UNIVERSITY OF PADUA
DEPARTMENT OF MATHEMATICS
MASTER DEGREE IN COMPUTER SCIENCE

CACHE POLLUTION ATTACKS AND DETECTION IN NAMED DATA NETWORKING

SUPERVISOR
Prof. Mauro Conti
University of Padua

CANDIDATE
Marco Teoli

CO-SUPERVISOR
Prof. Paolo Gasti
New York Institute of Technology

EXTERNAL REVIEWER
Prof. Gene Tsudik
University of California, Irvine

February 1, 2013

ABSTRACT

The current Internet architecture was designed as a mean of connecting pairs of hosts and allow them to reliably exchange packets. The way we use it every day deeply changed from pure communication to (mostly) content distribution. The wish to fill the gap between the underlying architecture and the information-centric nature of the current Internet traffic has inspired many new architectures with the goal of replacing the current infrastructure. Content-Centric Networking (CCN) is one of them.

In CCN, named content becomes a first-class citizen. In contrast with IP, addresses are not explicit. Content consumers simply request what they want to retrieve by name; the network is in charge of delivering the correct content. The decoupling of content and its location allows, among other things, the implementation of ubiquitous caching. Named Data Networking (NDN) is a prominent implementation of the CCN paradigm.

NDN's reliance on caches for performance, reliability and data replication allows an adversary to perform attacks on the architecture that are very effective and relatively easy to implement.

This thesis focuses on cache pollution attacks. In these attacks, the adversary's goal is to alter cache locality, increasing cache misses and link utilization for honest consumers.

We review and evaluate previous work on these attacks and their countermeasures. To perform a meaningful analysis, we simulate Internet traffic with appropriate models on a realistic network: the German Research Network (DFN). Finally, given their importance on cache resilience, we perform pollution attacks on caches that rely on different replacement policies.

Our results show that such attacks can be implemented in NDN using limited resources, and that their effectiveness is not limited to small topologies. We illustrate that countermeasures designed for IP are not easily applicable to NDN. On the other hand, existing mitigation techniques targeted to NDN provide only limited relief against realistic adversaries.

We propose and evaluate a new technique, designed to detect high and low rate cache pollution attacks. Our technique is based on the analysis of variation in traffic distribution as seen by individual routers. We discuss the feasibility of this technique and suggest possible improvements.

Part of this work has been submitted to the *Elsevier Computer Networks Journal* (Special Issue on Information Centric Networking) [19].

Showing gratitude is one of the simplest
yet most powerful things
humans can do for each other

Randy Pausch — The Last Lecture

ACKNOWLEDGMENTS

This work has been possible thanks to the support and good advices of my supervisor, prof. Mauro Conti, and of prof. Paolo Gasti, in quality of co-supervisor. During these months they gave me an insight about research and valuable advices to learn from.

I would also like to thank all those students and friends who worked with me in many different projects, and Master's candidates with whom I shared this adventure.

This achievement would not be possible without the support of my parents, who deserve gratitude for giving me the chance to continue my studies so far.

Finally, I would like to thank my sister, Deborah, and all the friends who shared with me important moments of my life.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

During 1960-s and 1970-s, the Internet was conceived and developed as a communication network, inspired by telephony. Researchers designed a way to connect different hosts in a network in which all the endpoints were able to communicate.

In recent years there have been several efforts to design a viable replacement for the current IP-based Internet [1, 2, 6, 7, 4]. These new architectures are designed to better serve today's needs and allow the current growth rate of the Internet to continue for the foreseeable future [26]. In particular, there have been significant efforts to provide better mobility, scalability and efficient content distribution. Additionally, strong security has been one of the main design requirements for these architectures.

In the following, we provide some motivation to support this need for change. We then shortly introduce NDN (a more extensive presentation is provided in the next chapter) and the work done with this thesis in contribution to NDN security.

1.1 BEYOND HOST-CENTRIC NETWORKING

Today's use of the Internet is very different from the original concept of communication network. People use the Internet to connect and keep in touch, to look for information and share content. The Internet today looks more like a huge cloud of information than pure communication network. There is a visible gap between the information-centric essence of the Internet and its host-centric architecture.

In the current host-centric Internet architecture, when a user – let's say, Alice – requests some content, the underlying host is in charge of fetching that content from the network. This generally means (1) obtaining the IP address of the server hosting that content and (2) forwarding the request to that server. Content is then available to Alice thanks to some end-to-end communication between Alice's client and the hosting server.

Alice was not indeed asking to contact some specific server: she was just requesting a content. The need to establish a communication with the source of information is the architectural way to fulfill Alice's request. It is not the only possible choice and content distribution provides some mean to improve this network behavior.

1.2 CONTENT DISTRIBUTION AND NDN

Content distribution is a service that allows replicas of the same content to be stored inside many different servers geographically distributed inside a network. Content distribution provides fast delivery, because clients can dynamically retrieve the content from the closest server. It consequently reduce network bandwidth and reduce the workload that would be experienced by a single server. Content distribution is extensively used in Content Delivery Networks (CDN).

The emerging Content Centric Networking (CCN), also known as Information-Centric Networking (ICN), is a paradigm that makes an impressive use of content delivery, for the sake of efficiency and to boost performance. In CCN, content – rather than hosts – occupies the central role in the communication architecture. Named Data Networking (NDN) [1] is a prominent example of Content-Centric Networking.

NDN is primarily oriented towards efficient large-scale content distribution. Rather than establishing direct IP connections with a host serving content, NDN consumers directly request (i.e., express *interest* in) pieces of content by name; the network is in charge of finding the closest copy of the content, and of retrieving it as efficiently as possible. One of the key features enabled by this decoupling of content and location is pervasive caching. Each NDN router can, in fact, provide an arbitrary amount of cache that can store forwarded content for subsequent retrieval. This allows NDN to transparently and automatically implement efficient multicast, content replication, load balancing and fault tolerance.

1.3 SIDE EFFECTS ON SECURITY

NDN provides several advantages, not limited to those mentioned in the previous section. An important goal of NDN is to provide mechanisms to improve network security. However, pervasive caching exacerbates security problems related to shared caches, including privacy [33], pollution [21] and poisoning [39]. In this thesis we focus on cache pollution attacks with respect to NDN. In a cache pollution attack, the goal of the adversary is to force routers (i.e., the victims of the attack) to cache non-popular content. Therefore, a successful attack reduces cache hits of content requests from legitimate consumers (*secondary* victims), affecting overall network performance and increasing link utilization.

Cache pollution attacks do not prevent users from retrieving content. Nonetheless, some work considers them a distributed denial-of-service (DDoS) [42, 21]. Deng et al. [21] show that even a moderate degradation of hit ratio, on large and popular content, can increase the amount of traffic not served by caches by a few orders of magni-

tude [21]. They point out that *"Long periods of severe service reduction* [...] *can on average degrade service more than classical high-rate DoS attacks are capable of"*. We are not interested in arguing whether cache pollution falls under the umbrella of DoS/DDoS attacks; besides, our experiments confirm that the impact of pollution attack on routers and on end users is significant and deserves careful study.

1.4 CONTRIBUTION

The contribution of this thesis includes:

- Previous work on cache pollution in NDN/CCN only focuses on small topologies (one to nine routers [42, 21]). In this work we show that cache pollution attacks scale to large topologies, without requiring substantially more resources on the adversary's side. Our findings provide further evidence that cache pollution attacks are a realistic threat in large-scale NDN deployments.
- We confirm that proactive countermeasures identified in previous work scale to complex networks, assuming the same (simple) adversarial behavior.
- We show how to improve the cache pollution attack considered in [42]. Under our attack, existing techniques for cache robustness do not provide real benefit. In particular, in some circumstances, caches implementing such countermeasures perform worse than the same caches without them.
- Given this state of affair, we argue that an effective approach for mitigating cache pollution attacks could be based on a lightweight detection phase, associated with a (possibly more expensive) reaction protocol. We propose a new cache pollution detection algorithm, and evaluate it via simulations. Our simulations are performed on realistic network topologies such as the Deutsches Forschungsnetz ("German Research Network", DFN) [30].

We emphasize that the aim of this work is not to solve the problem of cache pollution. Rather, we highlight that existing proactive techniques, which are believed to be effective against this attack, fail in presence of a slightly "smarter" – although still very simple – adversary. We then show that an approach different from the current state of the art can provide better performance at a lower cost in realistic scenarios.

2 NAMED DATA NETWORKING

Named Data Networking (NDN) aims at changing the paradigm on which the current Internet was built: NDN focuses on content rather than communication endpoints (hosts). To this end, while the current Internet is a communication network, NDN is a content distribution network.

There is currently a substantial research effort behind NDN, justified by the wish to overcome the limitations of the current Internet architecture, through the creation of something that fits the needs for which it is used today, but in a more appropriate manner.

An important improvement introduced by NDN with respect to the Internet is that NDN does not aim at securing the container (i.e., the host, as the current Internet does), but at securing the content. Once content is secured and the network is able to address content rather than hosts, data can be fetched from anywhere in a secure way, enabling pervasive caching of content.

The next sections introduce some conceptual, architectural and operational detail of the current NDN design, presenting both the state-of-the-art and the open challenges.

2.1 ARCHITECTURE AND OPERATION OF NDN

NDN is based on the pull model, where content is injected into the network only in response to a consumer's explicit request. NDN clients are called *consumers*, while servers are identified as *producers*. Consumers request content that is originally provided by producers. As it will become clear through this chapter, in NDN requests are routed to their respective contents, and only *consequently* to the nodes hosting such contents.

2.1.1 *Interests and contents*

NDN supports two types of messages: *interests* and *content objects* [3]. An interest corresponds to the request issued by a consumer, while a content object (also known as *Data* or *Content*) is the response given by the producer. A consumer might express an *interest* in a specific content, identified by name. It can otherwise provide a *name prefix* along with some other qualifications.

NDN names follow the same convention of filesystems: objects belong to the branch of the tree represented by the prefix that appear in their name; a name prefix can actually match with more than one

content object. When the consumer provides a name prefix, all the content objects that match the given prefix could actually be selected, but only one can be returned (*at most once* policy). Qualifications are used to restrict the set of content objects that could be selected. If some qualification is provided, only objects that satisfy the restriction applied by the aforementioned qualification are selectable.

Interest packets (see Figure 1) thus contain the Content Name and a set of other fields, simply shown as "Selector" in the figure, which are used to further qualify the content, where it might come from and – in case of multiple matching – the content object to select [5]. In particular, a special field of the interest message, called *ChildSelector*, is used to select which content object should be returned when there is multiple choice. Finally, interest packets contain a random *nonce*, used during routing to discard duplicate interests, thus avoiding loops.

Content objects have:

- a name (possibly human-readable);
- a data payload that represent the content of the message;
- a cryptographic signature and an identification of the signer (i.e., the *publisher*).

Furthermore, content objects contain some other signed information, such as:

- a *Timestamp*, i.e., an indication of when the object was sent;
- *FreshnessSeconds*, a time after which the object is considered stale;
- *KeyLocator*, the location of the signing key.

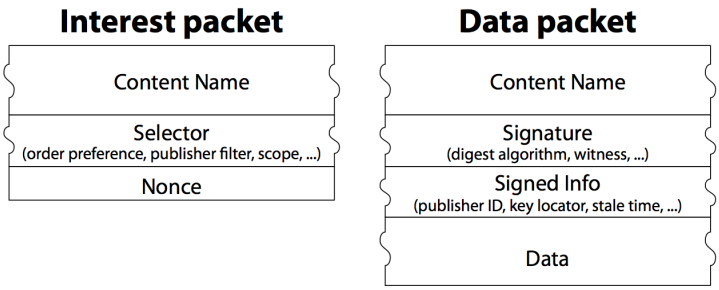


Figure 1: NDN packets [43].

2.1.2 *Names*

Names are made up of one or more *component* elements with hierarchical structure. Components are separated by a "/", as in

```
/ndn/it/unipd/math/cs
```

The hierarchical structure is essential for the scaling of today's routing system [43] and is also useful for applications to represent relationships between pieces of content. For example, fragment 5 of version 3 of Alice's video could be named

```
/flickr/alice/wonderland-holidays.mpg/3/5.
```

Names are opaque to routers and naming conventions (such as version and fragments) are opaque to the network, but applications can agree on some common naming convention in order to use content objects correctly. For example, if Alice requests her video without indicating a specific segment and the producer split the content into smaller fragments, her client must be able to correctly identify all those fragment by name, in a deterministic manner.

Finally, names are not required to be globally unique in a local environment, but their uniqueness becomes mandatory when the content has to be fetched globally. Some strategy to avoid name clashes is necessary; the hierarchical structure of names allows the adoption of some policy for top level names, but how to allocate them is still an open challenge [43].

2.1.3 *Name-based Routing*

NDN routers forward interests towards the content producer(s) responsible for the requested name, using name prefixes (instead of today's IP prefixes) for routing. For this reason, NDN packets do not contain any information about sender and receiver addresses, nor NDN has such a concept.

NDN routers are based on three important building blocks: a *Forwarding Information Base* (FIB), a *Pending Interest Table* (PIT) and a *Content Store* (CS).

The Forwarding Information Base is a lookup table populated by a name-based routing protocol and used to determine interfaces for forwarding incoming interests. Every time a router is requested to fetch some content from the network, it forwards the interest according to the information stored inside the FIB. Multiple concurrent entries for the same prefix are allowed, supporting multipath delivery.

