Computer Networks 57 (2013) 3178–3191

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A lightweight mechanism for detection of cache pollution attacks in Named Data Networking

Mauro Conti [a,*], Paolo Gasti [b], Marco Teoli [a]

[a] University of Padua, Department of Mathematics, Via Trieste, 63-35131 Padua, Italy
[b] New York Institute of Technology, School of Engineering and Computing Sciences, 1855 Broadway-New York, NY 10023-7692, USA

ARTICLE INFO

Article history:
Received 8 January 2013
Received in revised form 16 June 2013
Accepted 10 July 2013
Available online 28 August 2013

Keywords:
Named data networking
Cache pollution attack
Security

ABSTRACT

Content-Centric Networking (CCN) is an emerging paradigm being considered as a possible replacement for the current IP-based host-centric Internet infrastructure. In CCN, named content – rather than addressable hosts – becomes a first-class entity. Content is therefore decoupled from its location. This allows, among other things, the implementation of ubiquitous caching.

Named-Data Networking (NDN) is a prominent example of CCN. In NDN, all nodes (i.e., hosts, routers) are allowed to have a local cache, used to satisfy incoming requests for content. This makes NDN a good architecture for efficient large scale content distribution. However, reliance on caching allows an adversary to perform attacks that are very effective and relatively easy to implement. Such attacks include cache poisoning (i.e., introducing malicious content into caches) and cache pollution (i.e., disrupting cache locality).

This paper focuses on cache pollution attacks, where the adversary's goal is to disrupt cache locality to increase link utilization and cache misses for honest consumers. We show, via simulations, that such attacks can be implemented in NDN using limited resources, and that their effectiveness is not limited to small topologies. We then illustrate that existing proactive countermeasures are ineffective against realistic adversaries. Finally, we introduce a new technique for detecting pollution attacks. Our technique detects high and low rate attacks on different topologies with high accuracy.

1. Introduction

In recent years there have been several efforts to design a viable replacement for the current IP-based Internet [23,32,22,24,9]. 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. In particular, there have been significant effort to provide better mobility, scalability and efficient content distribution. Additionally, strong security has been one of the main design requirements for these architectures.

Named Data Networking (NDN) [23], is one of these architectures, as well as a prominent example of Content-Centric Networking (CCN) (also known as Information-Centric Networking, or ICN). In CCN, content – rather than hosts – occupies the central role in the communication architecture.

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

[*] Corresponding author. Tel.: +39 049 827 1488.
E-mail address: conti@math.unipd.it (M. Conti).

1389-1286/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2013.07.034

store forwarded content for subsequent retrieval. This allows NDN to transparently and automatically implement efficient multicast, content replication, load balancing and fault tolerance.

However, pervasive caching exacerbates security problems related to shared caches, including privacy [1], pollution [11] and poisoning [29]. In this paper 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) [33,11]. Deng et al. 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 magnitude [11]. 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 cache pollution on routers and on end users is significant and deserves careful study.

Contribution. In this paper we address cache pollution attacks in NDN. In particular:

- Previous work on cache pollution in NDN/CCN only focuses on small topologies (one to nine routers [33,11]); in this paper, we show that the attack scales 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 [33]. Under our attack, existing techniques for cache robustness do not provide any 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) [18].

We emphasize that the goal of this paper 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.

Organization. The rest of the paper is organized as follows. In Section 2 we provide a short overview of NDN. In Section 3 we introduce previous work on cache pollution, and overview state-of-the-art detection and mitigation techniques for IP caching proxies and NDN. In Section 4 we present the threat model and adversarial strategy considered in our work, while in Section 5 we describe the setting used in our simulations. In Section 6 we assess the effectiveness of cache pollution attacks in NDN. Next, we describe and evaluate our detection mechanism in Section 7. We conclude in Section 8.

2. NDN overview

NDN is based on the pull model, where content is injected into the network only in response to a consumer's explicit request. NDN supports two types of messages: *interests* and *content objects* [7]. Each content object has a name (possibly human-readable), a payload and a digital signature computed by the content producer. (Content objects contain additional fields not relevant for this work). Names are made up of one or more components, with hierarchical structure. In NDN notation, "/" separates name components, e.g., `/ndn/com/cnn/politics/front-page`. Large pieces of content are split into fragments with predictable names: e.g., fragment 5 of Alice's photo could be named `/flickr/alice/photo-1.jpg/5`. In case of multiple content objects under a given name, optional control information can be carried within the interest to restrict desired content.

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. *Forwarding Information Base* (FIB) is a lookup table used to determine interfaces for forwarding incoming interests. Multiple concurrent entries for the same prefix are allowed, supporting multipath delivery. Each NDN router maintains a Pending Interest Table (PIT) – a lookup table containing outstanding [*interest*, *arrival-interfaces*] entries. For efficiency's sake, multiple pending interests for the same content are *collapsed*: only the first interest is forwarded, and the returned content is sent back to all the arrival interfaces. 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 behavior described above applies mainly to content that is requested for the first 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 (CS). (In the rest of the paper we use the terms CS and cache interchangeably.) 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 allows NDN to implement efficient large-scale content distribution. Producers do not need to issue a copy of their content for each consumer request; consumers always retrieve the closest available copy of requested content.

3. Related work

Cache pollution attacks have been widely studied in IP-based networks, mostly with respect to caching of web traffic. In [11], Deng et al. propose two generic classes of cache pollution attacks: locality-disruption and false-locality. In a locality-disruption attack, the goal of the adversary is to maliciously alter cache content locality, e.g., by flattening the distribution of content requests. This attack can be implemented issuing requests for (new) non-popular content. A false-locality attack 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. Deng et al. 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.

