Enhancing Cache Robustness for Content-Centric Networking

Mengjun Xie
University of Arkansas at Little Rock
Email: mxxie@ualr.edu

Indra Widjaja
Bell Labs, Alcatel-Lucent
Email: iwidjaja@research.bell-labs.com

Haining Wang
College of William and Mary
Email: hnw@cs.wm.edu

***Abstract*—With the advent of content-centric networking (CCN) where contents can be cached on each CCN router, cache robustness will soon emerge as a serious concern for CCN deployment. Previous studies on cache pollution attacks only focus on a single cache server. The question of how caching will behave over a general caching network such as CCN under cache pollution attacks has never been answered. In this paper, we propose a novel scheme called *CacheShield* for enhancing cache robustness. CacheShield is simple, easy-to-deploy, and applicable to any popular cache replacement policy. CacheShield can effectively improve cache performance under normal circumstances, and more importantly, shield CCN routers from cache pollution attacks. Extensive simulations including trace-driven simulations demonstrate that CacheShield is effective for both CCN and today's cache servers. We also study the impact of cache pollution attacks on CCN and reveal several new observations on how different attack scenarios can affect cache hit ratios unexpectedly.**

I. Introduction

The performance and robustness of cache servers are extremely important for today's and future Internet. Numerous research efforts have been devoted to improve Web caching and media caching (e.g., [1], [2], [3]). Recently, with the appearance of frequent large-scale network attacks launched from botnets, the robustness of Internet cache servers under pollution attacks has also gained attention and been investigated [4], [5]. These studies show that the performance of a cache server (e.g., a Web proxy) can be seriously degraded when malicious requests do not follow normal Zipf-like distribution, but use uniform random access pattern.

Different from conventional DoS (Denial of Service) attacks, a cache pollution attack does not need to inject a large number of malicious requests to overload the victim server. The request flow of a cache pollution attack may appear as normal traffic; however, this low-profile malicious flow intends to violate the content locality in the server cache and can cause a flurry of cache misses, voiding the usefulness of caching at the victim server. There are primarily two types of cache pollution attacks: locality-disruption attack and false-locality attack [4]. The objective of locality-disruption attack is to request unpopular content objects to weaken content locality in a cache, while the objective of false-locality attack is to fill up a cache with unpopular content objects by repeatedly requesting those objects. A stealthy cache pollution attack can significantly degrade cache performance, posing a serious threat to a server cache system. Therefore, it is challenging but highly desirable for a large-scale distributed system to enhance cache robustness against cache pollution attacks.

Cache robustness becomes more crucial for content-centric networking (CCN). CCN, as a new network architecture that departs from the IP-based Internet, has been proposed to provide better security and much improved content delivery [6]. One critical mechanism of CCN for efficient content delivery is the use of caching. In CCN, every CCN router (node) is able to cache content objects, which essentially turns CCN into a caching network. The universal caching design makes CCN more flexible and efficient in handling a massive amount of user-generated contents, especially multimedia contents, than today's IP networking.

Our goal is to make caching in CCN more robust, especially against hard-to-detect locality-disruption pollution attacks. There are several principles that guide our exploration. First, we want the scheme to be generic, applicable to not only CCN but also other large-scale distributed systems, in particular, today's IP-based Internet. This network-architecture-independent feature is critical for incremental deployment. Second, we want the scheme to be effective, capable of working with different cache replacement policies. Third, we want the scheme to be simple, easy to be implemented and deployed. With these principles, we propose CacheShield, a generic mechanism that can effectively improve cache performance for both a single cache server and a caching network such as CCN, especially under cache pollution attacks.

CacheShield is a proactive mechanism. It aims to make caching in CCN remain robust in the first place under cache pollution attacks without performing detection *a priori*, while previous research focuses on detecting attacks and providing countermeasures *after* the attackers have been identified. Therefore, CacheShield is fully compatible with existing attack detection techniques. Moreover, CacheShield does not require coordination among different administration domains, which usually is the biggest obstacle for deployment of new network technologies. CacheShield does not require coordination among different CCN routers in the same domain either. Therefore, deployment can be targeted to the selected routers that may be more vulnerable to pollution attacks, for example, edge routers than core routers.

Intuitively, a cache pollution attack on CCN may seem to be similar to that on a single cache server. Surprisingly, our investigation reveals that this is not the case. For example, attacks on a given path may cause other CCN routers not on the path to be affected. Unlike content distribution network (CDN) where caches are strategically placed near the end users to reduce congestion in the backbone network, CCN nodes will likely be widespread during incremental deployment and gradually grow into complex topologies resembling those in IP networks. Because of complex transformation due to merging and splitting of requests for content objects at different locations, the behavior of caches in CCN cannot be easily predicted under pollution attack.

We investigate the impact of cache pollution attacks on CCN. To our best knowledge, cache pollution attack has only been studied for single isolated nodes but not at a network level where attack on one node can impact other nodes in an unpredictable manner. To understand how CCN is affected by pollution attacks and how effectively CacheShield performs over CCN, we conduct extensive simulations using three basic network scenarios including merging scenario, splitting scenario, and complex network scenario. The simulation results yield several interesting yet insightful observations.

The major contributions of this work include:

- We propose a novel mechanism, called CacheShield, to protect CCN routers against cache pollution attacks.
- We perform extensive experiments on a single router and show the robustness of CacheShield.
- We study the impact of pollution attack on CCN networks and provide several insightful observations.

In the next section, we provide basic background on CCN and survey related work. In Section III, we describe the structure and operation of CacheShield. In Section IV, we evaluate the effectiveness of CacheShield on a single router. We study cache pollution attacks on CCN network in Section V and conclude the paper in Section VI.