2.1.4 *Backward traversal with "Le Petit Poucet"*

The Pending Interest Table is a second lookup table containing outstanding interests. When a router receive a request for some content, it stores the corresponding interest inside its own PIT along with the name of the arrival interface. The interest is then forwarded through the network. If a new request for the same content is received, its arrival interface is added to the *arrival-interfaces* list for that interest. For efficiency's sake, multiple pending interests for the same content are

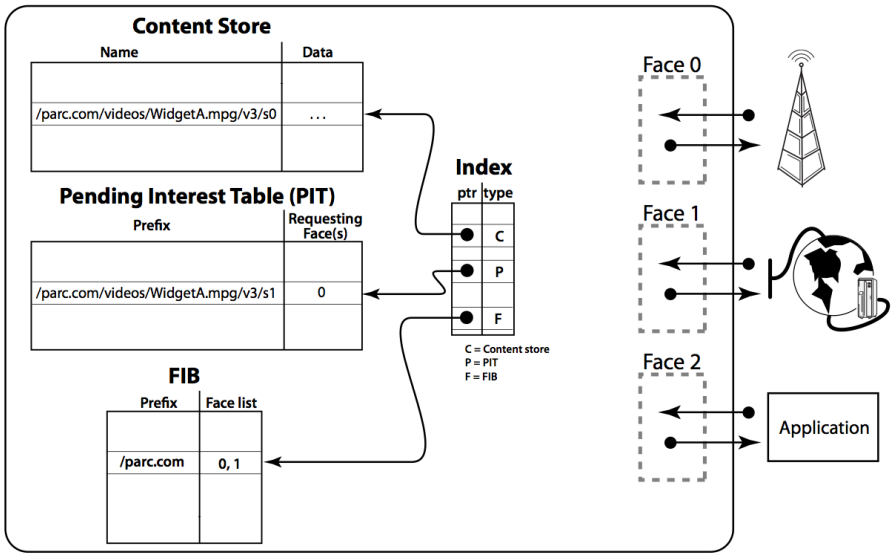


Figure 2: NDN router [43].

collapsed: only the first interest is forwarded, while the others are discarded. Upon receipt of the interest, the producer injects content into the network, thus *satisfying* the interest. The requested content is then forwarded towards the consumer, traversing – in reverse – the path of the corresponding interest. Each router on the path flushes the state (i.e., deletes the PIT entry) corresponding to the satisfied interest.

The backward in-order traversal of content objects is possible thanks to the information stored inside the PIT. This might remember *Le Petit Poucet* (*Hop-o'-My-Thumb*) and the small white pebbles he used to return home, which inspired the title of this section.

2.1.5 *Ubiquitous caching*

The behavior described above applies mainly to content requested for the first time or – more generally – that has not been requested for some time. In fact, a key feature of NDN is distributed caching. In particular, each router that forwards a content may cache a copy of the content in the router's local Content Store.[1]

NDN does not mandate any specific cache size or cache management algorithm. Each router is free to select what to cache according to local information. Distributed caching enables NDN to implement efficient large-scale content distribution. Producers do not need to issue a copy of their content for each consumer request, because consumers always retrieve the closest available copy of requested content. This in turn alleviate producers and reduces network traffic.

1 In the rest of this thesis, the terms CS and cache will be used interchangeably.

2.2 NDN SECURITY AND POSSIBLE THREATS

NDN aims at providing "security by design". Its security model differs from that of the current Internet because NDN secures – as we already introduced – the content rather than the host. In practice, this can be realized observing that content objects are signed. The binding between name, publisher and payload in a content object is immutable, because content objects must contain a valid signature.

While intermediate routers are not required to verify any content object (although they *can*), applications are required to verify the content and discard messages that fail to verify. The use of signatures provides data integrity and authentication (even for cached content), so that the consumer can trust the content origin.

Since all content in NDN is signed, the efficiency in signature generation and usage is critical and need further research efforts. Researchers are investigating for example fast signature schemes and methods for aggregate signature generation and verification. Technical report [43] describes some of the research work already planned a couple of years ago, while the NDN project Website [1] collects much of the improvements achieved so far.

Despite its design, NDN is not a panacea for network security. There are challenges to address in order to have NDN become effective and some possible threat. It is worth mention for example the need to design solutions for *revocation*, which is very important since content can be spread everywhere through the pervasive caching of NDN. In Section 2.2.1 we introduce a few possible threats, including the main focus of the research work addressed by this thesis: cache pollution attacks.

2.2.1 *IFA and pollution attacks*

Other important examples are Interest Flooding Attacks (IFA) and Content Pollution Attacks (CPA). IFA is similar to traditional Denial of Service (DoS) attacks, where the adversary sends a huge number of interests that cannot be satisfied (e.g., requests for non-existing content), thus filling the PIT with fake interests. Moreover, adversary's requests in IFA are supposed to be all distinct, so that the router is unable to aggregate them. For IFAs, NDN project authors propose to limit the number of unsatisfied interests that routers hold for a given domain [43]. A first solution, based on thresholds, has already been proposed in [18, 17].

In a Content Pollution Attack, the adversary introduces malicious content trying to make it match legitimate requests. This attack has probably a smaller impact because of the adoption of signatures for all content objects. Nonetheless, it deserves and already obtained the attention of the scientific community. A possible example of content

pollution is a malware generating spam at the source, that would be spread as legitimate content with a legal signature.

This thesis evaluates a class of attacks known in literature, which is now attracting increasing interest from the community with the advent of NDN in the research world: Cache Pollution Attacks. Those attacks differ from content pollution attacks because – instead of pushing malicious content inside of content stores – in this case the adversary tries to fill content stores with non-popular content to reduce the hit ratio of caches and – consequently – the performance of the network.

3

INTERNET TRAFFIC DISTRIBUTION AND WEB CACHING

There have been several studies on the distribution of Internet traffic [15, 28, 20], i.e., the *probability distribution* of requests for content on the Internet. Internet traffic distribution has an enormous impact on the benefits that ubiquitous caching might give to the network.

If the Internet traffic was following a uniform distribution, there would be probably no advantage in caching content. Indeed, according to many studies presented below, user requests are not uniformly distributed; some content is requested very often, while a large amount of data is only seldom – or almost never – requested.

Since our detection mechanism relies on the distribution of users' requests, in this chapter we overview previous work on this topic. We also present an overview of replacement strategies used for Web caching: such policies strongly influence the resilience of Web caches against cache pollution attacks. Finally, we explain the reasons that drove us to choose two of these policies in our simulations.

3.1 INTERNET TRAFFIC DISTRIBUTION

According to Breslau et al. [15], Web page requests follow a Zipf-like [44] distribution. In this section we first introduce Zipf-like distributions; afterwards, we describe the findings of Breslau et al., as well as other research work that argue on its applicability to the Internet traffic.

3.1.1 *The Zipf-like distribution of Web pages*

The Zipf's law states that in the English language, the probability of encountering the k^{th} most common word is approximately $P(k) = 0.1/k$, for k up to 1000 or so [41].

If we order words from the most frequent to the one that appears less frequently, the most common word appears approximately two times as often as the second, three times as often as the third, and so on. So, if the frequency of the first one is, let's say, 1, the 2^{nd} has frequency 1/2, the 3^{rd} 1/3, etc.

The Zipf distribution provides a mathematical model of this law. The probability mass function of the Zipf distribution is:

$$f(k, \alpha, N) = \frac{k^{-\alpha}}{\sum_{n=1}^{N} n^{-\alpha}} \tag{1}$$

where:

- $k \in [1, N]$ indexes the k^{th} most frequent item;
- $\alpha \in \mathbb{R}^+$ represent the decrease rate of the function; it is equal to 1 for the strict definition of the Zipf's law given above; values smaller than one are generally used to better represent the Internet traffic [15];
- $N \in \mathbb{N}\backslash\{0\}$ is the number of possible items.

Figure 4 shows the Zipf distribution; being a power-law function, due to its characteristics it is often represented in log-log scale. In that case, it looks linear.

Breslau et al. [15] collected three properties often found in literature that apply to Web proxies:

- the hit ratio grows in a log-like fashion as a function of their client population and the number of requests they see;
- the hit ratio grows in a log-like fashion as a function of the cache size;
- the probability that a document will be re-referenced after k requests is proportional to $1/k$.

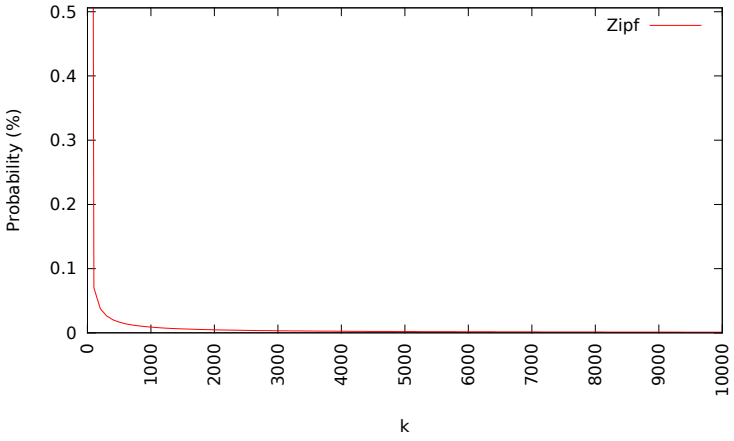
They propose a simple model in which they assume that Web page requests:

1. are independent (*independent reference model* [22]);
2. follow a Zipf-like distribution.

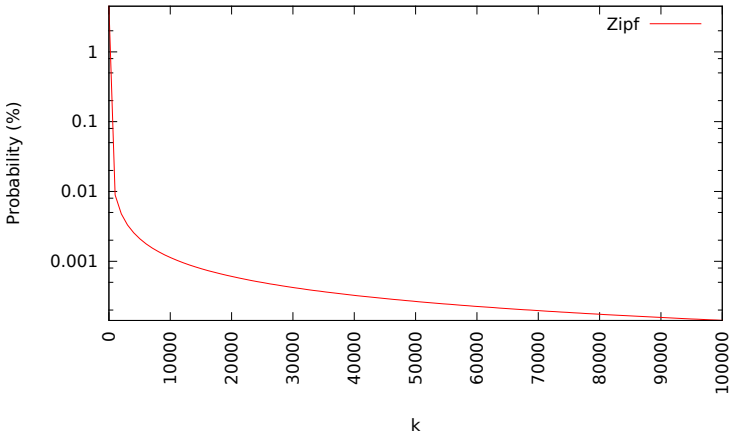
Although they recognize that there is some very weak correlation between access frequency and page size/rate of change, they show some evidence that the properties above still hold in their model. Therefore, they observe that their model "may be sufficient to understand certain asymptotic properties of cache performance". They found that all their traces were approximated by Zipf-like distributions with α values that vary between 0.66 and 0.78.

The same distribution is assumed in many subsequent papers (e.g., [21, 36, 42]).

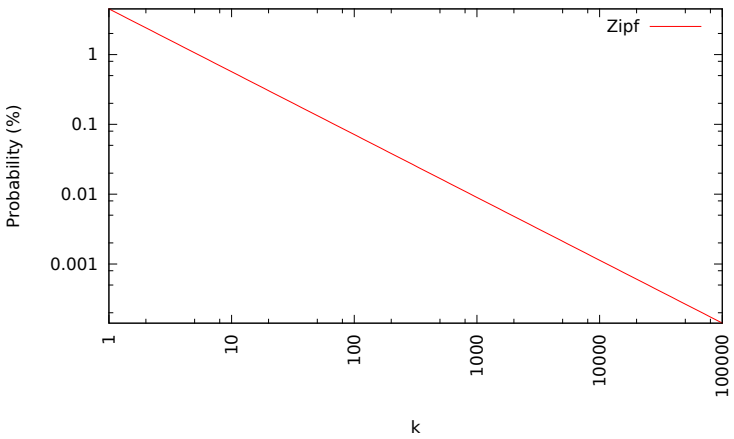
Nonetheless, according to other studies [28, 20], under some circumstances this does not apply to Internet traffic in general. We discuss some of their findings in the next section.



(a) Normal



(b) Log scale



(c) Log-log scale

Figure 3: Zipf-like distribution used in our simulations: $\alpha = 0.9$, $N = 100000$. N is too large to visualize the function in full scale in Figure 3.3(a).

3.1.2 *Other models and insights*

Zipf's law appear very frequently in natural, mathematical, social sciences, economics [34]; it seems reasonable to accept that Internet traffic is also subjected to the same law. Nevertheless, other studies based on other traces and/or different types of traffic show evidence that – under different conditions – Internet requests follow different models. For example, Guo et al. [28] doubt that the Zipf's law holds, claim that Internet traffic follows a Stretched Exponential (SE) distribution [29] and provide some results to support their thesis. According to their results, SE holds for (a) Web media systems, (b) Voice/Video on Demand media systems, (c) peer-to-peer (P2P) media systems and (d) live media systems.

As a last example, Dán et al. [20] provide an extensive analysis of peer-to-peer traffic over a long time period. They state that the Zipf's law might only hold for the instantaneous popularity; they conclude that over long time periods, P2P content popularity does not have a power-law tail, but it rather exhibits an exponential cutoff [27] that can be captured only with long term measurements.