In [11], the authors also propose a detection mechanism for both types of attack. In case of false locality, their technique keeps track of the absolute number of repeated requests, and of the ratio of repeated requests over the number of cache hits. When both metrics exceed given thresholds, corresponding sources are identified as malicious. Detection of locality disruption is performed measuring hit ratio and lifetime of cached content. When they drop below predefined values, the cache is considered under attack. Attack mitigation relies on identifying the source of malicious requests. For this purpose, the authors use content requestors' IP addresses.

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.

Park et al. [27] consider pollution attacks in the context of content caching. The authors propose a detection mechanism based on randomness of content requests, measured on caching nodes (e.g., NDN routers). Each node stores statistics on incoming requests in a matrix. When the entropy of the matrix falls below a threshold, the router detects an attack in progress. 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 [27] 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.

The storage cost of the approach of Park et al. is $O(\sqrt{N})$, where N is the size of the cache. 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; and (2) the computation of the rank of the matrix. This is in contrast with our approach, for which the amount of storage required for detection is independent from the cache size; moreover, we do not require the computation of collision-resistant hash functions.

Meta cache algorithms have been proposed (see, e.g., [20,8]) to improve the efficiency of multi-level caching architectures, such as web caching and CCNs. However, to the best of our knowledge, none of these techniques is designed to limit the effect of parties exhibiting adversarial behavior. Laoutaris et al. [21] introduce a technique that mitigates the effects of selfish behavior on multi-level caching. More related to our work, the authors also briefly discuss how to extend their approach towards mistreatment-resilient caching, although this is mostly left to future work.

As the time of this writing, CacheShield [33] is the only countermeasure for cache pollution attacks specifically designed for (and evaluated on) NDN. Given the relevance of CacheShield to our work, we provide a detailed overview below.

According to Breslau et al. [6] Web page requests on the Internet follow a Zipf-like [34] distribution. The same distribution is assumed in many subsequent papers (e.g., [11,27,33]). Nonetheless, we note that other works have argued that, under some circumstances, this does not apply to Internet traffic in general. For example, Guo et al. [16] claim that Internet traffic follows a Stretched Exponential distribution [17], and provide some results to support their thesis. Dán and Carlsson [10] provide an extensive analysis of peer-to-peer traffic. They conclude that content distribution follows a power-law with exponential cutoff [15]. In evaluating our approach, we 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.

Cache replacement algorithms play an important role in the analysis of cache pollution attacks. Deng et al. [11] observe that the impact of malicious users with respect to cache pollution strongly depends on the replacement algorithm in use. (This finding is confirmed by our results.) Different categorization of caching algorithms has been provided in several surveys [28,5,3]. We adopt two replacement techniques covered by the aforementioned literature: recency-based and frequency-based policies.

For the study performed in this work, we selected LRU (Least Recently Used) from the former class: when the cache is full, LRU always replaces the object that was requested less recently. From the latter class, we adopted LFU-DA (Least Frequently Used with Dynamic Aging). Both the algorithms have been selected, after extensive analyses [4,13,12], by Squid [30], one of the most popular caching proxies. Although caching proxies and routers' CS do not have identical goals and restrictions, we argue that cache replacement policies that perform well in the former are a good starting point for the latter.

3.1. CacheShield

In [33] Xie et al. introduce CacheShield, 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. 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 withCacheShield. 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 [31]; 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 [33], 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.

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., [33,11]). (While we do not exclude that attacks might come from compromised routers, we leave the investigation of this as future work.) 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 ratio of attack rate to legitimate users rate with γ. As an example, $\gamma = 0.02$ indicates that the adversary issues 2% of the total number of interests issued by legitimate users.

We argue that, under the assumption above, the adversary strategy for generating interests in Xie et al. [33] 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 paper, 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 6) this attack is not very effective against LFU-based cache replacement algorithms.

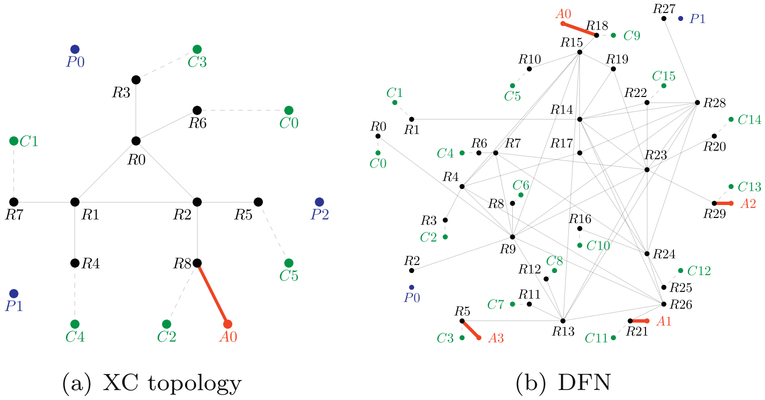
In order to implement a more effective attack (false-locality) taking into account the aforementioned bandwidth limitations, the adversary could focus on a carefully-chosen small subset of the available content. We refer to this strategy as *smart-selection*. Assuming Zipf-like traffic distribution,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

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).

5. Evaluation environment

We evaluate attacks and countermeasures discussed in this paper via simulations. We rely on ns-3 [25], a well-known open-source discrete-event network simulator. ns-3 supports wired and wireless networks based on IP or other protocols. NDN functionalities are provided by ndnSIM [2], a module for ns-3 developed at UCLA as part of the NDN project. The structure of ndnSIM is cleaner and more extensible than that of other NDN simulation environments [2]. 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.).

For our simulations, we consider two topologies, as illustrated in Fig. 1: Xie-complex (XC) and the German Research Network (DFN). XC corresponds to the "Complex Network" considered in [33], and allows us to compare our results with those in [33]. The DFN topology has been identified in previous work as a meaningful topology for



(a) XC topology

(b) DFN

Fig. 1. Network topologies used for simulations.

simulations [18]. This allows us to provide a more realistic assessment of our technique.