II. Background and Related Work

In this section, we first provide an overview of CCN that is pertinent to our investigation and then describe other work related to our paper. For convenience, we will use the terms router and node, user and client, interchangeably unless otherwise noted.

A. Overview of CCN

At a high level, CCN routers can be viewed like IP routers as they both run routing protocols and act as forwarders. One difference is that CCN performs delivery service at a higher content object (a content object, or content, consisting of two parts: content name and content data.) level than IP does at the IP-packet level. In addition, forwarding is *name*-based in CCN rather than address-based in IP.

A CCN router employs three functional components for content forwarding: (1) Forwarding Information Base (FIB), (2) Pending Interest Table (PIT), and (3) Content Store (CS). The structure of a CCN FIB is similar to an IP FIB except that CCN allows a match with multiple outgoing links (represented by "faces" in [6]). In addition, CCN performs a longest-prefix match in FIB using the name of the content instead of the IP address.

PIT is used to keep track of pending requests, termed *Interests*. When a CCN router does not have the content object being requested in its CS, the router will first record the content name of the pending request and its arriving link in PIT and then forward the request upstream toward the origin server(s) hosting the content object. Any router, which has the content object, along the path toward the server will terminate the request and reply with the content object. Thereafter, the content object travels back to the user following the chain of requests recorded in the PITs along the path. When a content object matches the request, the content object is said to consume that request and the corresponding entry in PIT is deleted. This ensures that duplicates are suppressed and loop cannot exist. When there are multiple pending requests for the same content object, the router only forwards one request (the first one) upstream.

In each CCN router, CS plays a critical role in improving network efficiency and enhancing user experience. When a user requests a content object, the router closest to the user along the path to the original source of content will terminate the request and deliver the content object to the user if it has the content object. After the delivery, a CCN router may keep the content in CS to maximize content sharing for future requests. CS may utilize any cache replacement algorithm.

In terms of how matching is functionally executed among the components, a CCN router first tries to match a given arriving request against the content name in CS. If there is a match, the router will discard the request and deliver the content to the user. Otherwise, the router will perform a match in PIT as explained above. If there is no match in PIT, the router will perform a match in FIB and forward the request to the next-hop router.

B. Related Work

Content-centric network [6] is the basis of the network architecture this paper focuses on. Other name-based routing architectures related to CCN include TRIAD [7] and DONA [8]. There have been studies on CCN or general caching networks in the context of performance measurement [9], analytical models [10], and energy impacts [11]. However, the impacts of pollution attacks on CCN have not been previously explored.

There are two existing studies on cache pollution attacks. In [4], Gao *et al.* propose and study two types of pollution attacks: locality-disruption attack and false-locality attack. They propose a *reactive* approach to detect such attacks. For false-locality attacks, the authors characterize false locality when the same IP client repeatedly requests the same unpopular files. For locality-disruption attacks, they use low hit ratio and short average life time of all cached files as metrics to detect such attack. They then identify the IP clients that request a very large number of unpopular files as the attackers. In [5], Manivel *et al.* only focus on false-locality attack. They use history of hits and misses of each IP client for detection. If the hit rate is below a certain threshold, the IP client is deemed to be an attacker; otherwise, the IP client is legitimate. While these studies only focus on a single proxy cache, we also investigate the impact at the network level in this paper and uncover new observations.

Our proposed mechanism, called CacheShield, is primarily intended to maintain cache robustness (i.e., hit ratio unaffected) under locality-disruption pollution attack. In the context of CCN, client IP addresses may no longer be visible. Contrary to the existing approach using IP-based detection, CacheShield adopts a *proactive* approach, which circumvents the need to identify the IP addresses of attackers.

The survey in [12] and [13] provide a wealth of information on many different cache replacement algorithms. We evaluate three representatives (recency-based, frequency-based, and function-based strategies) in our experiments.

III. CacheShield

Unlike packet buffers in an IP router, Content Store (CS) in a CCN router is essentially a cache. Its design purpose is to maximize the sharing of popular content objects among different users and make it as close as possible to the sources of requests. CS can employ any caching replacement algorithm. In this paper, we consider representative caching replacement algorithms including recency-based scheme such as Least-Recently Used (LRU), frequency-based scheme such as Least-Frequently Used (LFU), and function-based scheme such as Greedy Dual-Size Frequency (GDSF).

To counter stealthy cache pollution attacks to the CS of CCN, CacheShield exploits the distinction of characteristics between normal requests and malicious requests. While normal web requests follow Zipf-like distributions [1], [14], malicious requests usually follow uniform distributions due to the random access attack strategy taken by attackers for the ease of implementation and significance of damage. With random requests, attackers can render unpopular objects to be cached and popular objects to be evicted, which significantly degrades the hit ratio of normal requests. Thus, retaining popular contents in CS is critical to ensure cache robustness. Based on this principle, instead of unconditionally caching requested content objects, CacheShield favors those objects that are more frequently requested for caching. CacheShield employs a shielding function to identify relatively popular content objects and filter out unpopular ones. As unpopular content objects are more likely to be kept out of CS, the impact of pollution attack can be greatly reduced. More importantly, the performance of caching can also be improved under normal circumstances due to the denial of entrance of unpopular content objects into the cache.

A. Structure of CacheShield

CacheShield can be seen as an add-on to the CS and is compatible with any cache replacement algorithm. There are two essential components in CacheShield: a shielding function and a record of content names, as shown in Figure 1. While the shielding function is located at the front-end of a cache, content names reside in the same place as content objects.