We find indeed reasonable to observe that:

1. different types of traffic could exhibit a different traffic distribution than that shown by Breslau et al. in their paper for Web pages;

2. the evolution of the Internet might have some consequence on the Internet traffic in general; an increasing number of dynamic pages lead to less cacheable content [37], as well as the increasing bandwidth available to users, whom are now able to make a more extensive use of video streaming, VoIP calls, etc.

About the last observation, Raminovich et al. [38] measured that non-cacheable objects were already about 40% at the time of their writing (in 2001); nonetheless, this phenomenon does not really influence our work, because non cacheable content can simply be ignored (we will not simulate non-cacheable traffic).

The first observation is more critical for our purposes and has to be addressed. In evaluating our approach, we still assume that honest consumers request content according to a Zipf-like distribution. However, we show evidence that varying such distribution does not alter the performances of our technique. In particular, our approach performs detection on single nodes that are not aware of the overall traffic distribution. Therefore, we do not assume that the traffic they receive fully represents the overall traffic distribution.

3.2 WEB CACHING

As observed by Deng et al. [21], the impact of malicious users with respect to cache pollution strongly depends on the replacement algorithm in use. Their findings are also confirmed by our results.

All replacement policies provide two important operations that is worth mentioning before introducing them in detail. The first one is the *addition* of a new object. Usually – when the cache is full – addition of an object or content is associated with the *eviction* of another one (or more than one, depending on packet size). The second operation is the *lookup* for an object inside the cache. Lookups generally change the order of eviction of cached objects.

Different categorization of Web caching replacement algorithms has been provided in several surveys [37, 13, 11]. In this section we present an insight for a small selection of these algorithms.

3.2.1 *Least Recently Used (LRU)*

Recency-based policies are based on the idea that if a content has been requested recently, there is a good probability that it will be requested again in a relatively short time (time locality). Therefore, it is more convenient to replace content that has not been requested for longer.

A "pure" recency based algorithm is LRU (*Least Recently Used*), which relies only on recency. LRU conceptually maintains a list of cached objects, ordered by recency of access; new objects are positioned at the head of the queue, while the objects at the queue, being older, are replaced when the cache is full. In case of lookup, the object – if found – becomes the most recent and is thus moved at the top of the queue.

Time complexity of insertion and eviction in LRU is $O(1)$; lookups can be done in $O(\log(n))$ using the LRU list along with a second data structure (e.g., a tree).

3.2.2 *Frequency based strategies*

Frequency-based strategies (such as LFU, *Least Frequently Used*) take into account popularity of requests. A cache using LFU keeps track of the frequency of objects and use a data structure, such as a priority queue, to keep objects ordered by frequency. Time complexity of LFU is $O(\log(n))$.

The basic idea behind LFU is very simple; its implementation is instead very error-prone and contains hidden pitfalls.

There are two classes of LFU:

- *Perfect LFU*, in which counters persist after replacement;
- *In-cache LFU*, in which counters are initialized when the corresponding content is stored and then removed contextually to the eviction of the content.

With Perfect LFU, it would be possible to reject the insertion of objects that have lower frequencies than cached ones, because their frequencies are remembered. This is not what usually happens, because LFU is a replacement policy: when a cache receives a request for insertion, it uses the replacement policy to decide which objects should be evicted, without choosing between the incoming object and the outgoing one. The least frequent cached object is removed regardless of the fact that its frequency could be higher than that of the incoming object. Therefore, storing frequencies for every possible object (even those that are not cached), perfect LFU could effectively be used to keep in cache only (and exactly) the most frequent objects.[1]

When using In-Cache LFU, rejecting an object that has a frequency lower than every cached object would be a serious mistake. Objects frequencies are initialized to one at the time of the insertion and removed when objects are evicted; therefore, using this rejection mechanism, if all the cached objects reached a frequency greater than one, nothing else would ever be cacheable anymore.

Perfect LFU is not used in practice, because the Internet is a dynamic domain. The difference between static and dynamic domains in not negligible:

- in a *static domain*, the set of objects that can be requested by consumers is always the same;
- in a *dynamic domain*, such as the Internet, the set of objects that can be requested by consumers changes over time. Most of them, after some time, are not requested anymore. Others are instead created and become popular. Changes in a dynamic domain regard both (1) the pure "existence" of objects and (2) their popularity, which generally changes over time.

In a dynamic domain, Perfect LFU requires too much space to store all the counters, therefore it is not really used. In-cache LFU is instead feasible (if correctly implemented) because it does not have such space overhead. In the following, although not specified, all mentioned LFU policies are intended to be In-cache LFU variants.

1 Indeed, the most frequent are such with respect to the statistics collected at the time of the decision, but might become less frequent afterwards. LFU is clearly unable to predict future trend of requests.

3.2.3 *LFU with (dynamic) aging: LFU-DA*

A major issue with LFU is *aging*. An item might be very popular for a short period and then become forgotten. Afterwards, if not enough elements are requested a higher number of times, that item might remain cached for a very long time. This issue is generally addressed with the introduction of aging techniques.

An example is *LFU-Aging*. If the average of frequency counters exceed a predefined threshold, they are all divided by 2. This allows counters to decrease when contents loose popularity, thus reducing the effects of aging. The major drawback with this approach is the need to (periodically) change all the counters, because it is quite expensive for big caches can actually store an impressive number of objects. Furthermore, LFU-Aging introduces a parameter (the threshold) that should be properly tuned.

There are many other approaches to aging. We now describe one of them, i.e., an algorithm that we decided to use in our simulations: *LFU with dynamic aging* (LFU-DA) [23].

LFU-DA maintains items sorted according to a key value:

$$K_i = f_i + L$$

where f_i is the frequency of i and L is an aging factor, initially set to zero. LFU-DA replaces the object with the smallest K_i value, always updating L to the K_i of the evicted object.

3.2.4 *Other policies*

Recency and frequency can be used together in order to provide policies that take advantage of both the characteristics. There are different policies in this category (e.g., SLRU [37], LRFU [31], CSS [37]). To some extent, aging techniques introduced in LFU policies are ways to take recency into account.

Other than recency and frequency, size of the object is another important factor of many replacement policies. One example is GD-Size (*Greedy Dual Size*), which works similarly to LFU-DA:

$$H_i = \frac{c_i}{s_i} + L$$

where:

- L is the same aging factor described above;
- c_i is the cost to fetch object i from the network;
- s_i is the size of object i;
- the resulting H_i is used to determine which object should be evicted (i.e., the one with the smallest H_i).

GD-Size is clearly not a "pure" *size-based* algorithm. Strategies that take into account many different factors are often called *function-based* policies [37].

Figure 4 shows a clear categorization of many different replacement algorithms.

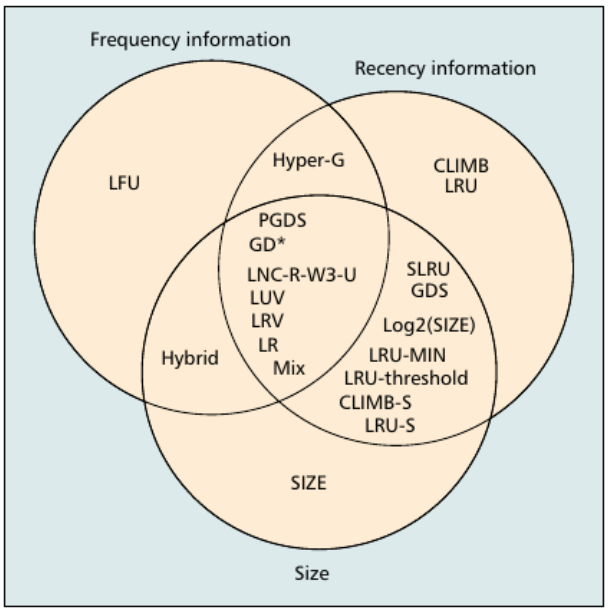


Figure 4: A classification of replacement policies [13].

There are many other classes of replacement policies. *Randomized strategies* for example use randomized decisions to evict an object. A basic example is RAND, which removes a random object.
In the first NDN technical report [43], authors hypothesize that randomized policies might provide over NDN performance very close to the other – more sophisticated – policies.

To conclude this overview, we mention *adaptive replacement* (adoption of different strategies depending on the network workload) and – as examples of very different strategies – policies that include neuro-fuzzy systems, neural networks, nonlinear and logistic regression models [11].

3.2.5 *Conclusion*

We adopt two replacement techniques covered by the aforementioned literature: recency-based and frequency-based policies. The aim to support both these policies has already been espressed in the original project specification [43]. For the study performed in this work, we selected LRU and LFU-DA. Both the algorithms have been selected, after extensive analyses [12, 23, 24], by Squid [9], one of the most popular caching proxies. Although caching proxies and routers' CS do not have identical goals and restrictions, we argue that cache re-

placement policies that perform well in the former are a good starting point for the latter.

Squid also introduced size-based policies, such us GDSF (GDS-Frequency), a variant of Greedy Dual-Size. For our purposes, we assume that all objects have the same size, therefore there would be no advantage in introducing size within the replacement policies supported by our environment.

We also exclude the other aforementioned policies, preferring LRU and LFU-DA. They are popular in proxy caches [9] and their implementation is desirable in NDN [43]; furthermore, they are different enough to provide evidence of the impact of replacement policies on cache resilience against pollution attacks.

4

CACHE POLLUTION ATTACKS AND EXISTING COUNTERMEASURES

Cache pollution attacks aim at spoiling cache locality in order to decrease the hit ratio and, consequently, affect the overall performance of the network.

Deng et al. [21] show that even a moderate degradation of cache hit-ratio can increase network traffic by one or more orders of magnitude when it affects large/popular content.

They propose two generic classes of cache pollution attacks:

- *locality-disruption*, where the goal of the adversary is to maliciously alter cache content locality. This attack can be implemented issuing requests for new non-popular content;
- *false-locality*, where the adversary aims at maliciously increasing the popularity of a (usually small) fraction of the available content. The adversary implements this attack issuing a large number of requests for the same pieces of content.

As a side note, it is interesting to note that the normal activity of robots and Web crawlers deployed by search engines can also cause locality disruption [21].

4.1 COUNTERMEASURES FOR THE CURRENT INTERNET

Cache pollution attacks have been widely studied in IP-based networks, mostly with respect to caching of Web traffic. Previous work on this subject is very relevant in the context of NDN – although NDN exhibits some relevant architectural difference – because there is a number of analogies that provide useful suggestions to our study.

4.1.1 *A countermeasure based on Bloom filters*

In [21], Den et al. propose detection mechanisms for both types of attack. They also combined their detection mechanisms to detect both attacks when executed at the same time.

DETECTION OF FALSE LOCALITY In case of false locality, their technique tracks:

- the number of repeated requests;
- the ratio of repeated requests over the number of cache hits.

When both metrics exceed given thresholds, corresponding sources are identified as malicious. The number of repeated requests is a value that changes very significantly during the attack, because the adversary need to repeat may times the same requests, so that the corresponding objects become popular.

In the second metric, they use the number of hits rather than the overall number of requests. They found that the attacker would be otherwise able to avoid detection by launching both false-locality and locality-disruption simultaneously.

As they observe, this approach have a high memory consumption without extra arrangements. As a first optimization, before counting repeated requests they apply a threshold on the request rate, so that they avoid measurements and waste of memory for useless information. An interesting aspect of their solution is that they use Bloom filters [16]. This choice allows to record the state needed to calculate request repetitions and optimizes memory consumption.

A Bloom filter is a probabilistic data structure used to test whether an element is inside a set. When asked about the presence of an element, the Bloom filter might return:

- *False*, where the negative answer means that the element is *definitely* not in set;
- *True*, where a positive answer *generally* means that the element is in set, but with some small probability the answer might be wrong.

A small uncertainty about the positive answer – which would lead to false positives – is generally accepted in exchange for an important reduction of memory consumption: an attractive quality of Bloom filters is their high space efficiency.

DETECTION OF LOCALITY DISRUPTION The mechanism used to detect locality disruption is also based on thresholds, where the metrics used for detection are:

- hit ratio;
- average lifetime of cached content.

While the hit ratio is obviously supposed to degrade during attack (for the attack to be effective), the latter metric looks more interesting.

During this attack, the adversary generally request different new objects. When pushed inside the cache, these objects generally replace other non-popular objects, starting from the tail of the list used by the replacement policy to evict objects. Popular objects are instead generally close to the head of the list, both for LRU (being repositioned after each lookup), and for LFU (due to their higher frequencies). All

these frequent replacements decrease the average lifetime of cached content.

In case of LFU, we could guess that during the attack, adversary's requests often replaces objects that she herself introduced in cache. In fact, adversary's requests in a locality disruption attack are unlikely to be for popular content; therefore, the respective objects are likely to occupy the tail of the eviction list used by LFU – exactly where LFU evicts non-popular content and inserts new objects. This also explains why locality disruption attacks are less effective against LFU, as it will become very clear from the measurements presented in the next chapter.

When hit ratio and lifetime of content drop below predefined values, the cache is considered under attack.

MIXED ATTACKS AND COUNTERMEASURE When both locality disruption and false locality attacks are launched simultaneously, the second metric used for detecting false locality is not affected due to the use of the overall number of hits instead of the overall number of requests. The ability to detect locality disruption would be instead affected, because false locality attacks increase the lifetime of content, compensating the decrease caused by locality disruption. To address this issue, this approach first performs false locality detection and identifies the objects affected by false locality; then it excludes such objects from the computation of the average lifetime.