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. [33], 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 [33]. We also include "no attack" as a reference, and indicate it with $\vartheta = 0$.

In [33], 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.

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

6. Effects of cache pollution attacks

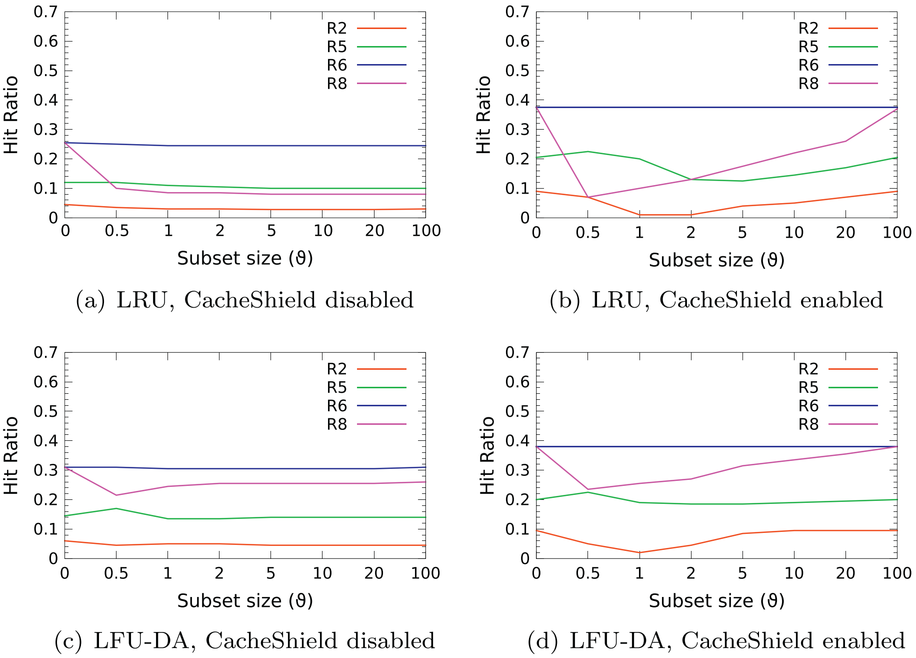
In this section, we assess how attacks based on broad-selection and smart-selection affect our topologies. In particular, we are interested in determining whether the adversary can have an impact on cache hit ratio, and consequently increase average hop count of consumers' requests. We emphasize that, as in Xie et al. [33], 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.

Effects of the Attack on Hit Ratio. Fig. 2 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 Fig. 2 (as well as in other figures in this paper), 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 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. [33]. 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 Fig. 2(b)), compared to 0.1 without CacheShield (see Fig. 2(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 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



(a) LRU, CacheShield disabled

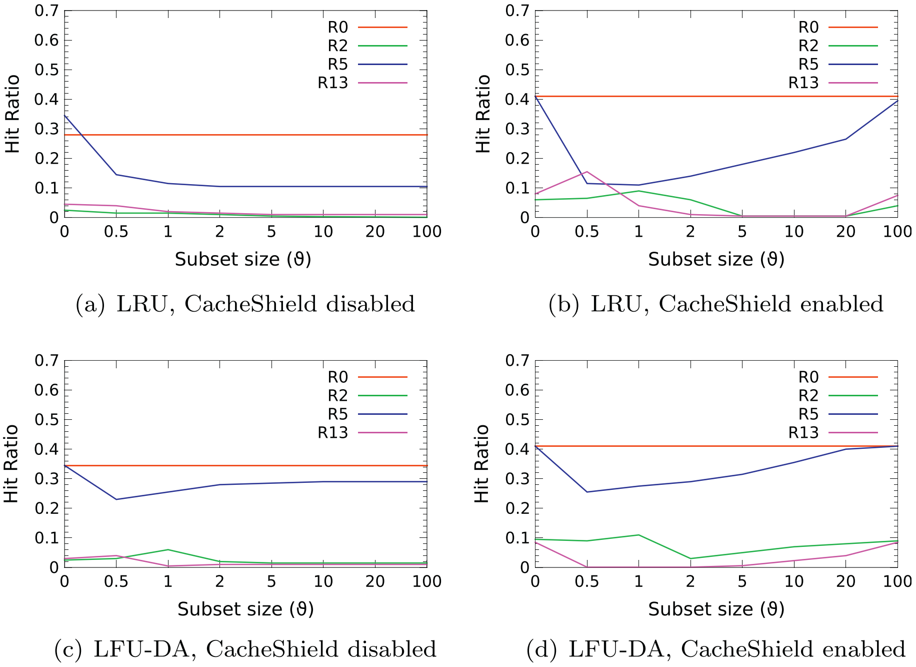
(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

Fig. 2. Hit-ratio on XC topology using different subset sizes, for $\gamma = 1.7$.

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 Fig. 2(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 Fig. 3. For the DFN and LFU-DA, the value ϑ for which router R2 experiences the best performance is $\vartheta = 1$: both with CacheShield (Fig. 3)) and without CacheShield (Fig. 3(c)). This is also true for LRU with CacheShield (Fig. 3(b)), but not for LRU without CacheShield (Fig. 3)(a). We also



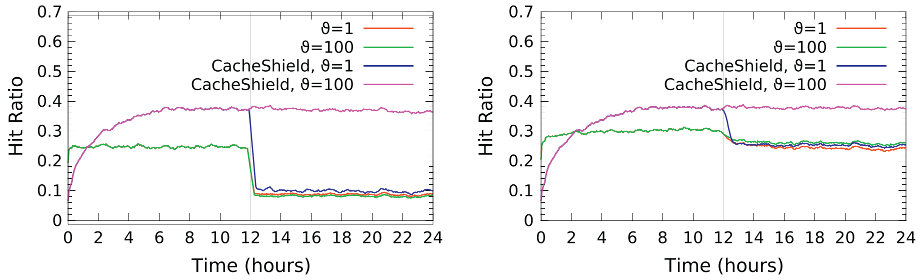
(a) LRU, CacheShield disabled

(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

Fig. 3. Hit-ratio on DFN topology using different subset sizes, for $\gamma = 2.5$.