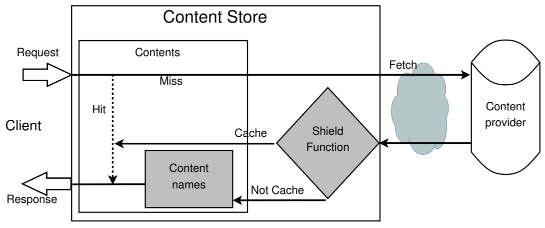

Fig. 1. CacheShield Structure.

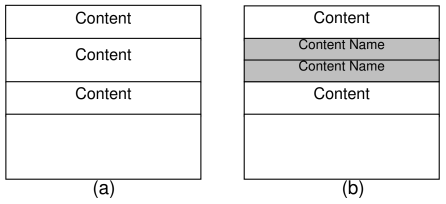

Fig. 2. CS (a) without CacheShield and (b) with CacheShield.

As can be seen from the figure, when there is a content hit upon request, the requested content object is returned immediately and the shielding function is not executed. In this case, CacheShield and normal caching operate identically. CacheShield operates differently only when there is a miss and the requested content object has to be fetched remotely (from either another CCN router or the original content provider). When there is a miss, the shielding function determines whether to cache a new content object or not. If content object C_i is not to be cached, the associated content name CN_i, if not present yet, will be cached. Content names are used to keep track of content objects that are not in the cache so as to help the shielding function make a decision. We will explain the function of content names by comparing normal caching with CacheShield in the next section.

B. Operation of CacheShield

To help understand the working mechanism of CacheShield, we use LRU as the replacement algorithm in this section. CS using LRU can be viewed as a stack, as shown in Fig. 2a. A content object that is most recently requested is placed at the top of the stack. If a content object is already in the stack prior to the request, it is simply moved to the top. Other content objects above the newly requested one are moved down by one position. If a new content object being brought into the stack causes an overflow, some existing content objects need to be evicted in order to make room for the new content object. Cache replacement algorithms deal with different methods of eviction. In LRU, existing content object(s) starting from the bottom of the stack will be evicted until the new content object can be accommodated.

Fig. 2b shows that content objects (Note that a content object includes its content name and payload data.) and content names can reside in the same storage with CacheShield. A content name is placed in a position of the stack exactly like a content object according to the caching algorithm. For example, the content name of a newly requested content object will be placed at the top of the stack if CacheShield decides not to cache the content object. Although content names consume additional storage, they are much smaller than typical content objects. When a request finds a matching content name but CacheShield decides not to cache the content object, it will record the number of attempted requests attached to the content name. CacheShield uses this information to make future decision. When a request finds a matching content name and CacheShield decides to cache the content object, the content name will be replaced by the content object. Different replacement algorithms use different policies for eviction. When a replacement algorithm evicts a given content object, it will also evict all the content names with key values less than that of the content object being evicted. Therefore, when content object x in LRU is to be evicted, any content name residing closer to the bottom than x will also be evicted.

Algorithm 1 lists the pseudo-code of CacheShield. The input is a request $Req(C)$ from a client. When a CCN router receives such a request, it will look up content object C in the cache. If C is in the cache, then C will be directly returned (lines 2–5). Otherwise, the router will forward the request and fetch the content object from either the original content provider or another router (line 6). What differentiates CacheShield from normal caching lies in the operations after the new content object is obtained. With CacheShield, the cache does not always accept and store new content, which may incur cache replacement. Instead, the cache uses the shielding function ψ to determine whether to store the content object or not (line 7). If the function decides to cache the content object, C will be stored and the internal cache data structure will be updated. At this moment, some content objects and content names will be evicted to make room for the new content object if the cache is full (lines 8–10). If the content object is not to be stored, CacheShield will either store the content name or update the number of requests associated with the content name in the cache (lines 12–16).

Algorithm 1 CacheShield Algorithm

```
1: INPUT: $Req(C)$ (Request for content object $C$)
2: if $C$ is in Cache then
3:     adjust cache internal structure
4:     return $C$ to the client
5: end if
6: forward request ... and get content object $C$
7: search content name of $C$ and use its # of requests (if
   exists) and shielding function $\psi$ to decide whether to cache
8: if $\psi$ decides to cache $C$ then
9:     store $C$ and adjust internal cache structure
10:    evict content objects (and content names) if necessary
11: else
12:    if content name of $C$ exists then
13:        update # of requests
14:    else
15:        insert content name of $C$ into cache store
16:    end if
17: end if
18: return $C$ to the client
```

C. Shielding Function

CacheShield is envisioned to deal more effectively with locality-disruption attack and intended to work with a variety of cache replacement algorithms. When a request arrives at CS and receives a hit (i.e., the requested content object is already in CS), CS uses the original caching algorithm. Otherwise, CacheShield runs a shielding function.

The primary purpose of the shielding function is to discourage unpopular content objects from being cached in CS. To thwart attackers from predicting the decision, we decide to use a probabilistic function as the shielding function. The function computes a caching probability for each requested content object. The more requests a content object receives, the higher the corresponding probability will be. In other words, the content object is more likely to be cached. One example shielding function is given by the following logistic function:

$$\psi(t) = \frac{1}{1 + e^{(p-t)/q}}, t = 1, 2, \cdots \quad (1)$$

where t denotes the t^{th} request for a given content object in CS, and p and q are the parameters of the function. With probability ψ, a new content object will be fetched into CS for possible future use. Otherwise, the content object will not be placed in CS. Instead, the associated content name and its number of requests (i.e., value t) will be recorded in CS.

As the ψ value of requests for unpopular content objects is small, unpopular content objects are less likely to be cached and cause relatively popular content objects in the cache to be evicted. Therefore, the use of the logistic function can effectively shield CS from pollution attacks in which requests are for very unpopular content objects. The parameters p and q provide each deployment site with fine control of the probability for popular requests. The setting of p and q will be further discussed in Section IV-D. Note that a shielding function is not necessarily to be the logistic function. Any function that can effectively differentiate requests for unpopular content objects from those for relatively popular content objects is a good candidate of the shielding function. Identifying such functions is subject to future research.