Attack mitigation relies on identifying the source of malicious requests. For this purpose, the authors use content requesters' IP addresses.

REMARKS While the approaches of Deng et al. work well on IP-based networks, it is not clear how to extend it to NDN (and, in general, CCN). Interests in NDN do not carry identifying information about the consumer(s) who issued them. As such, routers cannot keep statistics on repeated interests from the same host. Furthermore, the availability of caches at each router creates a "dampening" effect on consumers' interests: the traffic observed by a router varies depending on what its neighbors have retained in their cache.

4.1.2 *Detection through randomness checks*

Park et al. [36] consider pollution attacks in the context of content caching. The authors propose a detection mechanism primarily focused on locality disruption attacks. Their approach is based on randomness of content requests, measured on caching nodes (e.g., NDN routers).

In their approach, summarized in Figure 5, they periodically compute a binary matrix $M = (m_{i,j})$; object names are associated to an entry of the matrix using a mapping function.

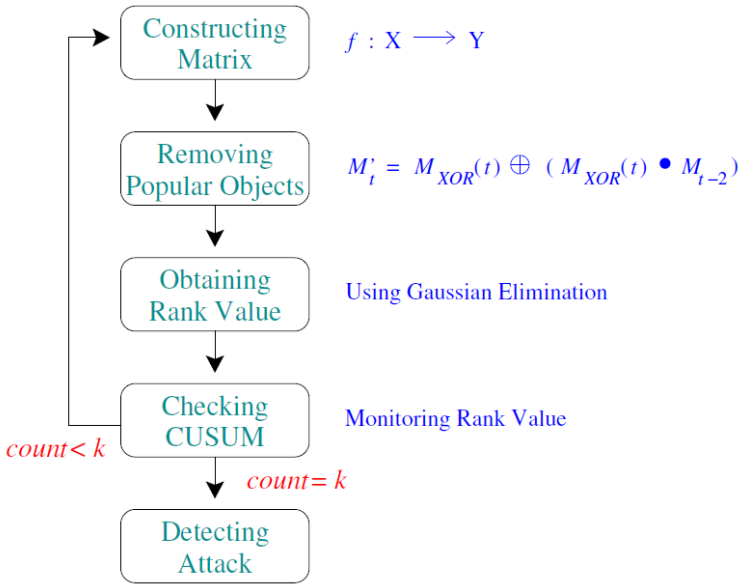


Figure 5: Park's approach to attack detection [36].

The mapping function computes the coordinates (i,j) as follows:

$$\begin{cases} i = Hash_1(c) \mod n \\ j = Hash_2(c) \mod m \end{cases}$$

where $Hash_1$ and $Hash_2$ are fast cryptographic functions and $n \times m$ is the size of the matrix. In practice, in their simulations they used SHA-1 both for $Hash_1$ and $Hash_2$, while the matrix size is

$$m = n \simeq [\sqrt{S_C}]$$

where S_C is defined as the average number of objects in cache.

Matrix entries are initially set to zero. Each time an object is requested, the corresponding entry is set to one.

Popular objects that are repeatedly requested by legitimate users are likely to map the same indexes in contiguous time units; on the other hand, the adversaries are performing a locality disruption attack and their requests – being always different – always map to different indexes.

Therefore, requests made by legitimate users exhibit a different behavior if compared with those issued by the adversary: they have a different level of randomness. When the adversary starts issuing requests, it changes the level of randomness of the matrix; if we can measure such changes, we can detect the attack.

Before explaining how to measure the randomness, it is important to observe that its value is correlated with the time passed since the

initialization of the matrix. At the beginning, all values are zeros. After a long time, most values would be equal to one. Therefore, the "right" time should be found "in the middle" (not at the beginning, but before it is too late), such that the entropy inside the matrix is maximum. Because that time is dependent on the rate of consumers and adversaries, it is more appropriate to use a different measure, i.e., the number of requests received by the node. The authors measured that number (called "time unit") and suggest that nodes should calculate the randomness each time they receive about $3n$ requests.

Then, nodes compute the matrix to measure the randomness and then restart with a new, empty matrix.

The measure of randomness used for attack detection is not actually performed on the aforementioned matrix. Before measuring randomness, each node should first perform some operations that actually reset the entries corresponding to popular objects, i.e., objects often requested by legitimate users. This operations require that nodes maintain the values of the last three matrices, M_{t-2}, M_{t-1}, M_t, which are then re-elaborated as follows:

$$M_{XOR(t)} = M_{t-1} \oplus M_t$$
$$M'_t = M_{XOR(t)} \oplus (M_{XOR(t)} \bullet M_{t-2})$$

where $\oplus$ and $\bullet$ represent the *XOR* and *AND* operations, respectively. Computations needed for attack detection are then executed on M'_t.

The detection algorithm then calculates the rank r_t of M'_t using Gaussian elimination.

The next step of attack detection is based on the *Cumulative Sum* (CUSUM) algorithm, which is used in anomaly detection [14].

CUSUM is used along with the average of the rank; this can be calculated using an *Exponential Weighted Moving Average* (EWMA):

$$\mu_t = \beta\mu_{t-1} + (1-\beta)r_t$$

where μ_t is the average of the rank (r_t), and β is the EMWA factor.

CUSUM is then defined as follows:

$$\begin{cases} g_t = [r_t + g_{t-1} - \mu_t]^+ \\ g_0 = 0 \end{cases}$$

where $[x]^+ = x$ if $x > 0$, $[x]^+ = 0$ otherwise.

In its last step, this detection mechanism notifies the presence of attack if g_t exceeds a given threshold h for k consecutive time units.

REMARKS This technique does not rely on source addresses of requests and can be applied to both IP- and NDN-based networks. In fact, while Park et al. consider Internet caching at large, they mention CCN as a possible scenario.

The analysis presented in [36] is based on a single caching node, connected to a consumer, a producer and an adversary. The results of this analysis cannot be directly extended to realistic NDN topologies, generally composed of many consecutive caching routers, because of the effects of the aforementioned "dampening" effect on traffic randomness. Moreover, since the detection mechanism ignores repeated requests, it cannot be used to detect false locality attacks.

Finally, we observe that the storage cost of the approach of Park et al. is $O(\sqrt{N})$, where N is the cache size. Indeed, each matrix occupies a storage space proportional to S_C; since cache replacement policies tend to fill the cache as much as possible (they evict objects when needed to store new content), it is safe to approximate C_S to the cache size. From the computational point of view, the two most expensive operations are:

1. the computation of a collision-resistant hash function for each forwarded packet;
2. the computation of the rank of the matrix.

4.2 A COUNTERMEASURE FOR CCN: CACHESHIELD

To the best of our knowledge, CacheShield [42] is the only countermeasure for cache pollution attacks specifically designed for (and evaluated on) NDN. Given the relevance of CacheShield to our work, we also evaluate (in the next chapter) the quality of its detection in our scenarios.

CacheShield is a technique with the goal of improving NDN cache robustness against pollution attacks – in particular locality disruption. CacheShield tries to prevent non-popular content from being cached.

The detection mechanism is summarized in Figure 6.

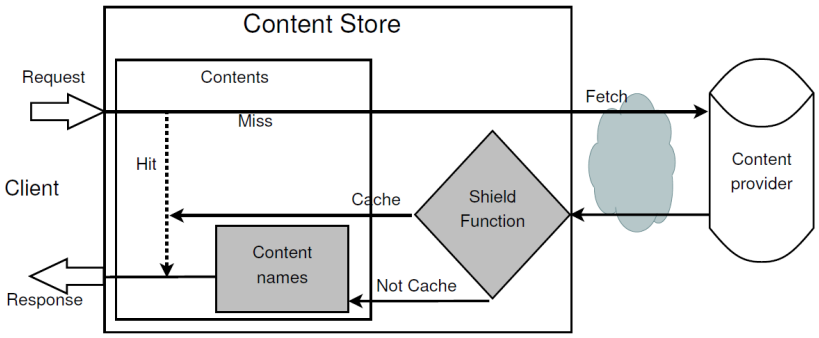


Figure 6: Xie's approach to cache pollution attacks [42].

When a CacheShield-enabled router receives a content object, the CS evaluates a shielding function that determines whether the content object should be cached:

- if the shielding function returns *true*, the router forwards the interest and caches the corresponding content object;
- if the shielding function returns *false*, only the name of the returned content object (or the hash of its name) and a counter are stored in the cache as a placeholder.

If the same (still not cached) content object is requested again, the corresponding counter is increased and the shielding function is re-evaluated on the updated counter.

Any replacement policy (e.g., LRU, LFU) can be used in conjunction with CacheShield. Once the cache is full, content placeholders (i.e., NDN names and counters) are subject to the same replacement rules as cached content.

In order to establish whether a content object should be cached, the shielding function executes a random choice based on a logistic function [40]; the authors propose the following instantiation:

$$\psi(t) = \frac{1}{1 + e^{(p-t)/q}}$$

where parameters p and q can be tuned in order to achieve the best results. Using synthetic data and multiple small topologies (with up to nine routers), Xie et al. determined that the shielding function generally works well with $p = 20$ and $q = 1$. For a fair comparison with CacheShield, we also evaluate our approach using the largest of the topologies in [42], in addition to a larger (and more realistic) one.

Further details and discussion on the implementation of CacheShield in our simulations are provided in Section 5.1.

REMARKS Xie et al. proposed an interesting approach to cache pollution in CCN and measured its performance in different small topologies. According to their results, their approach is effective and provide good performance against locality disruption attacks.

There are a few aspects that remain unanswered. Their work does not show if their approach scales to larger and more realistic topologies. Moreover, although they performed experiments with different replacement policies, those experiments were limited to a single node. It would be interesting to study the effect of the attack on larger topologies with different replacement policies. Since resilience against cache pollution attacks also depends on the replacement policy used, the possibility of the adversary to impact many subsequent nodes might be limited.

Furthermore, their approach is based on an unrealistic assumption: the space needed to store names in place of the respective objects is negligible. Names – or their hashes – require clearly less space than the corresponding objects, since they have no payload, but the amount of names that are eventually added to the cache by CacheShield is potentially not negligible.

Finally, this detection algorithm is mostly oriented to locality disruption attacks, thus there is no direct evidence about the effectiveness of this approach against false locality attacks.

In the following chapters we evaluate cache pollution attacks and this countermeasure providing many experimental details. We then propose a different approach, limited to attack detection, but able to reveal attacks that cannot be addressed by CacheShield.

5

EFFECTIVENESS OF ATTACKS AND COUNTERMEASURE

This chapter starts with a preliminary description of the simulator used to run our experiments, including a set of features that we implemented to obtain the results reported in this thesis.

We then illustrate the topologies used in our simulations and report the settings used in our analysis; in particular, we configured CacheShield preserving the settings in [42]. We also followed the assumption that storage needed by CacheShield statistics (names of non-cached objects) is negligible, and thus rounded to zero.[1] This assumption is kept in favor of CacheShield. On the other hand, our threat model is adapted to stress CacheShield on a more realistic scenario; in our attacker model, we restrict the selection of content requested by the adversary in a way that it actually creates false locality.

Finally, we evaluate the effectiveness of the attack with and without CacheShield and show that – under these assumptions – CacheShield becomes ineffective.

5.1 EVALUATION ENVIRONMENT

We evaluate attacks and countermeasures discussed in this thesis via simulations. We rely on ns-3 [8], a well-known open-source discrete-event network simulator; NDN features are introduced in ns-3 by ndnSIM, a module for ns-3 developed at the University of California, Los Angeles, as part of the NDN project.

5.1.1 *ns-3*

ns-3 supports wired and wireless networks based on IP or other protocols. The simulator is mainly composed by a core and a set of simulation models, which allow the user to set up the scenario of interest. ns-3 is a C++ library and does not use or require any domain-specific modeling language.

Users can interact with core and simulation models of ns-3 using C++, but also through Python scripts, thanks to Python bindings that wrap the ns-3 core and models. It makes an extensive use of callbacks (events are scheduled on the simulator and then executed through

1 The cache storage available to statistics is consequently unlimited, because statistics do not occupy any space.

callbacks) and provides an attribute system to manage simulation parameters.

ns-3 is also oriented to emulation, because of its internal representation of packet structure, sockets and network devices that makes its architecture aligned with real operating systems. This design allows the simulator to interact with the real world, sending out packets in a real network interface.

5.1.2 *ndnSIM*

The structure of ndnSIM is cleaner and more extensible than that of other NDN simulation environments [10]. ndnSIM is implemented as a new network-layer protocol, which can run on top of any available link-layer protocol (point-to-point, CSMA, wireless, etc.).

ndnSIM provides a set of routing techniques very specific of NDN. It implements all the building components of NDN routers (FIB, PIT, content stores) and some basic consumer and producer.

In a pure ns-3 fashion, ndnSIM also includes a set of helpers that simplifies the set-up of NDN networks, making it easier to install the network stack and populate routing tables.

5.1.3 *Changes and new features*

Our simulations have been executed mainly on two clusters. The first one belongs to the department of Mathematics at the University of Padua; the second belongs to Caspur, a consortium of universities, currently part of Cineca (one of the most important computing centers worldwide).