(a) Router 8, LRU

(b) Router 8, LFU-DA

Fig. 4. Effects of attacks on hit-ratio on XC topology for selected routers, with $\gamma = 1.7$.

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 [27], is not sufficient to determine the effectiveness of pollution attacks and detection/countermeasures.

Figs. 4 and 5 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 [33] and reported in Section 3.1, 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 h 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. 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.

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 [14].[1] Therefore, we consider this a meaningful metric for determining the impact of the attack on consumers.

Fig. 6 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 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 Fig. 6). 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. This 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. Fig. 7 summarizes our findings with respect to average hop count on DFN.

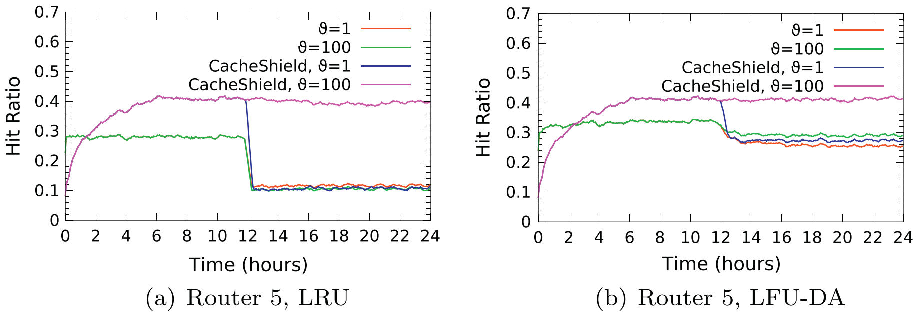
7. Detection of cache pollution attack

Proactive techniques, such as CacheShield, may not be the most appropriate approach against cache pollution attacks. CacheShield must store a non-negligible 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 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 then trigger possibly 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.

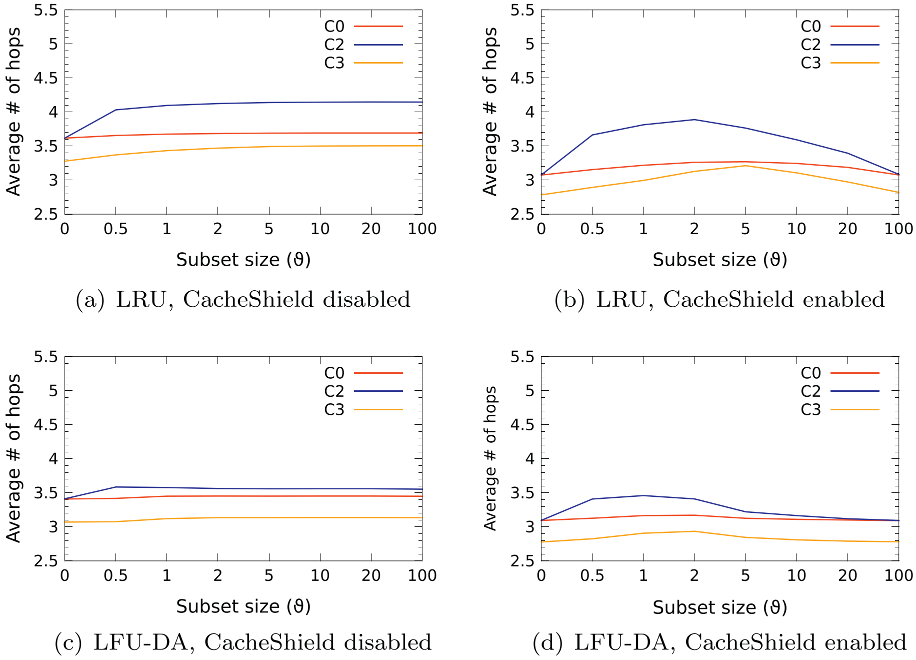
[1] Not surprisingly, experiments on the official NDN testbed confirm that this holds also for NDN.



(a) Router 5, LRU

(b) Router 5, LFU-DA

Fig. 5. Effects of attacks on hit-ratio on DFN topology for selected routers, with $\gamma = 2.5$.



(a) LRU, CacheShield disabled

(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

Fig. 6. Average number of hops on XC topology using different subset sizes, for $\gamma = 1.7$. Results are illustrated per consumer.

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.

7.1. Sampling interests distribution

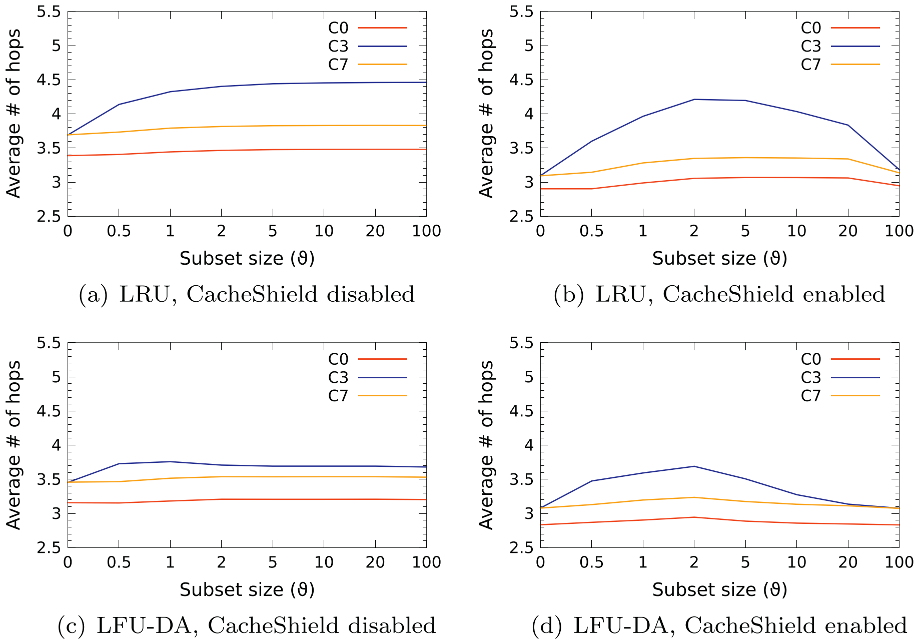
Routers can learn how the forwarded traffic is distributed by counting how often each content object is returned in response to an interest, normalizing the results by the amount of forwarded traffic.[2] Let $p(i)$ be the probability value associated with content object i. We have:

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

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)

[2] 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.



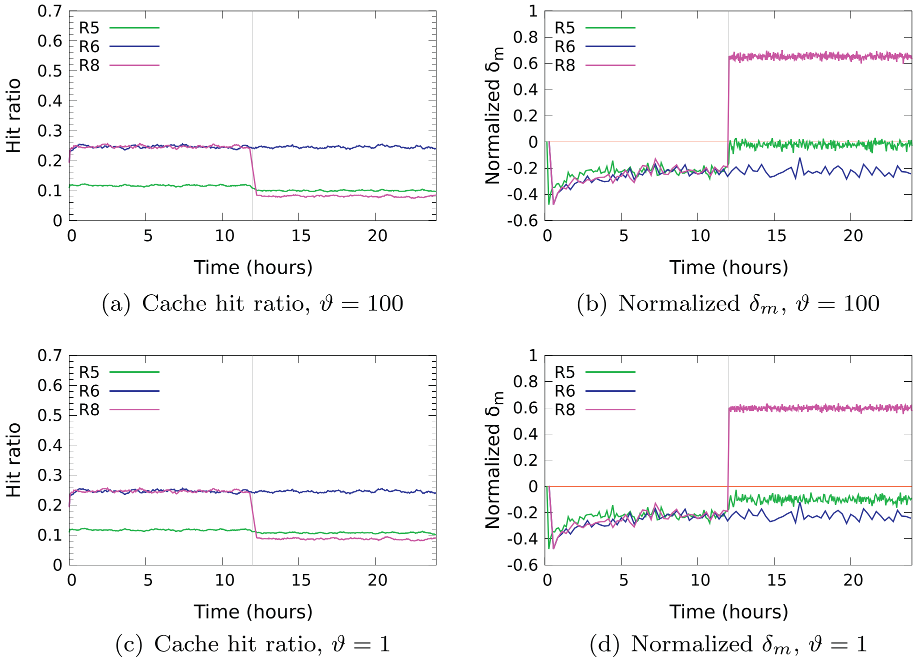
(a) LRU, CacheShield disabled

(b) LRU, CacheShield enabled

(c) LFU-DA, CacheShield disabled

(d) LFU-DA, CacheShield enabled

Fig. 7. Average number of hops on DFN network for different subset sizes, for $\gamma = 2.5$. Results are illustrated per consumer.



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

(b) Normalized δ_m, $\vartheta = 100$

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

(d) Normalized δ_m, $\vartheta = 1$

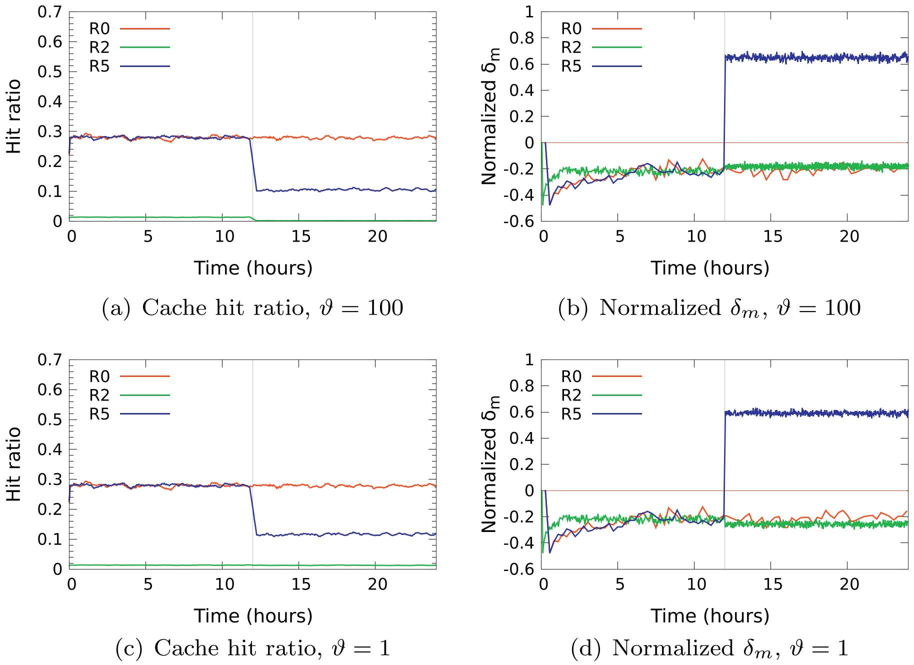
Fig. 8. Attack detection on XC topology using LRU, $\gamma = 1.7$.

amount of (still fresh) historical data and avoid reporting false positives.