IV. Evaluation on a Single Node

In this section, we evaluate CacheShield as a single CCN router in isolation.

A. Methodology

We use both synthetic traffic and real traffic traces in our simulation experiments. To pursue an extensive investigation, we use synthetic traffic for the evaluation in the first part, due to limited publicly available empirical traces that contain a very large number of requests. It is necessary to have a large number of requests to deal with hit-ratio distribution as a function of content IDs. Later on, we provide verification with empirical traces.

It is well known that web request streams are highly influenced by content popularity and can be characterized by Zipf-like distributions [1][14], which is of the form $Pr\{C_k\} \propto$

$k^{-\alpha}$, where $Pr\{C_k\}$ is the probability of requesting the k^{th} most popular content object and $0 < \alpha < 1$. With synthetic traffic, legitimate users request content objects over a given content object space $\mathcal{C} = \{C_1, ..., C_K\}$ according to a Zipf-like distribution, while the attackers attempt to disrupt locality by choosing a uniform distribution over the same space. Although other distributions are certainly possible, they are less typical and more difficult to implement by attackers. Due to the space limit, our evaluation focuses on uniform distribution based attacks. We assume that requests from legitimate users or attackers follow the Independent Reference Model (IRM) [15]. Unless otherwise stated, we assume that $K = 10^6$ and cache size $B = 10^4$. We also assume that all content objects have the same size, which is set to 1, in synthetic traffic traces.

With empirical traces, legitimate users request content objects using the information from trace files of real-world traffic, which contains a list of 3-tuple items (request index, object ID, object size). The attackers request content objects according to a uniform distribution over the same content object space provided by the trace. Upon requesting a given content object, the attackers will use the object size specified in the trace.

Locality-disruption attackers pollute a cache by requesting content objects following a uniform distribution. To make the worst-case impact, the attackers may request bogus content objects belonging to a content space that differs from the space used by legitimate users, as assumed in [4]. This scheme boosts cache pollution as the attackers will never accidentally improve the hit rate of content objects requested by legitimate users. However, the scheme requires the attackers to first identify the bogus content objects that are rarely requested by legitimate users, which may not be easy in practice. Moreover, using bogus files belonging to a content space that is never accessed by legitimate users could also lead to easier detection. Therefore, we take a different assumption, in which the attackers access the same content space as legitimate users. We believe our assumption is more flexible and realistic. For example, the flat video id space of YouTube [16] makes it very easy to launch a cache pollution attack by randomly requesting YouTube videos. Obviously, the impact of attack is less powerful in our experiments. Nevertheless, our experiments equivalently reveal the effectiveness of such attacks.

Since content names (or their hashed values) are much smaller than content objects, we do not take into account the storage for content names in the evaluation. Our measurements confirm that the overhead due to content names is very small.

To study the characteristics of CacheShield in multiple dimensions, we first use LRU for the CCN Content Store. Later, we provide comparison of LRU with other cache replacement algorithms.

B. Hit Ratio

To accurately reflect the impact of pollution attack on the users that matter (i.e., legitimate users), the hit ratio in this paper refers to the hit ratio for legitimate users, instead of the hit ratio for both malicious and legitimate users that is used in [4]. The hit ratio is measured when the cache is in a steady state, i.e., the cache has been fully warmed up. Fig. 3 plots the overall content hit ratio for legitimate users vs. the ratio of attack rate to legitimate-user rate, r_a. The results are based on synthetic traffic using Zipf-like distributions (with $\alpha = 0.8$ and $\alpha = 0.9$). We set $p = 20$ and $q = 1$. As can be seen, the hit ratio for legitimate users decreases as r_a increases using normal LRU, making CCN router less efficient. Meanwhile, the efficiency of CCN router with CacheShield is almost unaffected by the attackers. We consistently observe that CacheShield not only improves the robustness of Content Store under attack, but generally also increases the caching effectiveness as indicated by its higher hit ratio even without attack.

To gain better understanding of the behavior of CacheShield compared to normal caching, we measure the hit ratio on each content object and plot the hit ratio as a function of content ID in Fig. 4. With normal caching, the hit ratios over most content objects decrease appreciably when a pollution attack (with $r_a = 1$) is present. In contrast, the hit ratios remain essentially unchanged with CacheShield as the two curves for CacheShield at the top are basically identical. The robustness of CacheShield against locality-disruption attacks lies in that it prevents many unpopular content objects requested by attackers from being cached. The figure confirms this effect for the least-popular content objects (with content IDs approximately larger than 10,000). The hit ratios for these content objects with CacheShield are lower than those with normal caching. This, in turn, enables semi-popular content objects (with IDs approximately 100-10,000) to be more likely cached, as exhibited by the higher hit ratios with CacheShield than normal caching. Since the least-popular content objects have significantly lower hit ratios than the more popular ones, the overall hit ratio with CacheShield increases as a result. Note that the very-popular content objects (with IDs 1-10) are also affected positively by CacheShield albeit in a more limited degree.

C. Effect of Cache Size

To further explore the robustness of CacheShield, we experiment with different cache sizes. Obviously, the performance of CacheShield and that of normal caching become identical in the extreme case where a cache can accommodate every content object. We explore cases where a cache can become large but is still dictated by economic or practical reasons. Fig. 5 plots the overall hit ratio versus cache size. The cases under attack ($r_a = 1$) and no attack ($r_a = 0$) are compared.