The two clusters feature different Linux distributions and configurations. Some features, such as Python bindings and especially Boost libraries, were not perfectly working with ndnSIM and/or ns-3, due to versioning and configuration issues. Libraries available on those clusters were in some case outdated and the mechanisms to load custom versions were heterogeneous. This sometimes made it more convenient to replace limited portions of source code when such libraries were not strictly needed.

In order to perform our simulations, we deactivated Python bindings (because not needed) and modified parts of ndnSIM. In particular, content stores were almost rewritten, along with the implementation of the LRU replacement policy.

We also added some extra-features:

- we implemented the cache replacement policies Perfect LFU, In-cache LFU[2] and LFU with dynamic aging, because they were not available in ndnSIM;

2 Perfect LFU and in-cache LFU have been used at the beginning to study the behavior of NDN networks and to understand the different performances of these replace-

- in order to evaluate the current state-of-the-art solution and to set-up the baseline for our simulations, we implemented CacheShield and integrated its implementation to the cache;
- we added a consumer that requests contents according to the Zipf distribution;
- we implemented an adversary that follows the attacker model described in Section 5.4;
- we modified some other parts of ndnSIM to collect the statistics presented in this thesis; we implemented the data structures needed to elaborate such statistics, as well as the code needed to produce our plots.

5.2 TOPOLOGIES USED FOR SIMULATIONS

For our simulations, we consider two topologies (see Figure 7):

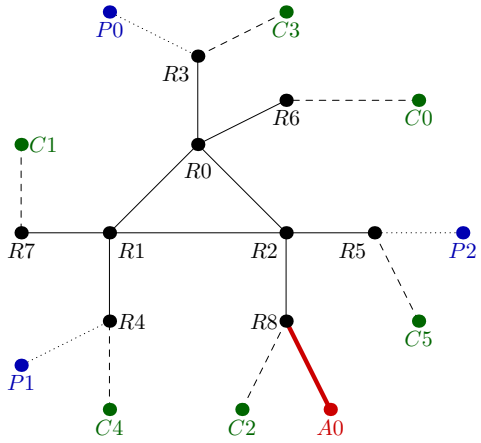
- Xie-complex (XC) corresponds to the "Complex Network" considered in [42], and allows us to compare our results with those of Xie et al.;
- the German Research Network (DFN), which has been identified in previous work as a meaningful topology for simulations [30]. This allows us to provide a more realistic assessment of our technique.

We also performed a number of simulations on two smaller topologies:

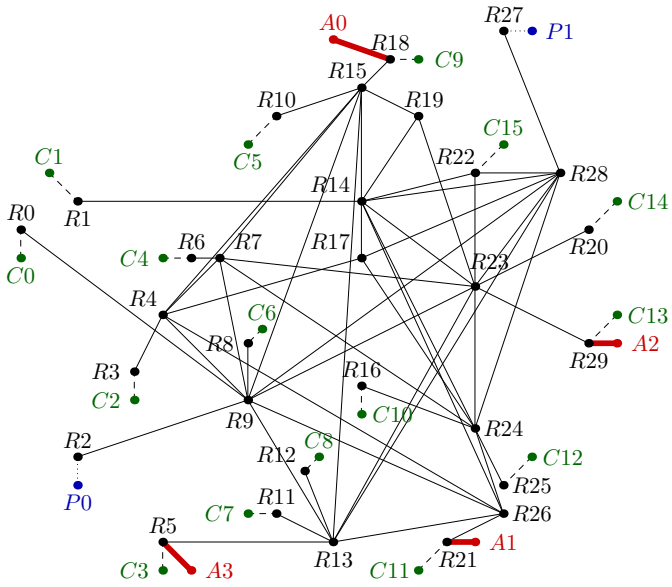
- a *simple* topology, consisting in a router, a producer, a consumer and an adversary; this topology was used only to assess the correctness of the simulation environment, especially for the new features that were originally missing in the simulator;
- the topology used for the merging scenario in [42]; we used this topology mainly to replicate the settings used by Xie et al. and to verify the conformity of our implementation of CacheShield; to this end, we have been able to replicate their results.

Since XC and DFN are fully representative of our findings, the other two topologies are not used to present our results in this thesis.

ment Algorithms. Nevertheless, they have not been used any further. We found LRU and LFU-DA more valuable for our purposes.



(a) XC topology



(b) DFN

Figure 7: Network topologies used for simulations.

5.3 SIMULATION SETTINGS

All our simulations span over twenty-four hours, and follow a similar pattern. During the first twelve hours, all interests are issued only by legitimate consumers; requests follow the Zipf law. This allows caching algorithms, and especially CacheShield, to "stabilize" their internal state according to the forwarded traffic. Then, adversary-controlled consumers start issuing interests.

As in Xie et al. [42], the domain of possible content objects is static, and the CS on all routers is limited to the same size. We fix the total amount of content available on the network, and limit the size of

routers' CS to 1% of the content domain. Consumers' requests follow a Zipf-like distribution (Zipf with parameter α set to 0.9), while the adversary requests content according to the uniform distribution. For the sake of simplicity, we also follow the assumption that all content objects have the same size.

Let ϑ indicate the total number of content objects used by the adversary during the attack, as a fraction of routers' cache size. We vary ϑ from 50% of routers' cache ($\vartheta = 0.5$) to the whole content domain ($\vartheta = 100$). The latter scenario is equivalent to the uniform attack of [42]. We also include "no attack" as a reference, and indicate it with $\vartheta = 0$.

5.4 THREAT MODEL AND ATTACK STRATEGY

We assume that the adversary can compromise a (small) set of consumers, and use them to issue interests. The adversary does not have any special privilege – e.g., it cannot alter setup parameters of caching algorithms. These assumption are similar to those in previous work (e.g., [42, 21]).[3] The adversary is not allowed to generate new content. However, it can request any existing content object from legitimate producers.

We impose restrictions on resources available to the adversary. In particular, the aggregate bandwidth available to the consumers controlled by the adversary is between 2% and 250% of the bandwidth of honest consumers. This allows us to explore the effects of moderate- and low-bandwidth attacks, and whether countermeasures are able to identify or mitigate them. We indicate the fraction of interests issued by the adversary with respect to the total traffic with γ. As an example, $\gamma = 0.02$ indicates that the adversary issues 2% of the total number of interests in the network.

We argue that, under the assumption above, the adversary strategy for generating interests in Xie et al. [42] is sub-optimal. In particular, in their work the authors assume that the adversary issues interests for all existing content with equal probability. In the rest of the thesis, we refer to this strategy as *broad-selection*. While broad-selection works relatively well against caching algorithms based on LRU, according to our results (see Section 5.5) this attack is not very effective against LFU-based cache replacement algorithms.

In order to implement false locality taking into account the aforementioned bandwidth limitations, the adversary could focus on a carefully-chosen small subset of the available content. We refer to this strategy as *smart-selection*. Assuming Zipf-like traffic distribution, smart-selection focuses on the tail of such distribution – i.e., on the least-requested content.

3 While we do not exclude that attacks might come from compromised routers, we leave the investigation of this as future work.

In [42], the authors state that CacheShield's statistics for names should be stored in the CS. This implies that the more CacheShield's statistics grow, the less space is available for caching content. Furthermore, the authors do not discuss how to partition the CS between content caching and caching statistics for best performance. For this reason, we implement CacheShield using a separate storage area: new statistics added to this area do not reduce the space available to cached content. Moreover, we do not impose any limit to the space used by statistics. Hence, our simulations provide an upper bound on the performances that can be expected from CacheShield.

The Zipf distribution is characterized by a long tail – i.e., there is a large number of items which are requested rarely, with roughly the same number of requests. The number of non-popular items is significantly larger than the number of popular ones, and in practice larger than deployed caches. Hence, the impact of the attack does not vary significantly if the adversary does not pick exactly the least-requested content objects. In practice, the adversary will implement this attack making use of publicly available information about distribution of content requests. It is in fact safe to assume that the adversary can construct a list of globally not-so-popular content (e.g., local news from ten years ago), or locally not-so-popular pieces of content (e.g., news in French language in a non-French-speaking country).

Finally, we consider LRU and LFU with dynamic aging (LFU-DA), as cache replacements techniques.

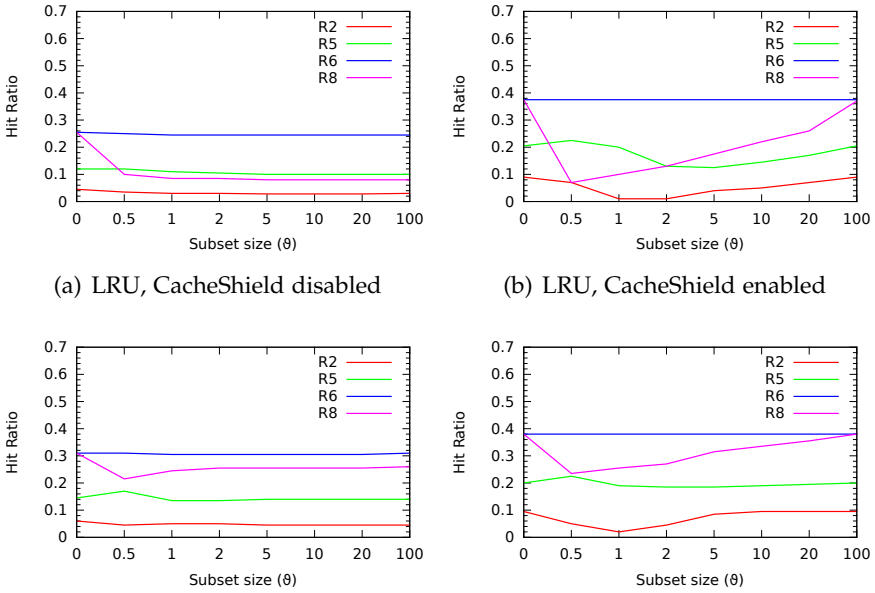
5.5 ATTACK EFFECTIVENESS

In this section, we assess how attacks based on broad-selection and smart-selection affect our topologies. In particular, we are interested in determining whether the adversary can have an impact on cache hit ratio, and consequently increase average hop count of consumers' requests. We emphasize that, as in Xie et al. [42], we limit our adversary to only issue interests. In other words, the adversary cannot generate content on one side of the network, and request it on the other. This way, it must rely on legitimate producers to implement the attack.

5.5.1 *Effects of the Attack on Hit Ratio*

Figure 8 shows the average hit ratio (per router) of routers' caches in the XC network. The figure reports hit ratio at the end of the simulations. For clarity, in Figure 8 (as well as in other figures in these last chapters), we report only results for selected routers.

We observe that the impact of the attack on routers implementing LRU is greater than that on routers using LFU-DA, especially for large ϑ and without CacheShield. With LFU-DA the adversary



(a) LRU, CacheShield disabled

(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

Figure 8: Hit-ratio on XC topology using different subset sizes, for $\gamma = 1.7$.

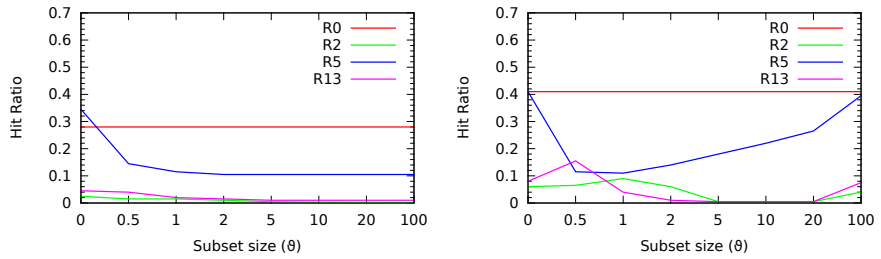
must in fact request the same non-popular content objects often to make them "artificially" popular. Spreading the attack over a large set of content objects alters the request frequency of each content only slightly, causing limited impact on LFU-DA. However, when using LRU, requesting new content pushes older (legitimate) content objects out of routers' caches, making the attack effective.

Our results confirm that CacheShield performs well with $\vartheta = 100$ – as expected from the findings of Xie et al. [42]. However, varying the subset size we were able to significantly limit the effectiveness of CacheShield. In particular, with $0.5 \leqslant \vartheta \leqslant 2$ CacheShield provides no advantage over straightforward LRU or LFU-DA. In our experiments we noticed that, under specific conditions, activating CacheShield decreases the ratio of cache hits. For example, for $\vartheta = 0.5$ router R8 with LRU and CacheShield, has a hit ratio of 0.07 (see Figure 5.8(b)), compared to 0.1 without CacheShield (see Figure 5.8(a)). Another example of loss of performance when activating CacheShield is router R2 for $\vartheta = 1$, using both LRU and LFU-DA.

The effects of the attack on the adversary's first hop determines how the attack propagates to the rest of the network. Content cached on the first hop prevents part of the the attack to spread further. Intuitively, if the number of content objects requested by the adversary is smaller than the size of the cache, maliciously crafted interests will not propagate besides the first victim, since they are satisfied by the first router. This accounts, e.g., for the increased hit ratio observed by R5 in Figure 5.8(c) for $\vartheta = 0.5$. Analogously, while the impact of the

attack on R8 for $\vartheta > 1$ is somewhat reduced, the effect on subsequent routers is more pronounced.

