009.

[11] L. Zhang, k. claffy, C. Papadopoulos, L. Wang, P. Crowley and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 3, 2014.

[12] I. Seskar, K. Nagaraja, S. Nelson and D. Raychaudhuri, "MobilityFirst Future Internet Architecture Project," in *Proceedings of the ACM AINTec 2011*, Bangkok, Thailand, 2011.

[13] A. Venkataramani, J. F. Kurose, K. Raychaudhuri, K. Nagaraja, S. Banerjee and Z. M. Mao, "MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 3, pp. 74-80, 2014.

[14] D. Lagutin, K. Visala and S. Tarkoma, "Publish/subscribe for internet: PSIRP perspective," in *Towards the Future Internet - Emerging Trends from European Research*, IOS Press, 2010.

[15] N. Fotiou, P. Nikander, D. Trossen and G. Polyzos, "Developing Information Networking Further: From PSIRP to PURSUIT," *International ICST Conference of Broadband Communications, Networks, and Systems (BROADNETS),* 2010.

[16] A. Seetharam, "On Cachine and Routing in Information-Centric Networks," *IEEE Communications Magazine,* vol. 56, no. 3, pp. 204-209, 2018.

[17] R. Tourani, T. Mick, S. Misra and G. Panwar, "Security, Privacy, and Access Control in Information-Centric Networking: A Survey," *IEEE Communications Surveys and Tutorials,* vol. 20, no. 1, 2018.

[18] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communication Societies*, New York, NY, 1999.

[19] T. Yamakami, "A Zipf-Like Distribution of Popularity and Hits in the Mobile Web Pages with Short Life Time," in *IEEE Proceedings of the Seventh Internation Conference on Parallel and Distributed Computing*, Taipei, Taiwan, 2006.

[20] A. Afanasyev, I. Moiseenko and L. Zhang, "ndnSIM: NDN simulator for NS-3," NDN Technical Report NDN-0005, 2012.

[21] M. D'Ambrosio, C. Dannewitz, H. Karl and V. Vercellone, "MDHT: A hierarchical name-resolution service for information-centric networks," *Information Workshop on Information-Centric Networking (ACM),* 2011.

[22] "Visual Networking Index: Forecast and methodology, 2016-2021 (White Paper)," Cisco, 2017. [Online]. Available: http://www.cisco.com/go/vni. [Accessed April 2018].

[23] C. E. Perkins, S. R. Alpert and B. Wolf, Mobile IP: Design Principles and Practices, Boston, MA: Addison-Wesley, 1997.

[24] P. Eugster, P. Felber, R. Guerraoui and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys,* vol. 35, no. 2, pp. 114-131, 2003.

[25] G. Xylomenos, C. N. Ververids, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros and G. C. Polyzos, "A Survey of Information-Centric Networking Research," *Communication Surveys and Tutorials,* vol. 16, no. 2, pp. 1024-1049, 2014.

[26] E. Ngai, B. Ohlman, G. Tsudik, E. Uzun, M. Wählisch and C. A. Wood, "Can We Make a Cake and Eat it Too? A Discussion of ICN Security and Privacy," in *ACM SIGCOMM Computer Communication Review*, Dagstuhl, Germany, 2016.

[27] C. Dannewitz, J. Golic, B. Ohlman and B. Ahlgren, "Secure Naming for a Network of Information," *INFOCOM IEEE Conference on Computer Communication Workshops,* 2010.

[28] M. D'Ambrosio, C. Dannewitz and H. Karl, "MDHT: A hierarchical name resolution service for information-centric networks," *ACM SIGCOMM Workshop on Information-Centric Networking,* 2011.

[29] Y. Afek, E. Gafni and M. Ricklin, "Upper and Lower Bounds for Routing Schemes in Dynamic Networks," in *Proceedings of the 30th Symposium on Foundations of Computer Science*, 1989.

[30] D. Krioukov, K. Claffy, K. Fall and A. Brady, "On Compact Routing for the Internet," *SIGCOMM Computer Communication Review,* 2007.

[31] M. Aamir and S. Zaidi, "Denial-of-service in content centric (named data) networking: A tutorial and state-of-the-art survey," *Security and Communication Networks,* vol. 8, no. 11, 2014.

[32] P. Gasti, G. Tsudik, E. Uzun and L. Zhang, "DoS and DDoS in Named Data Networking," in *22nd International Conference on Computer Communications and Networks (ICCCN)*, Nassau, Bahamas, 2013.

[33] C. Ghali, G. Tsudik and E. Uzun, "Needle in a Haystack: Mitigating Content Poisoning in Named-Data Networking," in *NDSS Workshop on Security of Emerging Networking Technologies*, San Diego, CA, 2014.

[34] G. Acs, M. Conti, P. Gasti, C. Ghali, G. Tsudik and C. A. Wood, "Privacy-Aware Caching in Information-Centric Networking," *IEEE Transactions on Dependable and Secure Computing (Early Access),* 2017.

[35] P. Denning, "The Locality Principle," *Communications of the ACM,* vol. 48, no. 7, pp. 19-24, 2005.

[36] M. Xie, I. Widjaja and H. Wang, "Enhancing Cache Robustness for Content-Centric Networking," in *2012 Proceedings of the IEEE INFOCOM*, 2012.

[37] M. Conti, P. Gasti and M. Teoli, "A lightweight mechanism for detection of cache pollution attacks in Named Data Networking," *Computer Networks 57 (2013),* 2013.

[38] A. Karami and M. Guerrero-Zapata, "An ANFIS-based cache replacement method for mitigating cache pollution attacks in Named Data Networking," *Elsevier Computer Networks 80,* pp. 51-65, 2015.

[39] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich and T. Jin, "Evaluating Content Management Techniques for Web Proxy Caches," Hewlett Packard, Palo Alto, CA, 1999.

[40] S. Mastorakis, A. Afanasyev, I. Moiseenko and L. Zhang, "ndnSIM 3: An updated NDN simulator for NS-3," NDN Technical Report NDN-0028 (Revision 2), 2016.

[41] "NS-3 Discrete-Event Network Simulator," [Online]. Available: https://www.nsnam.org/. [Accessed 28 October 2017].

[42] "Named Data Networking Forwarding Daemon (NFD) Documentation," [Online]. Available: http://named-data.net/doc/NFD/current/. [Accessed 27 October 2017].

[43] "ndnSIM Documentation," [Online]. Available: http://ndnsim.net/2.4/doxygen/ndn-consumer-zipf-mandelbrot_8cpp_source.html. [Accessed 25 March 2018].

[44] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *IEEE INFOCOM*, 1999.

[45] Y. Gao, L. Deng, A. Kuzmanovic and Y. Chen, "Internet Cache Pollution Attacks and Countermeasures," in *Proceedings of the 2006 14th IEEE International Conference on Network Protocols*, Santa Barbara, CA, 2006.

[46] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs and R. Braynard, "Networking Named Content," *ACM CoNEXT '09,* 2009.

[47] J. Rexford and C. Dovrolis, "Future Internet architecture: clean slate versus evolutionary research," *Communications of the ACM,* vol. 53, no. 9, pp. 36-40, 2010.