Focusing first on normal caching (the lower two curves), observe that the hit ratio under pollution attack degrades meaningfully even as the cache size is increased. This indicates that increasing cache size will not help in mitigating the cache performance when CCN router is under pollution attack. Now turning to the top two curves when CacheShield is deployed, we observe that the hit ratios with and without attack are almost identical, indicating that CacheShield maintains

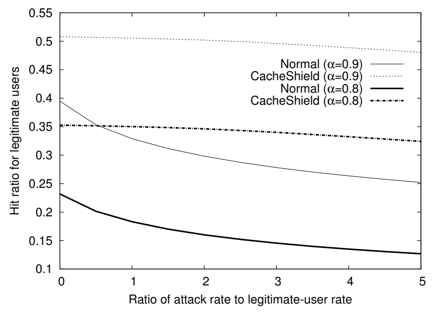

Fig. 3. Comparison of hit ratios.

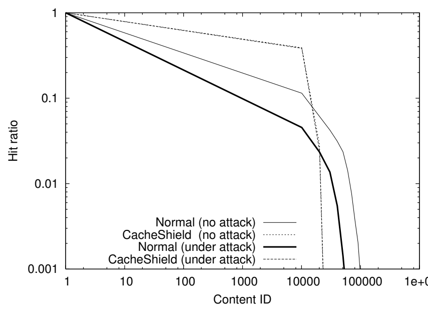

Fig. 4. Hit ratio for each content object.

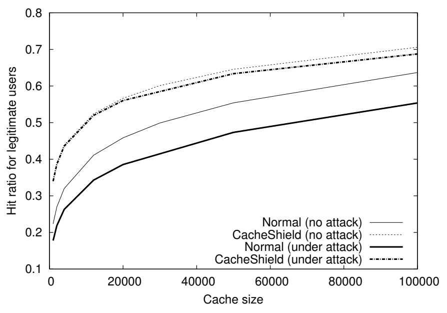

Fig. 5. Hit ratio as a function of cache size.

robustness across a wide operating range of cache sizes. Note that there is a slight difference in hit ratios when the cache is very large. This is because the least-popular content objects have higher chances of being cached in the Content Store. But these objects are vulnerable as they are also more likely to be evicted by the attackers.

D. Parameters for Logistic Function

We now examine how the parameters in the logistic function, namely p and q, affect cache performance. As the value of q increases, $\psi(t)$ increases more rapidly near the inflection point at $p = t$, and hence, CacheShield becomes more deterministic. In the limit, CacheShield will not cache a new content object if $t < p$ and will cache it with the probability close to 1 if $t > p$. At the inflection point, CacheShield will cache a new content object with the probability of 0.5.

Intuitively, we expect that the filtering of content objects by malicious users becomes more effective as p increases, because CacheShield will increasingly block less-popular content objects from being cached. This behavior without attack is depicted in Fig. 6, and the behavior under attack behaves similarly. Regardless of the value of q, the hit ratio with CacheShield converges to the same maximum attainable value as p increases. Although it seems desirable to set p to a high value to ensure the optimal hit ratio, this might slow the adaptation of CacheShield to dynamic changes in popularity. Thus, proper setting in a dynamic environment needs to be further studied.

E. Validation with Real Traces

We perform an experiment using real traces to validate the behavior of CacheShield. As opposed to synthetic traffic where requests follow IRM, real traces usually contain temporal correlation [17]. Fig 7 shows the hit ratios based on two different traces of web traffic workloads (trace 1 = worldcup [18] and trace 2 = rtp [19]). Trace 1 has a higher α value than trace 2, and its most-popular content objects have been requested several hundred-thousand times in comparison to about ten-thousand times in trace 2. As shown in the figure, CacheShield under pollution attack remains robust while the performance of normal caching deteriorates remarkably. We also find out that resetting all content sizes to a fix value does not change the qualitative results. To check whether there is any meaningful effect of temporal correlation on CacheShield, we scramble the traces by randomly permuting the request indices to break up temporal correlation that might exist in the original traces. We observe reduced hit ratios with both scrambled traces, verifying that the original traces contain temporal correlation. We also observe that the advantage of CacheShield over normal caching remains under IRM or otherwise. Lastly, we measure the byte hit ratio, which can be used as an alternative metric. As the attack rate varies, there is no qualitative distinguishable differences between the two metrics in normal caching and CacheShield.

F. Different Replacement Algorithms

To test CacheShield with different cache replacement algorithms (e.g., see [12][13]), we perform an experiment using representatives from recency-based, frequency-based, and function-based replacement algorithms as represented by LRU, LFU, and GDSF, respectively, and their variations.

For LFU, we use a variation with an aging mechanism so that the key value, K_i, of a newly requested content object, C_i, will be set to $K_i = f_i + L$, where f_i is the frequency value of C_i and L is the key value of the most recently evicted content object. The content object with the smallest K_i-value is selected for eviction. Multiple objects may be evicted to make room for a new object. The value of L is initially set to 0.

For GDSF, we use the key value that is computed as $K_i = f_i^{a_1}/s_i^{a_2} + L$, where s_i is the size of content object C_i. The values of a_1 and a_2 are used to weigh the importance of f_i and s_i, respectively. The value of L is set the same way as in LFU and content objects are also evicted the same way as in LFU.

We perform an experiment using trace-based simulation. Fig. 8 shows how the different replacement algorithms behave in terms of hit ratio under normal caching and CacheShield. Whereas the rankings under different cache replacement algorithms may not always be the same with different traces, we consistently observed the superior and robust performance of CacheShield against attacks. We also measured the byte hit ratios and and the results qualitatively reveal the same behavior.