The same observation holds for a larger and more complex topology – namely, the DFN – as shown in Figure 9. For the DFN and LFU-DA, the value ϑ for which router R2 experiences the best performance is $\vartheta = 1$: both with CacheShield (Figure 5.9(d)) and without CacheShield (Figure 5.9(c)). This is also true for LRU with CacheShield (Figure 5.9(b)), but not for LRU without CacheShield (Figure 5.9(a)). We also observe that for R5 and $\vartheta = 0.5$, the hit ratio of LRU is 0.15, which is 25% higher than that of LRU with CacheShield (0.12). This observation highlights that a single-router topology, as used in previous work [36], is not sufficient to determine the effectiveness of pollution attacks and detection/countermeasures.



(a) LRU, CacheShield disabled

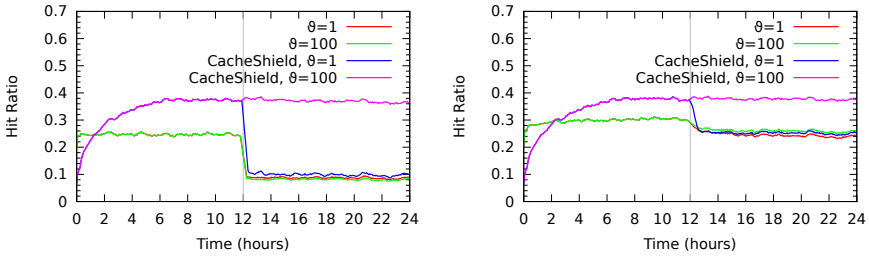
(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

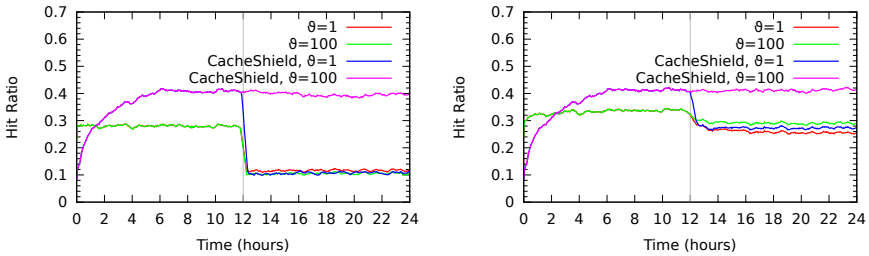
Figure 9: Hit-ratio on DFN topology using different subset sizes, for $\gamma = 2.5$.

Figures 10 and 11 compare the effects of attacks over time with $\vartheta = 1$ and $\vartheta = 100$, for one router in each topology. Using the settings for CacheShield identified in [42] and reported in Section 4.2, content objects are requested about 20 times before being cached. For this reason, CacheShield negatively affects cache hits over the course of the first 2-3 hours in our simulations. In general, this may indicate that a CacheShield-enabled CS requires a considerable amount of time to adjust to changes in content requests distribution. Additionally, the cost of CacheShield in terms of storage and computation is significant. Despite the authors' claim that the space required to store placeholders containing names (or their hashes) is negligible, our experiments show that the number of names stored during network activity is one order of magnitude higher than the number of objects in cache.



(a) Router 8, LRU (b) Router 8, LFU-DA

Figure 10: Effects of attacks on hit-ratio on XC topology for selected routers, with $\gamma = 1.7$.



(a) Router 5, LRU (b) Router 5, LFU-DA

Figure 11: Effects of attacks on hit-ratio on DFN topology for selected routers, with $\gamma = 2.5$.

Routers could allocate a small partition of CS for placeholders; however, this may negatively affect CacheShield's performance.

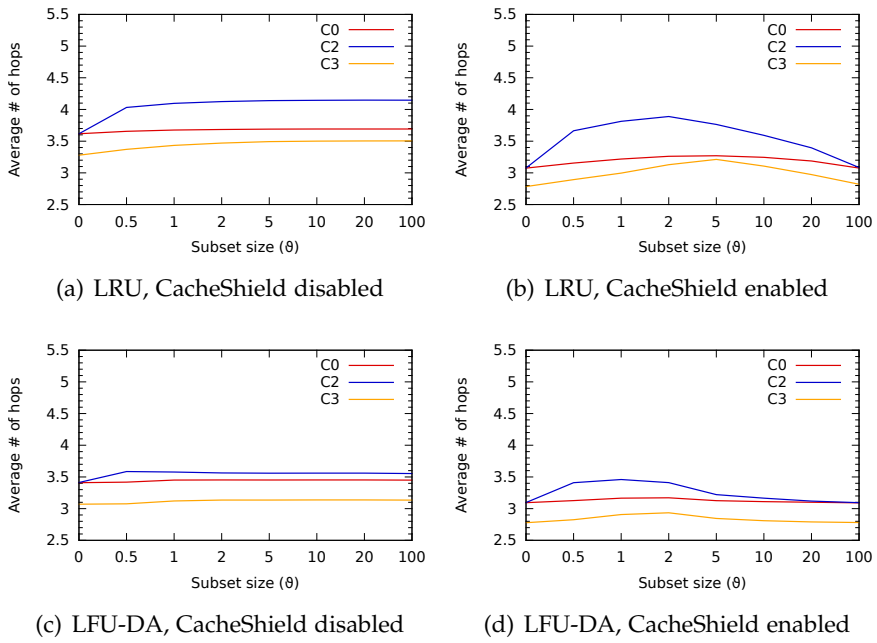
Small values for ϑ may not allow the attack to propagate besides one or two hops. For this reason, measuring the effects of the attack only in terms of hit ratio may not be sufficient. Therefore, we measure average hop count of content objects.

5.5.2 *Effects of the Attack on Average Hop Count*

We measure average hop count to better assess the effects of cache pollution attacks on different topologies. It is well known that, at least in the current IP-based Internet, round-trip time is directly connected to hop count [25].[4] Therefore, we consider this a meaningful metric for determining the impact of the attack on consumers.

Figure 12 shows the average hop count for different values of ϑ. As expected the worst-case scenario for LRU is with $\vartheta = 100$. In this case, CacheShield is able to successfully mitigate this attack. However, for smaller ϑ the benefits of CacheShield are almost non-existent. A similar behavior can be observed for LFU-DA and LFU-DA with

4 Not surprisingly, experiments on the official NDN testbed [1] confirm that this holds also for NDN.



(a) LRU, CacheShield disabled

(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

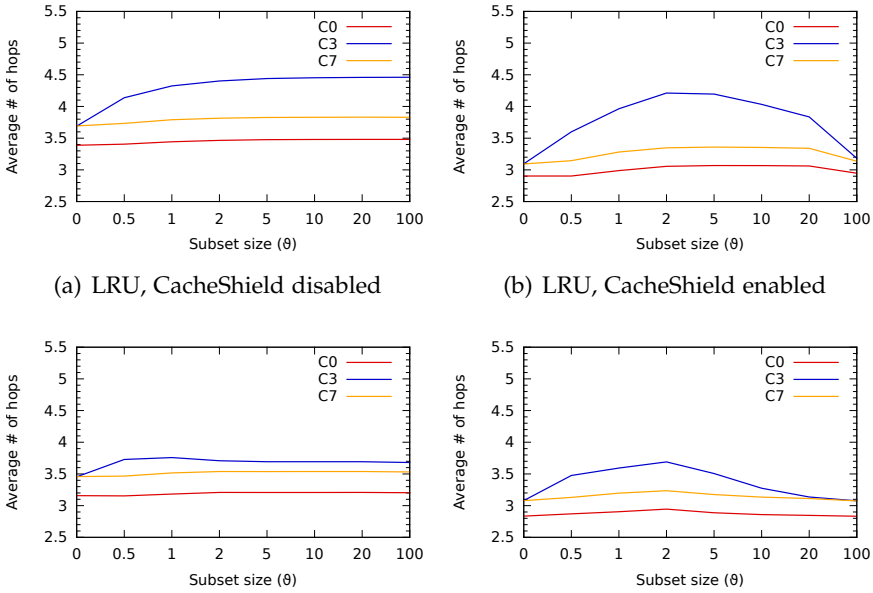
(d) LFU-DA, CacheShield enabled

Figure 12: Average number of hops on XC topology using different subset sizes, for $\gamma = 1.7$. Results are illustrated per consumer.

CacheShield. In general, attacks with ϑ set to a value slightly higher than 1 provide the strongest impact, since malicious interests cannot be immediately satisfied by the router one hop away from the adversary.

In the XC topology the effects of the attack are more evident for consumer C2 (see Figure 12). In fact C2 shares its first-hop router with the adversary. The effects on C0 and C3 are more limited, since they are a few hops away from the adversary. These results also show that LRU with CacheShield's behaves similarly to LFU-DA, since both algorithms maintain analogous information about requests frequencies. With both LFU-DA and CacheShield, the highest impact is obtained with ϑ close to one.

The same experiments were also performed on the DFN topology, leading to similar results. These supports our claim that cache pollution affects both small and large topologies, and the effectiveness (or lack thereof) of CacheShield does not depend on the network size or structure. Figure 13 summarizes our findings with respect to average hop count on DFN.



(a) LRU, CacheShield disabled

(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

Figure 13: Average number of hops on DFN network for different subset sizes, for $\gamma = 2.5$. Results are illustrated per consumer.

6 ATTACK DETECTION

Proactive techniques, such as CacheShield, may not be the most appropriate approach against cache pollution attacks. CacheShield must store a large amount of state, corresponding to name placeholders and related statistics, reducing the space available to cache content. Additionally, this state may be used by the adversary to implement attacks specific to CacheShield: if a router does not specify a maximum quota for placeholders, an adversary may be able to use them to saturate the router's Content Store (CS); if it *does* specify a quota, the adversary's goal may become filling this state with useless information.

CacheShield must run continuously, and therefore consume routers' constrained storage and computing resources, even when no attack is in progress. Our simulations show that CacheShield is ineffective against some (realistic) pollution attacks.

Therefore, we argue that relying on a lightweight attack detection technique (which could potentially trigger a more resource-intensive reaction phase) may be a better approach. The detection phase should make limited assumption on the adversary's behavior. Contrary to more traditional DDoS attacks, where the service is actually denied, routers victims of a cache pollution attack are not usually able to determine whether the attack is in progress. In fact, content is still delivered to consumers when routers are under attack – albeit with reduced performance.

We introduce a detection mechanism that is able to identify cache pollution attacks, while requiring only limited resources. We show that realistic (successful) cache pollution attacks vary the distribution of content requests in a way that is detectable by routers.

6.1 SAMPLING INTERESTS DISTRIBUTION

Routers can learn how the forwarded traffic is distributed by counting how often each content object is returned in response to an interest, normalizing the results by the amount of forwarded traffic.[1] Let $p(i)$ be the probability value associated with content object i. We have:

$$p(i) = \frac{n_r(i)}{\sum_{j \in S} n_r(j)} \tag{2}$$

1 Counting how many different interests are forwarded does not provide reliable figures, since in NDN interests carrying different names may retrieve the same content object due to longest prefix match.

where $n_r(i)$ is the number of occurrences of content object i and S is the reference sample set, i.e., a randomly selected subset of the content domain. Unfortunately, even considering a relatively small S, analyzing all forwarded content packets is not feasible on resource-constrained routers.

Randomly selecting part of the forwarded traffic reduces the cost of computing these statistics. There is however a tradeoff between the granularity of the sampling and the amount of noise added by the process. In particular, a coarser sampling increases the probability of assigning a wrong "weight" to items towards the tail of Zipf distributions. On the other hand, the cost of a very fine-grained sampling is difficult to justify with ever-changing content, since expensive to compute statistics quickly become useless as the content domain varies. In such circumstances, the process of computing content statistics must be as fast as possible, in order to maintain some (limited) amount of (still fresh) historical data and avoid reporting false positives.

Our approach measures the variation across samples to determine what is normal behavior. Let m be a single measurement; N_r^m indicates the size of such measurement. We define the variability of a measurement as:

$$\delta_m = \sum_{i \in S} \left(\frac{n_r^m(i)}{N_r^m} - p(i) \right)$$

and the maximum acceptable variability (threshold) as the average of all measured δ_{m_i}, plus λ times their standard deviation:

$$\tau = \frac{\sum_{i=0}^{M} \delta_{m_i}}{M} + \lambda \cdot \sqrt{\frac{1}{M} \sum_{i=0}^{M} \left(\delta_{m_i} - \frac{\sum_{i=0}^{M} \delta_{m_i}}{M} \right)^2} \tag{3}$$

where M is the total number of measurements.

Routers compute τ in what we call *learning phase*. The actual implementation we describe in the following exhibits a few differences from Equation 3:

1. τ is computed on-line, so that its cost can be amortized across the whole learning phase. Additionally, routers temporarily suspend updating this value once it becomes "stable enough". This makes the algorithm more flexible, and less dependent on a specific network topology or position of a router within this topology.

2. We use Knuth's recurrence formulas [32] to compute the mean and variance needed to determine τ.

Knuth's recurrence formulas are as follows:

$$\begin{aligned} M_1 &= x_1, \quad & M_k &= M_{k-1} + (x_k - M_{k-1})/k \\ S_1 &= 0, \quad & S_k &= S_{k-1} + (x_k - M_{k-1}) \cdot (x_k - M_k) \end{aligned} \tag{4}$$

where M_k represents the average of the first k values and S_k is used to compute the standard deviation, as $\sigma_k = \sqrt{S_k/(k-1)}$ (for the first k elements).