Once $p(i)$ has been estimated, our approach uses it to measure variation across samples to determine "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:

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

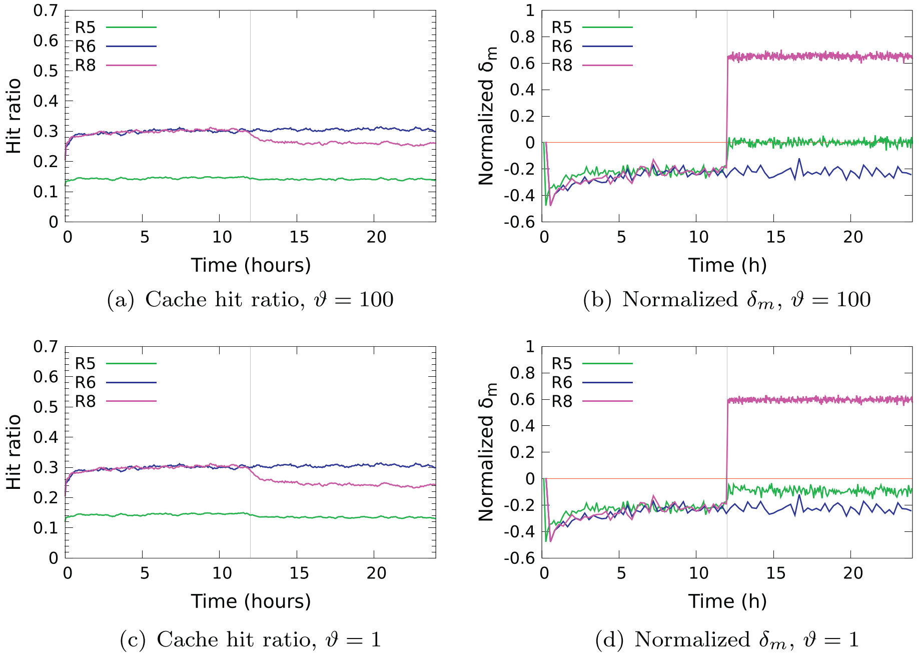
The maximum acceptable variability (threshold) as the average of all measured δ_{m_i}, plus λ times their standard deviation:



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

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

Fig. 9. Attack detection on DFN topology using LRU, $\gamma = 2.5$.



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

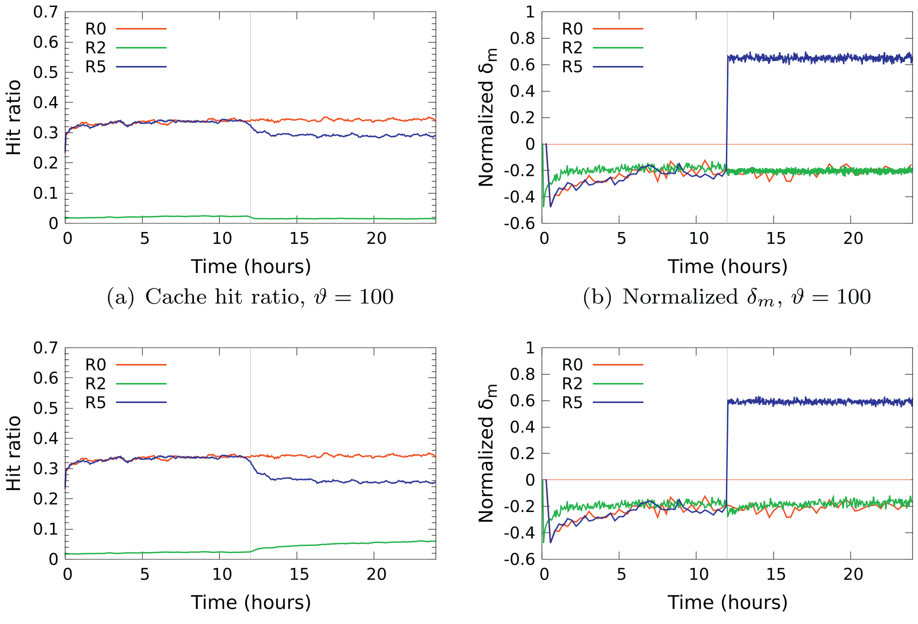
(c) Cache hit ratio, $\vartheta = 1$ (d) Normalized δ_m, $\vartheta = 1$

Fig. 10. Attack detection on XC topology using LFU-DA, $\gamma = 1.7$.

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

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 Eq. (1):



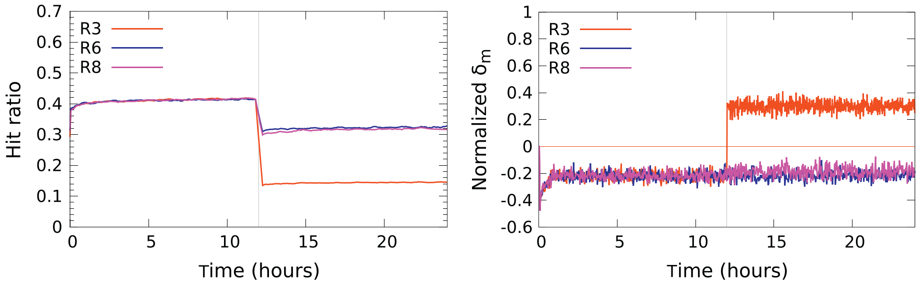
(a) Cache hit ratio, $\vartheta = 100$

(b) Normalized δ_m, $\vartheta = 100$

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

(d) Value δ_m, $\vartheta = 1$

Fig. 11. Attack detection on DFN topology using LFU-DA, $\gamma = 2.5$.



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

(b) Normalized δ_m, $\vartheta = 1$

Fig. 12. Attack detection on XC topology using LFU-DA, $\gamma = 0.02$.

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 [19] to compute the mean and variance needed to determine τ.

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. 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.