V. Experiments on CCN

To understand the impact of cache pollution attacks on CCN and the effectiveness of CacheShield over CCN, we

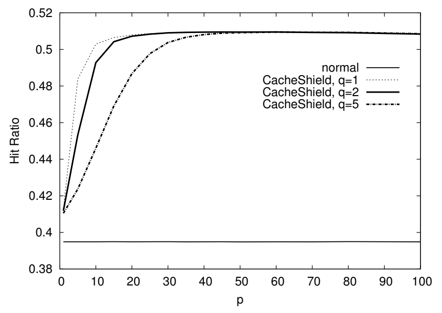

Fig. 6. Hit ratio as a function of p (with given q).

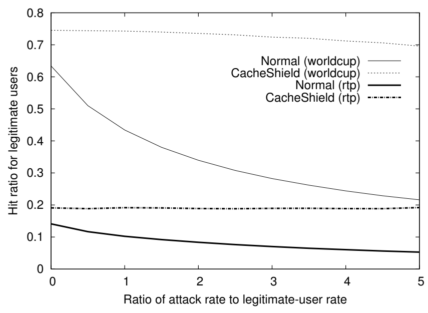

Fig. 7. Hit ratios with real traces.

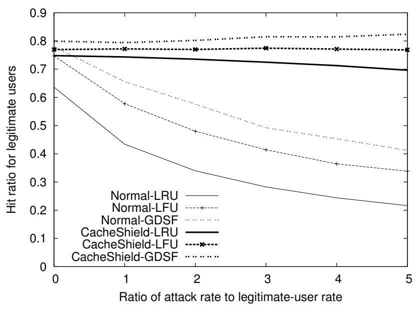

Fig. 8. Behaviors under different cache replacement algorithms.

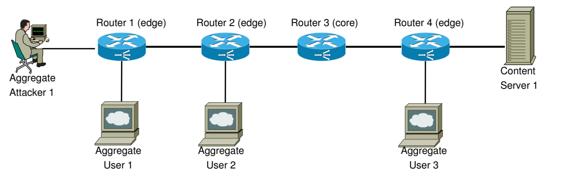

Fig. 9. Merging Scenario

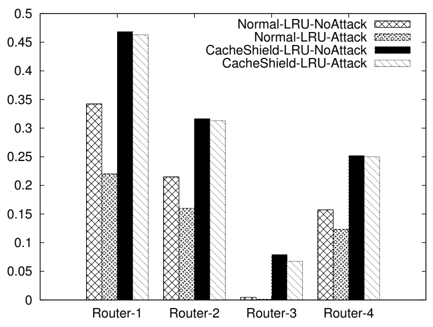

Fig. 10. Hit ratio of each router for merging scenario

conduct extensive simulations for the following three network scenarios. They are (1) merging scenario, (2) splitting scenario, and (3) complex network scenario. The merging scenario is designed to study how CCN caching behaves when normal requests are merged at intermediate routers towards the same destination. The splitting scenario focuses on studying the behavior of CCN caching when requests from the same source are split at intermediate routers. The complex network scenario consists of both merging and splitting with a more complex network topology.

As CacheShield is independent of specific cache replacement algorithm, and the performance comparison with and without CacheShield is very similar for different cache replacement algorithms, we only evaluate CacheShield with LRU in the experiments for CCN. Our simulation does not take content object size into consideration. In other words, the size of each object is fixed to one. In the simulation, all content servers (if multiple servers are used) host the same amount of content objects (100,000) and their Zipf parameter α is fixed to 0.9. The cache size of each router is also the same in each configuration and is 1% of total number of content objects.

A. Merging

We first consider a *Merging scenario* as depicted in Fig. 9. We define a CCN router as an *edge router* when there are users (clients) attached to it. A CCN router that acts only as a transit router with no users attached to it, is defined as a *core router*. As shown in the figure, there are four CCN routers connected in tandem. Three of them are edge routers and one router (Router 3) is a core router. There is also one content server attached to Router 4. The Content Store in each CCN router implements LRU cache replacement

We investigate the content hit ratios of each router with normal caching and CacheShield, first with no attack present. Then, we add the attacker that are connected to Edge Router 1, as shown in the figure. The sending rate of the attacker equals the aggregate sending rate of normal/legitimate users. It is important to note that in CCN, the attacker cannot selectively choose to pollute a particular router. This is because content objects are referred to by names, instead of addresses. Attackers can only inflict the router directly attached and other routers along the path toward the content server.

In Fig. 10, we show the hit ratios for legitimate users in each router with normal caching and CacheShield for the cases with no attack and under attack. Notice that the performance with CacheShield essentially remains unchanged when there is an attack, while the performance with normal caching under attack deteriorates. From the figure, we can state the following observation: *The hit ratio generally decreases as the request stream travels downstream toward the content server*. This is attributed to the "filter effect" [20], which eliminates the popular content objects downstream. If a router along the path is an edge router, the reduction in hit ratio is somewhat marginal since new (unfiltered) requests are injected by users attached to it. Meanwhile, we can see that the hit ratio at the core router is significantly reduced. Second, It is also interesting to observe that the hit ratio with CacheShield is higher than with normal caching at the core router. This is because CCN router behaves as a low-pass filter with a cutoff frequency of τ^{-1}, where τ is the characteristic time of a cache [2]. In other words, requests with inter-arrival time higher than τ (or frequency lower than τ^{-1}) has a high chance to have a miss and will go to the next-hop router. Since the shielding function increases the value of τ, there are higher misses with less popular (low frequency) content objects with CacheShield than with normal caching at an edge router. The higher miss rate with CacheShield translates to a higher hit ratio of less popular content objects in Router 3.