We then calculate the threshold τ_k, i.e. the value of τ at the k^{th} step, as:

$$\begin{aligned} \tau_1 &= M_1 \\ \tau_k &= M_k + \lambda \cdot \sigma_k \end{aligned} \tag{5}$$

During our simulations, we observed that τ takes longer to become stable when computed on routers several hops away from consumers. This can be explained by the effect of routers' caches close to consumers in "dampening" consumer's requests visible to"core" routers.[2] Once caches are populated and start getting a significant hit rate, the distribution of interests they forward stabilizes. This is another reason for allowing each router to autonomously evaluate when τ have become stable, rather that specifying a common learning timeframe.

6.2 DETECTION ALGORITHM

Our *attack detection* technique is based on two separate phases. The first is a learning phase, in which nodes observe the traffic and define the correct value for the threshold. When the learning phase is finished, it starts a detection phase in which the current traffic is compared against the threshold.

This mechanism is detailed in Algorithm 1. We envision attack detection as part of the caching algorithm of NDN routers. Our technique relies on two sub-routines, which correspond to a *learning step* (Algorithm 2) and an *attack test* (Algorithm 3). Those sub-routines reflect the analyses performed in the aforementioned phases.

The caching algorithm first builds a set S of content object IDs by performing random sampling of a suitable number of packets. The detection algorithm will keep track of this set and use it for determining whether an attack is in progress. Once S has been populated, the router invokes Algorithm 1 on each forwarded content object CO.

The algorithm first checks whether CO is in S and eventually updates the corresponding counter (co_count[CO]). Afterwards, if it has enough data for the analysis (as described in the next subsection), it invokes the learning algorithm or the detection algorithm, depending on the phase.

2 We introduce this "dampening" effect in the remarks of Section 4.1.2

Algorithm 1: Attack_Detection

input : CO: ID of currently received content object;
S: reference sample set;
$snap\_size$: size of measurement window;
$analyzed\_cos$: number of analyzed content objects;
co_count: array of counters for content objects in S;
co_freq: array of frequencies of content objects; τ: threshold value;
σ_τ: standard deviation of values assumed by τ;
σ_{max}: maximum acceptable value for σ_{max} to be considered stable;
output: **true** if under attack, **false** otherwise

```
1: result_of_detection ← false
2: Increment analyzed_cos
3: if CO ∈ S then
4:    Increment co_count[CO]
5:    if ((analyzed_cos + 1) mod snap_size) = 0 then
6:       if Learning_step(analyze_cos, snap_size, S, co_freq) then
7:          // learning phase is not completed yet
8:          for all co ∈ S do
9:             co_count[co] ← 0
10:         end for
11:      else
12:         // learning phase is completed
13:         result_of_detection ←
            Attack_Test(CO, S, snap_size, co_freq, co_count)
14:      end if
15:   end if
16: end if
17: return result_of_detection
```

6.2.1 *Analyzing object frequencies*

Algoritms 2 and 3 are invoked at fixed intervals defined in the number of object observed by the node after their last execution. We call this number *snap_size*.[3]

This approach is due to the need to extract features from forwarded traffic, such as the frequencies of content objects. Such features are then used to determine whether an attack is in progress.

Once every *snap_size* content objects, the algorithm calculates the frequencies of samples; frequencies actually approximate the theoretical probability in Equation 2. Therefore, *snap_size* must be large enough to approximate the probability of samples with a sufficient level of precision. We observe that *snap_size* depends:

- on the value of the Zipf parameter α, because higher values of α lead to a faster decrease of the Zipf function, thus reducing the precision of computed frequencies for samples positioned at the tail of the Zipf distribution;

3 The variable *snap_size* is set at router deployment.

Algorithm 2: Learning_Step

input : $analyzed\_cos$: n. of analyzed cont. objects;
$snap\_size$: size of measurement interval;
S: set of content objects;
co_freq: array of freq. of cont. objects;
τ: threshold value;
σ_τ: st. deviation of values assumed by τ;
σ_{max}: max acceptable value for σ_{max};
output: true if learning phase is not yet complete

1: **if** $\sigma_\tau < \sigma_{max}$ **and** $\frac{analyzed\_cos}{snap\_size} > 20$ **then**
2: **return false**
3: **else**
4: $\delta_m \leftarrow 0$
5: **for all** $i \in S$ **do**
6: $prev\_co\_count \leftarrow co\_freq[i] \cdot (analyzed\_cos - snap\_size)$
7: $co\_freq[i] \leftarrow \frac{prev\_co\_count + co\_count[i]}{analyzed\_cos}$
8: $\delta_m \leftarrow \delta_m + \left| \frac{co\_count[i]}{snap\_size - co\_freq[i]} \right|$
9: **end for**
10: Update τ as in Eq. 5
11: Update σ_τ as in Eq. 4
12: **return true**
13: **end if**

- on the selected samples, because samples collected from the tail of the distribution would lead to unpredictable results, due to the imprecise approximation of frequencies.

This approach in selecting samples makes the detection independent on the traffic distribution. Samples are simply chosen within the most frequent objects, regardless of the distribution. This also provides a positive side effect: the use of samples with probabilities that are more likely to be correctly approximated; besides, this approach allows to reduce *snap_size*[4] along with the detection time. In our simulations, modeling the Internet traffic with a Zipf distribution, and with Zipf's $\alpha = 0.9$, we found that a *snap_size* of 10000 objects was enough to approximate the frequency of samples.

6.2.2 *Learning phase*

The aim of the learning algorithm is to extract features from forwarded traffic. The main feature computed by the learning algorithm is τ, i.e., the threshold for attack detection. This value corresponds to what is computed by Equation 5. Similarly, the variance of the current threshold is computed using Knuth's formulas (see Equation 4).

4 As long as the smaller *snap_size* does not excessively impact the precision of frequencies.

Algorithm 3: Attack_Test

input : CO: Id of received cont. object;
S: reference sample set;
snap_size: size of measur. window;
co_freq: array of freq. of cont. objects;
co_count: array of counters
for cont. objects in S;
output: True if under attack, False otherwise

1: result_of_detection ← **false**
2: $\delta_m \leftarrow 0$
3: **for all** $i \in S$ **do**
4: $\quad \delta_m \leftarrow \delta_m + \left| \frac{co\_count[i]}{snap\_size - co\_freq[i]} \right|$
5: **end for**
6: **if** $\delta_m > \tau$ **then**
7: $\quad$ result_of_detection ← **true**
8: **end if**
9: **for all** $co \in S$ **do**
10: $\quad$ co_count[co] ← 0
11: **end for**
12: **return** result_of_detection

The end of the learning phase is determined by Learning_Step() – in particular, once enough samples have been obtained, and their variance is below a threshold (σ_{max}), new invocation of the learning algorithm do nothing and simply return false.

The number of samples that should be collected before ending the learning phase is a multiple of *snap_size*, since each *snap_size* represent a measurement – i.e., a *learning step*. If the number of learning steps is too small, the threshold would be imprecise, because based on a mean and a variance that are computed on a small number of measurements. In our simulations, we ended the learning phase after 20 steps (as in Line 1 of Learning_Step()).

6.2.3 *Attack detection phase*

Algorithm 3 (Attack_Test()) works similarly to Learning_Step(). It calculates the δ_m value for the current measurement, but in this case δ_m is compared with the threshold. If $\delta_m > \tau$, the algorithms reports an attack.

6.3 EVALUATION

To evaluate our detection mechanism, we implemented it on top of ndnSIM. The behavior of the adversary is the same as described in Section 5.5. We considered different values of ϑ (between 1 and 100) and γ (0.02 to 1.7), to assess how these parameters affect the accuracy of our detection algorithm.

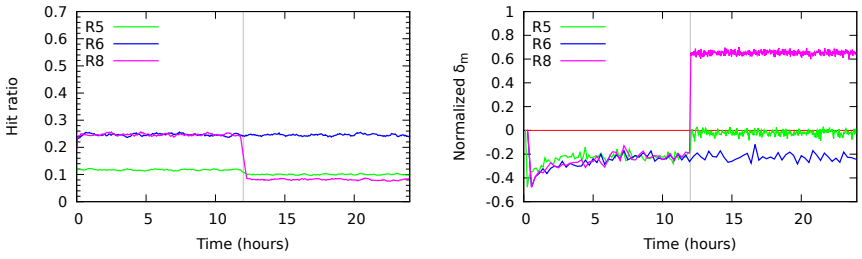
The results of our simulations are summarized in figures 14, 15, 16, and 17, for XC topology with LRU, DFN with LRU, XC with LFU-DA, and DFN with LFU-DA, respectively.

For the sake of clarity, we only report the results for $\vartheta = 1$ and $\vartheta = 100$; intermediate values provide very similar results.

The results reported in figures 14 and 16 (i.e. XC topology) are all for $\gamma = 1.7$, while the results in figures 15 and 17 (i.e. DFN topology) are for $\gamma = 2.5$. We measure the performance of our detection algorithms in terms of *normalized* δ_m, defined as $(\delta_m/\tau) - 1$.

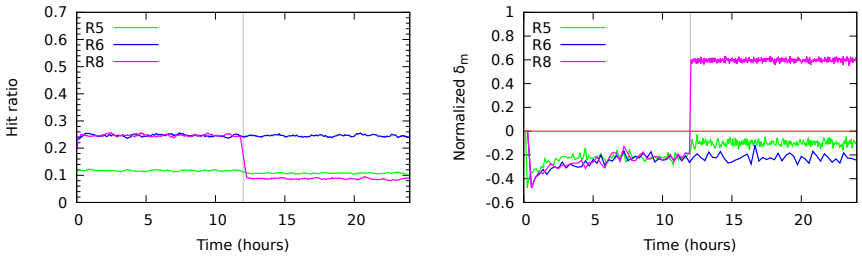
Simulation results show that our technique can quickly determine when a router is under attack or, more generally, *affected* by the attack. In particular, figures 14 and 15 show that routers using LRU successfully detect the attack for all values of ϑ – even those for which CacheShield is ineffective. These results confirm that topology characteristics only have limited impact on the performance of our detection algorithm.

We notice that for XC and $\vartheta = 100$, the algorithm only barely "flags" R5 as under attack, i.e. the standard deviation is beyond the threshold (see Figure 6.14(b)). This is due to the fact that the attack has only minimal impact on R5.



(a) Cache hit ratio, $\vartheta = 100$

(b) Normalized δ_m, $\vartheta = 100$



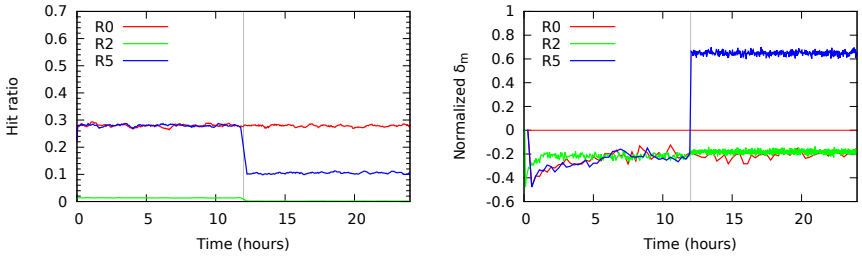
(c) Cache hit ratio, $\vartheta = 1$

(d) Normalized δ_m, $\vartheta = 1$

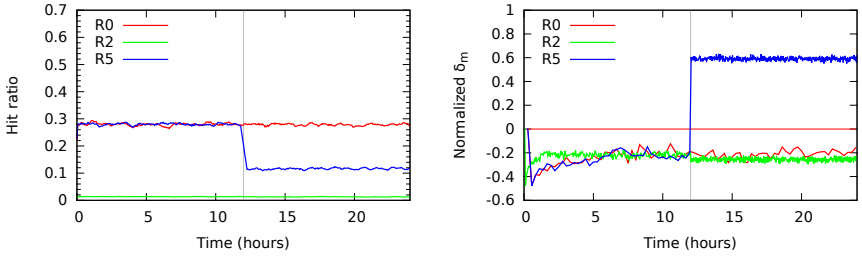
Figure 14: Attack detection on XC topology using LRU, $\gamma = 1.7$.

Switching to LFU-DA, our detection algorithm performs equally well (see figures 16 and 17). Even though the impact of the attacks is more limited than with LRU, our mechanism detects them on R8 and R5 on the XC topology, and on R5 on DFN.

Attacks with very low rate, such as $\gamma = 0.02$, are also detected by our approach, as shown in Figure 18.