7.2. Algorithm description

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

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;
    $\tau$: threshold value;
    $\sigma_\tau$: standard deviation of values assumed by $\tau$;
    $\sigma_{max}$: maximum acceptable value for $\sigma_{max}$
    to be considered stable;
output: true if under attack, false otherwise
$result\_of\_detection \leftarrow$ false
Increment $analyzed\_cos$
if $CO \in S$ then
  Increment $co\_count[CO]$
  if $((analyzed\_cos + 1) \mathrm{mod} snap\_size) = 0$ then
    if Learning_step ($analyze\_cos, snap\_size, S, co\_freq$) then
      // learning phase not yet completed
      for all $co \in S$ do
        $co\_count[co] \leftarrow 0$
      end for
    else
      // learning phase completed
      $result\_of\_detection \leftarrow$ Attack_Test
        ($CO, S, snap\_size, co\_freq, co\_count$)
    end if
  end if
end if
return $result\_of\_detection$
```

The algorithm first checks whether CO is in S. If it is so, the algorithm updates the corresponding counter ($co\_count[CO]$) and invokes the learning algorithm. The aim of the learning algorithm is to extract features from forwarded traffic. Such features are then used, once every *snap_size* content objects, to determine whether an attack is in progress. (*snap_size* is set at router deployment.) 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 Eq. (1), and is determined using Knuth's algorithm [19]. Similarly, the variance of the current threshold is computed using Knuth's algorithm.

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.

Algorithm 3 (*Attack*_Test()) works in a similar fashion: it calculates the δ_m value for the current measurement, and compares it with the threshold.

7.3. 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 6. We considered different values of ϑ (between 1 and 100) and γ (0.02–2.5, i.e., the combined bandwidth available to the adversary is up to 2.5 times the combined bandwidth of all legitimate users), to assess how these parameters affect the accuracy of our detection algorithm.

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;
    $\tau$: threshold value;
    $\sigma_\tau$: st. deviation of values assumed by $\tau$;
    $\sigma_{max}$: max acceptable value for $\sigma_{max}$;
output: true if learning phase is not yet complete
if $\sigma_\tau < \sigma_{max}$ and $\frac{analyzed\_cos}{snap\_size} > 20$ then
  return false
else
  $\delta_m \leftarrow 0$
  for all $i \in S$ do
    $prev\_co\_count \leftarrow$
      $co\_freq[i] \cdot (analyzed\_cos - snap\_size)$
    $co\_freq[i] \leftarrow prev\_co\_count + \frac{co\_count[i]}{analyzed\_cos}$
    $\delta_m \leftarrow \delta_m + \left|\frac{co\_count[i]}{snap\_size - co\_freq[i]}\right|$
  end for
  Update $\tau$ as in Eq. (1), using Knuth [19]
  Update $\sigma_\tau$ using Knuth for variance [19]
  return true
end if
```