B. Splitting

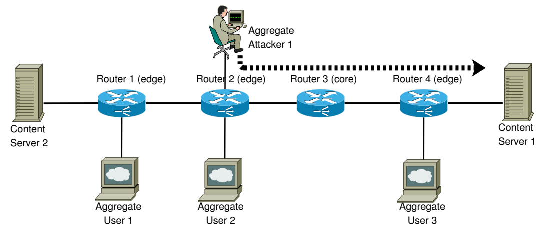

Fig. 11. Splitting Scenario

The second case we consider is the *Splitting scenario* with an example depicted in Fig. 11. In this scenario, another content server (Content Server 2) is added and attached to Router 1. The users request content objects from both of the servers, thus allowing requests to split in edge routers. We assume that users request objects from both servers equally. The attackers are attached to Router 2 in this scenario. The attackers target a victim by sending requests for content objects hosted only on Server 1. In the experiment, we assume that the attack rate is equal to the aggregate legitimate-user rate. If Server 2 is not present, it is easily deduced that the hit ratio of Router 1 will not be affected by the attack. In general, *when requests are not split in the network, the hit ratio of a router upstream from an attack path will not be affected.*

Fig. 12 displays the hit ratios of legitimate users for various cases (normal caching vs. CacheShield and with/without attack) in the splitting scenario. First, the results of Routers 2, 3, and 4 (the left two bars of each cluster) confirm our previous observation that the hit ratio of the router on an attacking path will degrade under attack. However, the performance with CacheShield for the corresponding router (the right two bars of each cluster) is much better (30% higher in terms of hit ratio) than that with normal caching and is also hardly degraded by the attack.

Next, the hit ratios of Routers 1, 2, and 4 (the first bar of each cluster) for normal LRU are similar when there is no attack. This behavior is different from that observed in the merging scenario. This is because the users have equal likelihood to send requests to either of the two content servers and the majority of their requests are cached by the local router.

As expected, we find that the hit ratios of routers, e.g., Router 1 in Fig. 12 not on the attacking path, do not degrade under attack. Surprisingly, we see that in the splitting scenario, the hit ratio with normal LRU of Router 1 actually improves under attack. This is quite distinct from those experienced by Routers 2, 3, and 4. This can be explained by examining the hit ratio of content objects on Router 1. Focusing on content objects from Server 1 that are cached in Router 1, we see in Fig. 13(a) that the difference of hit ratios of these objects with or without attack is very minor. Turning our attention to the content objects from Server 2 that are cached in Router 1, we see in Fig. 13(b) that the hit ratios for popular objects from Server 2 increase under attack. This is because the attack causes request streams from users 2 and 3 to experience higher miss ratios in Router 2 and Router 4. This results in higher request rate for popular content objects from Server 2 directed to Router 1, which evicts the least popular content objects. We add another observation: *When requests are split, the hit ratio of a router attached to an attack path may be affected but in a positive way.*

TABLE I
OVERALL HIT RATIO OF FOUR CASES

	No-Attack	A-4	A-9	A-6	A-7
Normal	23.10%	21.61%	20.87%	19.57%	19.52%
CacheShield	34.99%	34.19%	32.96%	32.52%	32.81%

C. Complex Network

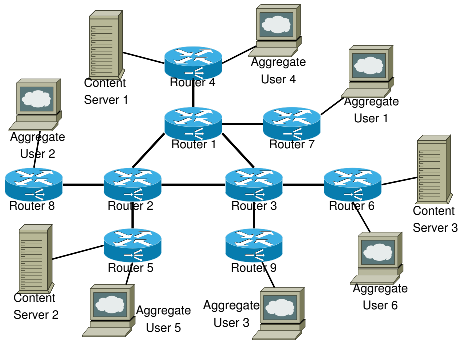

Fig. 14. Network Scenario

In this scenario, we construct a network with nine CCN routers, three as core routers and six as edge routers. Three content servers and six pools of users are attached to edge routers, as shown in Fig. 14. This scenario captures the essential components of a network and helps to better understand the overall impact on the network. Using this topology, we want to answer a key question: Given a specific aggregate attack rate, what is the best strategy for an attacker to degrade the caching performance of the network? We confine our study to the case where the attacker is is attached to an edge router. We believe this study serves as a first step toward understanding a better picture of cache pollution attack on CCN.

Due to the symmetry of network topology, Routers 4, 5, and 6 play the same role when the request rates from all users are the same without attack. This is also true for Routers 7, 8, and 9. Therefore, there are four cases to consider. Assume that the targeted server is Content Server 1. Then the attacker may be annexed to one of Routers 4, 7, 5/6, and 8/9.

We use the overall hit ratio of the six edge routers as the metric to measure the damage caused by pollution attack. This is because hit ratios of core routers are orders of magnitude lower than those of edge routers. Therefore, the impact of the attack on core routers can be safely ignored.

The overall hit ratio with no attack and the ratios of the four cases under attack are shown in Table I. The symbol 'A-4' in the table refers to the attacker attached to Router