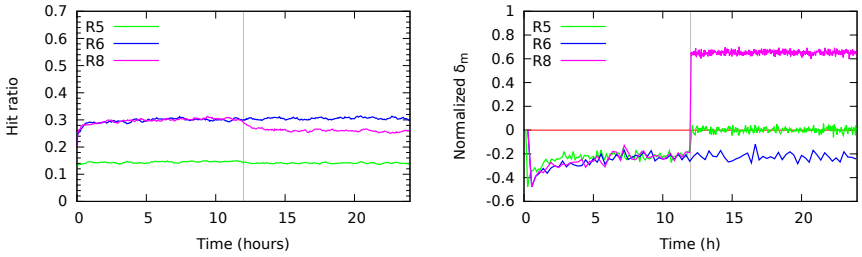


(a) Cache hit ratio, $\vartheta = 100$ (b) Normalized δ_m, $\vartheta = 100$

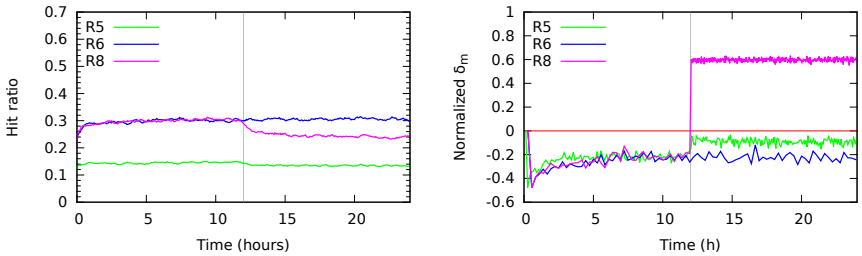


(c) Cache hit ratio, $\vartheta = 1$ (d) Normalized δ_m, $\vartheta = 1$

Figure 15: Attack detection on DFN topology using LRU, $\gamma = 2.5$.



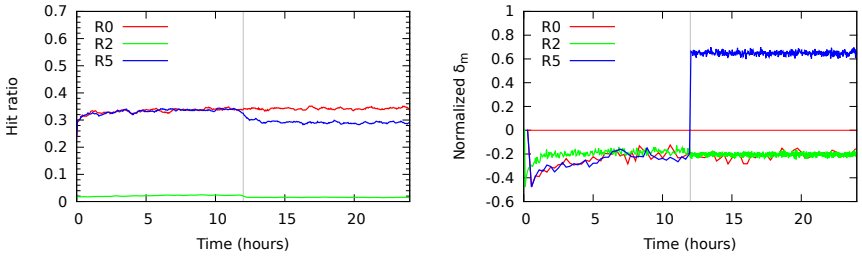
(a) Cache hit ratio, $\vartheta = 100$ (b) Normalized δ_m, $\vartheta = 100$



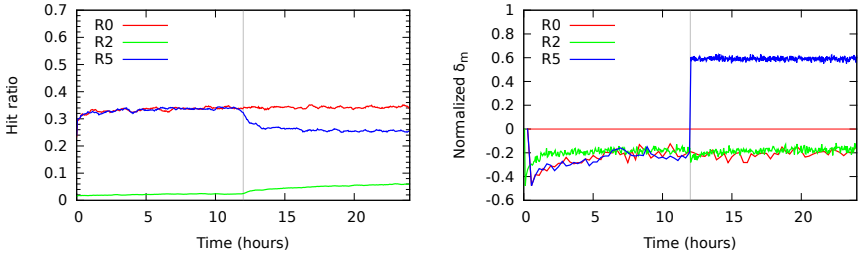
(c) Cache hit ratio, $\vartheta = 1$ (d) Normalized δ_m, $\vartheta = 1$

Figure 16: Attack detection on XC topology using LFU-DA, $\gamma = 1.7$.

Low rate detection was one of the headline features of the work of Park et al. [36], and our results show that our algorithm performs equally well. Additionally, the cost of our approach both in terms of computation and communication is significantly lower than that



(a) Cache hit ratio, $\vartheta = 100$ (b) Normalized δ_m, $\vartheta = 100$



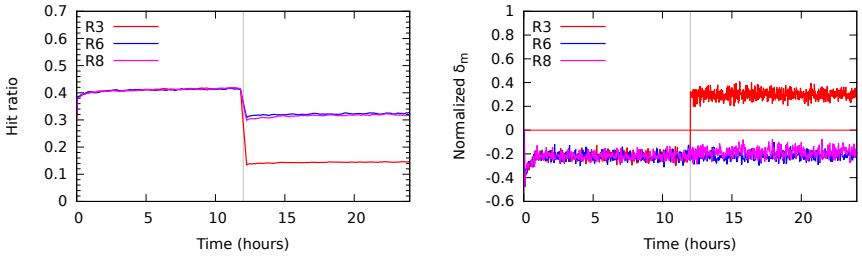
(c) Cache hit ratio, $\vartheta = 1$ (d) Value δ_m, $\vartheta = 1$

Figure 17: Attack detection on DFN topology using LFU-DA, $\gamma = 2.5$.

of Park et al. In fact, for each packet, their approach requires the computation of a collision-resistant hash function.[5]

Moreover, their algorithm computes the rank of an $n \times n$ matrix once every 2,000 requests (the detection window for which they get the best performance), where n is $O(\sqrt{N})$, with N indicating the size of the cache. Since average case complexity for Gaussian elimination is $O(n^{2.5})$ with worst case $O(n^3)$ [35], the cost of the algorithm is super-linear in the size of the cache. This means that increasing the size of the cache on a router may not provide the expected benefits due to the increased cost of the detection algorithm.

Hence, compared to [36], our approach is less expensive both per-packet (we do not compute any collision-resistant hash function) and per-detection-window.



(a) Cache hit ratio, $\vartheta = 1$ (b) Normalized δ_m, $\vartheta = 1$

Figure 18: Attack detection on XC topology using LFU-DA, $\gamma = 0.02$.

5 SHA-1 was suggested as a viable candidate in their paper.

7 CONCLUSION

In this thesis, we have shown that cache pollution attack is a realistic threat on NDN. Our experiments confirm that the attacks previously evaluated on very small topologies extend to larger and more realistic networks, with no additional effort.

We have analyzed existing approaches against cache pollution attacks. We found that some approaches designed for IP are not easily applicable to NDN. Compared with previous proactive approaches, we have argued that detecting and limiting the attack may prove to be a better strategy. We provided a two-phase detection technique. In the first phase (*learning phase*), nodes observe and analyze the Internet traffic (under the assumption that they are not already under attack). In the second phase (*detection phase*), nodes use the features extracted during the previous phase to detect the attack. An interesting property of our detection mechanism is that – although we need to collect statistics for a number of objects – the amount of storage required for detection is independent from the cache size. Moreover, none of such objects (samples) need to be requested by the adversary in order to detect the attack: any possible request that follow a different traffic distribution would inevitably change the frequency of samples.

Simulations show that our lightweight detection technique provides accurate results. Our results apply to different topologies and are independent from the distribution of the traffic routed by each node.

While we do not address attack reaction techniques, there are some possible approaches that could be further investigated. For example, a node may collect separate statistics for different interfaces, identify which of them are carrying the anomalous traffic, and then limit their rate. Alternatively, the node could cache only content that would not significantly change the detection statistics (this would probably require some structural changes to the detection mechanism).

These – and possible other countermeasures – are potentially more expensive in terms of time and space complexity, because they might require some extra effort (and data structures) to counteract the attack. Nonetheless, the advantage of a lightweight detection mechanism is the possibility of enabling a separate (more expensive) reaction mechanism only when needed. So, the detection mechanism can work alone until it detects an attack, and then activate the countermeasure.

Our analysis has been performed using synthetic traffic and a static environment, in which there is a fixed number of content objects, all of them requested with a predefined probability (according to the

traffic distribution). This approach is also common in other analyses (for example, in [42]) and useful to simplify the research of possible solutions.

Nonetheless, relaxing the assumption of static environment is a mandatory step toward the adoption of real traces and real network environments. As future work, it would be interesting to study how this approach can be extended to support dynamic environments, where the probability of objects being requested changes continuously. For this purpose, this mechanism could be adjusted in order to run the learning phase and detection phase concurrently: once the traffic has been first analyzed, detection could be performed without stopping the learning process. The trade-off would be in how much the frequencies could change, over time, without and under attack.

As a further step, this mechanism could be implemented in a real testbed and with real traces, to assess its behavior without any side-effect that might be due to limitations of simulations environments.

Finally, in an architecture that supports ubiquitous caching, where any single node can keep a copy of forwarded content, there is the potential for a collaborative detection of cache pollution attacks; in fact, nodes could communicate to have a better knowledge of possible sources of attack. All this – to the best of our knowledge – has yet to be investigated.

BIBLIOGRAPHY

[1] Named data networking project (NDN). http://named-data.org. Retrieved Jan. 2013.

[2] XIA - eXpressive Internet Architecture. http://www.cs.cmu.edu/~xia/. Retrieved Jan. 2013.

[3] CCNx protocol. http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html. Retrieved Jan. 2013.

[4] ChoiceNet - Evolution through Choice. https://code.renci.org/gf/project/choicenet/. Retrieved Jan. 2013.

[5] Interest message specification. URL http://www.ccnx.org/releases/latest/doc/technical/InterestMessage.html. Retrieved Jan. 2013.

[6] MobilityFirst FIA Overview. http://mobilityfirst.winlab.rutgers.edu. Retrieved Jan. 2013.

[7] Nebula. http://nebula.cis.upenn.edu. Retrieved Jan. 2013.

[8] ns-3. http://www.nsnam.org. Retrieved Jan. 2013.

[9] Squid Web caching proxy. http://www.squid-cache.org/.

[10] A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN, October 2012. URL http://www.named-data.net/techreport/TR005-ndnsim.pdf.

[11] W. Ali, S.M. Shamsuddin, and A.S. Ismail. A survey of web caching and prefetching. *Int. J. Advance. Soft Comput. Appl*, 3(1): 18–44, 2011.

[12] M.F. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. *SIGMETRICS Performance Evaluation Review*, 27(4):3–11, 2000.

[13] A. Balamash and M. Krunz. An overview of web caching replacement algorithms. *Communications Surveys & Tutorials, IEEE*, 6(2):44–56, 2004.

[14] M. Basseville, I.V. Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, NJ, 1993.

[15] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, pages 126–134, 1999.

[16] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[17] A. Compagno. Poseidon: Mitigating interest flooding attacks in named data networking. Master's thesis, University of Padua, 2012.

[18] A. Compagno, M. Conti, P. Gasti, and G. Tsudik. Ndn interest flooding attacks and countermeasures. *In 28th Annual Computer Security Application Conference (ACSAC 2012)*, 2012. (poster).

[19] M. Conti, P. Gasti, and M. Teoli. A lightweight mechanism for detection of cache pollution attacks in named data networking. *Computer Networks, Special Issue on Information Centric Networking*, 2013 (submitted).

[20] G. Dán and N. Carlsson. Power-law revisited: A large scale measurement study of p2p content popularity. In *Proceedings of the 9th international conference on Peer-to-peer systems*, pages 12–12. USENIX Association, 2010.

[21] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic. Pollution attacks and defenses for internet caching systems. *Comput. Netw.*, 52 (5):935–956, April 2008. ISSN 1389-1286. doi: 10.1016/j.comnet.2007.11.019. URL `http://dx.doi.org/10.1016/j.comnet.2007.11.019`.

[22] P.J. Denning and S.C. Schwartz. Properties of the working-set model. *Communications of the ACM*, 15(3):191–198, 1972.

[23] J. Dilley and M.F. Arlitt. Improving proxy cache performance: Analysis of three replacement policies. *IEEE Internet Computing*, 3(6):44–50, 1999.

[24] J. Dilley, M. Arlitt, and S. Perret. Enhancement and validation of squid's cache replacement policy. *HP Laboratories Technical Report HPL*, (69), 1999.

[25] B. Eriksson, P. Barford, J. Sommers, and R. Nowak. A learning-based approach for ip geolocation. In *PAM*, pages 171–180, 2010.

[26] K. Gammon. Networking: four ways to reinvent the internet. *Nature*, 463(7281):602–604, 2010.

[27] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Internet Measurment Conference*, pages 35–48, 2005.

[28] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang. Does internet media traffic really follow zipf-like distribution? In *SIGMETRICS*, pages 359–360, 2007.

[29] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang. The stretched exponential distribution of internet media access patterns. In *PODC*, pages 283–294, 2008.

[30] O. Heckmann, M. Piringer, J. Schmitt, and R. Steinmetz. On realistic network topologies for simulation. In *ACM SIGCOMM MoMeTools*, pages 28–32. ACM Press, 2003.

[31] C.S. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12), 2001.

[32] D.E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1997. ISBN 0-201-03822-6.

[33] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda. Privacy risks in named data networking: what is the cost of performance? *Computer Communication Review*, 42(5): 54–57, 2012.

[34] M.E.J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[35] M. Olschowka and A. Neumaier. A new pivoting strategy for gaussian elimination. *Linear Algebra and its Applications*, (240): 131–151, 1996.

[36] H. Park, I. Widjaja, and H. Lee. Detection of cache pollution attacks using randomness checks. In *ICC*, pages 1096–1100. IEEE, 2012. ISBN 978-1-4577-2052-9.

[37] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4):374–398, 2003.

[38] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2001. ISBN 0-201-61570-3.

[39] S. Son and V. Shmatikov. The hitchhiker's guide to dns cache poisoning. In *SecureComm*, pages 466–483, 2010.

[40] Eric W. Weisstein. Logistic equation. `http://mathworld.wolfram.com/LogisticEquation.html`, . Retrieved on Jan. 2013.

[41] Eric W. Weisstein. Zipf's law, . URL `http://mathworld.wolfram.com/ZipfsLaw.html`. Retrieved Jan. 2012.

[42] M. Xie, I. Widjaja, and H. Wang. Enhancing cache robustness for content-centric networks. In *INFOCOM*, 2012.

[43] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, E. Uzun, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh. Named Data Networking (NDN) project. Technical report, PARC, 2010.

[44] G. Zipf. In *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.