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
$result\_of\_detection \leftarrow$ false
$\delta_m \leftarrow 0$
for all $i \in S$do
  $\delta_m \leftarrow \delta_m + \left|\frac{co\_count[i]}{snap\_size - co\_freq[i]}\right|$
end for
if $\delta_m > \tau$then
  $result\_of\_detection \leftarrow$true
end if
for all$co \in S$do
  $co\_count[co] \leftarrow 0$
end for
return $result\_of\_detection$
```

The results of our simulations are summarized in Figs. 8–11, for XC topology with LRU, DFN with LRU, XC with LFU-DA, and DFN with LFU-DA, respectively. Due to lack of space, we only report the results for $\vartheta = 1$ and $\vartheta = 100$ – intermediate values provide very similar results. The results reported in Figs. 8 and 10 (i.e. XC topology) are all for $\gamma = 1.7$, while the results in Figs. 9 and 11 (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, Figs. 8 and 9 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 Fig. 8(b)). This is due to the fact that the attack has only minimal impact on R5.

With LFU-DA, our detection algorithm performs equally well (see Figs. 10 and 11). 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 Fig. 12. Low rate detection was one of the headline features of the work of Park et al. [27], 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 of Park et al. In fact, for each packet, their approach requires the computation of a collision-resistant hash function (SHA-1 was suggested as a viable candidate in their paper). Moreover, their algorithm computes the rank of an $n \times n$ matrix once every 2000 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 $O(n^3)$ [26], 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 [27], our approach is less expensive both per-packet (we do not compute any collision-resistant hash function) and per-detection-window.

8. Conclusion

In this paper we have shown that cache pollution attack is a realistic threat on NDN. Our experiments confirm that the attacks previously demonstrated on very small topologies extend on larger and more realistic networks with no additional effort. We have then showed that existing (proactive) countermeasures are ineffective against realistic adversaries. We have argued that detecting and limiting the attack may prove to be a better strategy. Simulations show that our lightweight detection technique provides very 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, we point out that possible strategies include rate limiting traffic corresponding to malicious interests, to caching only content that would not significantly change the detection statistics. We leave further investigation on attack reaction to future work.

Acknowledgments

Mauro Conti is supported by a EU Marie Curie Fellowship for the Project PRISM-CODE (Grant No. PCIG11-GA-2012-321980). This work has been partially supported by the TENACE PRIN Project (Grant No. 20103P34XC) funded by the Italian MIUR.

References

[1] G. Acs, M. Conti, P. Gasti, C. Ghali, G. Tsudik, Cache privacy in named-data networking, in: ICDCS 2013, 2013.

[2] A. Afanasyev, I. Moiseenko, L. Zhang, ndnSIM: NDN Simulator for NS-3. Technical Report NDN-0005, NDN, October 2012.

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

[4] M.F. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, T. Jin, Evaluating content management techniques for web proxy caches, SIGMETRICS Perform. Eval. Rev. 27 (4) (2000) 3–11.

[5] A. Balamash, M. Krunz, An overview of web caching replacement algorithms, Commun. Surveys Tutorials IEEE 6 (2) (2004) 44–56.

[6] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, in: INFOCOM, 1999.

[7] CCNx Protocol, Retr. January 2013. <http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html>.

[8] W. Chai, D. He, I. Psaras, G. Pavlou, Cache “less for more” in information-centric networks, in: Networking, 2012, pp. 27–40.

[9] ChoiceNet – Evolution Through Choice, Retr, January 2013. <https://code.renci.org/gf/project/choicenet/>.

[10] G. Dán, 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, USENIX Association, 2010, pp. 12.

[11] L. Deng, Y. Gao, Y. Chen, A. Kuzmanovic, Pollution attacks and defenses for internet caching systems, Comput. Netw. 52 (5) (2008) 935–956.

[12] J. Dilley, M. Arlitt, S. Perret, Enhancement and Validation of Squid’s Cache Replacement Policy, HP Laboratories Technical Report HPL, vol. 69, 1999.

[13] J. Dilley, M.F. Arlitt, Improving proxy cache performance: analysis of three replacement policies, IEEE Internet Comput. 3 (6) (1999) 44–50.

[14] B. Eriksson, P. Barford, J. Sommers, R. Nowak, A learning-based approach for IP geolocation, in: PAM, 2010, pp. 171–180.

[15] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, Measurements, analysis, and modeling of BitTorrent-like systems, in: Internet Measurement Conference, 2005, pp. 35–48.

[16] L. Guo, E. Tan, S. Chen, Z. Xiao, X. Zhang, Does internet media traffic really follow Zipf-like distribution? in: SIGMETRICS, 2007, pp. 359–360.

[17] L. Guo, E. Tan, S. Chen, Z. Xiao, X. Zhang, The stretched exponential distribution of internet media access patterns, in: PODC, 2008, pp. 283–294.

[18] O. Heckmann, M. Piringer, J. Schmitt, R. Steinmetz, On realistic network topologies for simulation, in: ACM SIGCOMM MoMeTools, ACM Press, 2003, pp. 28–32.

[19] D.E. Knuth, The Art of Computer Programming, Seminumerical Algorithms, third ed., vol. 2, Addison-Wesley, 1997.

[20] N. Laoutaris, H. Che, I. Stavrakakis, The LCD interconnection of LRU caches and its analysis, Perform. Eval. 63 (7) (2006) 609–634.

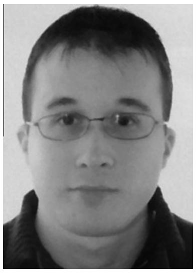
[21] N. Laoutaris, G. Smaragdakis, A. Bestavros, I. Matta, I. Stavrakakis, Distributed selfish caching, IEEE Trans. Parall. Distrib. Syst. 18 (10) (2007) 1361–1376.

[22] MobilityFirst FIA Overview, Retr. January 2013. <http://mobilityfirst.winlab.rutgers.edu>.
[23] Named Data Networking (NDN), Retr. January 2013. <http://named-data.net>.
[24] Nebula, Retr. January 2013. <http://nebula.cis.upenn.edu>.
[25] ns-3, Retr. January 2013. <http://www.nsnam.org>.
[26] M. Olschowka, A. Neumaier, A new pivoting strategy for Gaussian elimination, Linear Algebra Appl. (240) (1996) 131–151.
[27] H. Park, I. Widjaja, H. Lee, Detection of cache pollution attacks using randomness checks, in: ICC, IEEE, 2012, pp. 1096–1100.
[28] S. Podlipnig, L. Böszörmenyi, A survey of web cache replacement strategies, ACM Comput. Surveys (CSUR) 35 (4) (2003) 374–398.
[29] S. Son, V. Shmatikov, The hitchhiker's guide to DNS cache poisoning, in: SecureComm, 2010, pp. 466–483.
[30] Squid Web Caching Proxy. <http://www.squid-cache.org/>.
[31] E.W. Weisstein, Logistic Equation, Retr. on January 2013. <http://mathworld.wolfram.com/LogisticEquation.html>.
[32] XIA – eXpressive Internet Architecture, Retr. January 2013. <http://www.cs.cmu.edu/xia/>.
[33] M. Xie, I. Widjaja, H. Wang, Enhancing cache robustness for content-centric networks, in: INFOCOM, 2012.
[34] G. Zipf, In Human Behaviour and the Principle of Least-Effort, Addison-Wesley, 1949.

Mauro Conti received the Ph.D. degree from Sapienza University of Rome, Italy, in 2009. In 2008, he was a Visiting Researcher at the Center for Secure Information Systems, George Mason University, Fairfax, VA, USA. After earning his Ph.D. degree, he was a Postdoctoral Researcher at Vrije Universiteit Amsterdam, The Netherlands. From 2011, he is an Assistant Professor at the University of Padua, Italy. In 2012, he was a Visiting Assistant Professor at the University of California, Irvine, CA, USA. His main research interest is in the area of security and privacy. In this area, he has published more than 60 papers in international peer-reviewed journals and conferences. Dr. Conti was a Panelist at ACM CODASPY 2011. He served as program committee member of several conferences, and he is General Chair for SecureComm 2012 and ACM SACMAT 2013. In 2012, he has been awarded by the European Commission with a Marie Curie Fellowship.

Paolo Gasti is an Assistant Professor of Computer Science at the School of Engineering and Computing Sciences at New York Institute of Technology. His research focuses on privacy-preserving genomic computation, biometrics, secure multi-party protocols and network security. Prof. Gasti served as a member of the NDN project, which is an NSF-sponsored endeavor with the goal of designing a new Internet architecture. His work has been featured in articles by the New Scientist and MIT Technology review.

Prof. Gasti worked as a research scholar at University of California, Irvine. He received a Fulbright scholarship, under which he visited Johns Hopkins University. He received his Ph.D. in Computer Science from University of Genoa, Italy when his research pertained to the design of cryptographic schemes and network security.



Marco Teoli has received his M.S. degree in Computer Science at the University of Padua, Italy, in 2013. His research interests include network performance and security.