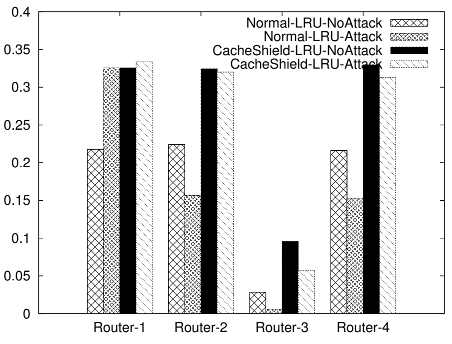

Fig. 12. Hit ratio of each router for splitting scenario

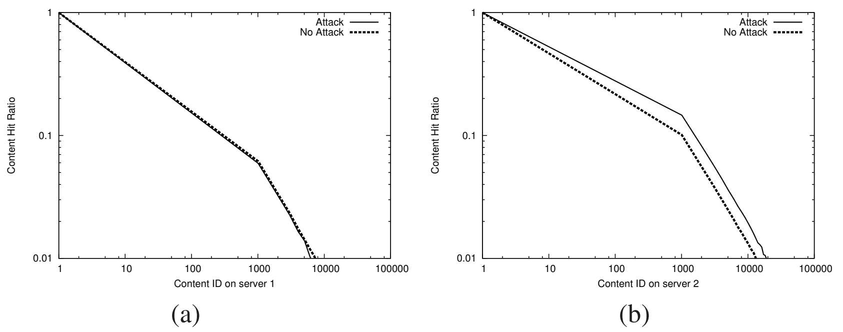

Fig. 13. Hit-ratio of content objects on Router 1.

4. With normal caching, we can see that the reduction in hit ratio is maximized when the attacker is attached to Router 7, and minimized when the attacker is attached to Router 4. The observations from the previous scenarios are helpful in analyzing the differences between these four cases. For cases A-4 and A-7, it is easy to see that only Router 4 is degraded in A-4 while both Routers 4 and 7 are degraded in A-7 from the observation in the Merging scenario. By examining the hit ratio of each individual edge router, we find that the hit ratio of Router 4 is degraded by almost the same mount (from 17.6% to 10.3%) in all four cases. This explains why the overall impact in case A-4 is minimal.

The impact on overall hit ratio in case A-9 should be similar to that in case A-7 because Routers 7 and 9 play the same role. However, careful examination of attack path will reveal the difference between A-7 and A-9. Although Router 7 in A-7 and Router 9 in A-9 are affected in a similar manner, there is one more hop involved in A-9, which is due to Router 3. As mentioned before, Router 3 can be ignored in the analysis. However, its connection with Router 6, which is linked to Content Server 3, essentially makes the difference. In case A-9, when an attack is launched on Router 9, the hit ratio of Router 6 will be improved due to the observation in the Splitting scenario. When unaffected routers are removed (Routers 2, 5, 7 and 8), case A-9 is essentially a variation of the Splitting scenario.

VI. Conclusion

In this paper, we have proposed and presented a novel mechanism, called CacheShield, that enhances cache robustness for content-centric networks. The performance of CacheShield under a variety of scenarios has been investigated in the context of CCN specifically, but applicable for a general caching network. The robustness of CacheShield has been demonstrated under pollution attack. Even without attack, CacheShield achieves higher hit ratio compared to normal caching. Furthermore, CacheShield is adaptable to different cache replacement algorithms. We have studied attack scenarios in several network topologies. In all cases, CacheShield maintains its robustness under different attack scenarios. In our study, we also discovered several revealing observations. For example, we found that the performance of a CCN router in terms of content hit ratio can actually improve if attacks are done in a certain way.

References

[1] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *INFOCOM*, 1999, pp. 126–134.

[2] H. Chee, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *INFOCOM*, 2001, pp. 1416–1424.

[3] S. Chen, B. Shen, S. Wee, and X. Zhang, "Sproxy: A caching infrastructure to support internet streaming," *IEEE Transactions on Multimedia*, vol. 9, no. 5, pp. 1062–1072, 2007.

[4] Y. Gao, L. Deng, A. Kuzmanovic, and Y. Chen, "Internet cache pollution attacks and countermeasures," in *IEEE ICNP*, 2006, pp. 54–64.

[5] V. Manivel, M. Ahamad, and H. Venkateswaran, "Attack resistant cache replacement for survivable services," in *SSRS '03: Proceedings of the 2003 ACM workshop on Survivable and self-regenerative systems*, 2003, pp. 64–71.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM CoNEXT*, 2009, pp. 1–12.

[7] D. Cheriton and M. Gritter, "Tria: A new next-generation internet architecture," in *Stanford Technical Report*, 2000.

[8] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *SIGCOMM*, 2007, pp. 181–192.

[9] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "Voice over-content-centric network," in *ACM ReArch*, 2009, pp. 1–6.

[10] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *INFOCOM*, 2010.

[11] V. H. Uichin Lee, Ivica Rimac, "Greening the internet with content-centric networking," in *Proceedings of the 1st Int'l Conf. on Energy-Efficient Computing and Networking (e-Energy)*, 2010.

[12] S. Podlipnig and L. Böszörményi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.

[13] A. Balamash and M. Krunz, "An overview of web caching replacement algorithms," *IEEE Comm. Surveys and Tutorials*, vol. 6, no. 2, pp. 44–56, 2004.

[14] A. Mahanti, C. Williamson, and D. Eager, "Traffic analysis of a web proxy caching hierarchy," *IEEE Comm. Surveys and Tutorials*, vol. 14, no. 3, pp. 416–23, 2000.

[15] J. Edward G. Coffman and P. J. Denning, *Operating System Theory*. Prentice-Hall, Inc., 1973.

[16] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang, "Reverse engineering the youtube video delivery cloud," in *Hot Topics in Multimedia Delivery (HotMD)*, 2011.

[17] R. C. Fonseca, V. Almeida, M. Crovella, and B. D. Abrahao, "On the intrinsic locality properties of web reference streams," in *INFOCOM*, 2003.

[18] "Ircache home," *http://ita.ee.lbl.gov/html/traces.html*.

[19] "Internet traffic archive," *ftp://ftp.ircache.net/Traces/DITL-2007-01-09/*.

[20] C. L. Williamson, "On filter effects in web caching hierarchies," *ACM Trans. Internet Techn.*, vol. 2, no. 1, pp. 47–77, 2002.