Security of Cached Content in NDN
Dohyung Kim, Jun Bi, Athanasios V. Vasilakos, and Ikjun Yeom

Abstract—In Named-Data Networking (NDN), content is cached in network nodes and served for future requests. This property of NDN allows attackers to inject poisoned content into the network and isolate users from valid content sources. Since a digital signature is embedded in every piece of content in NDN architecture, poisoned content is discarded if routers perform signature verification; however, if every content is verified by every router, it would be overly expensive to do. In our preliminary work, we have suggested a content verification scheme that minimizes unnecessary verification and favors already verified content in the content store, which reduces the verification overhead by as much as 90% without failing to detect every piece of poisoned content. Under this scheme, however, routers are vulnerable to verification attack, in which a large amount of unverified content is accessed to exhaust system resources. In this paper, we carefully look at the possible concerns of our preliminary work, including verification attack, and present a simple but effective solution. The proposed solution mitigates the weakness of our preliminary work and allows this paper to be deployed for real-world applications.

Index Terms—NDN, network cache, content poisoning attack, signature verification.

I. INTRODUCTION

NAMED-DATA Networking [2] has been proposed as a new networking paradigm. In NDN, users request content by specifying the content name rather than a location identifier such as an IP address. The request packet, referred to as the interest, is routed by the name and is served either by the content source or any intermediate nodes that have a copy of the content. Since routers in NDN store copies of the content that they relay, popular content is distributed over the network and users can obtain easy access to the content in terms of the content name. Obviously, the named-based access and in-network caching of NDN successfully minimize the amount of traffic over the link and provide efficient content retrieval.

However, despite these great advantages, NDN raises concerns in terms of security. Since in-network caching allows any node in the network to be a content provider, malicious users can inject poisoned content into the network. Once the poisoned content is placed on the network cache, referred to as the content store (CS), it is effectively distributed by the system itself. As a result, caches are contaminated by poisoned content, which seriously degrades the caching performance and isolates users from valid content.

To resolve this problem, NDN makes use of a digital signature that is verified either by routers or end-hosts. However, according to [3], routers cannot afford to verify all content, which can arrive at a rate above hundreds of Gbps, due to the large verification overhead. Many researchers have proposed practically feasible solutions, but these all have drawbacks/challenges. In [3], the authors presented a scheme that verifies content probabilistically. The hash values of content are used for verification instead of digital signatures. This approach significantly decreases the verification overhead, but it cannot overcome inter-packet dependency and trust management issues. In [4], a self-certifying name was discussed as a measure to mitigate content poisoning. And the authors designed a scheme that exploits users’ feedback to exclude poisoned content. However, as described in [5], this scheme is at risk of excluding valid content from the CS based upon the fabricated feedback.

In the preliminary version of this paper [1], we present an efficient scheme that drastically mitigates the verification overhead while also preventing the malicious effects of cached poisoned content. In conventional research, cache integrity has been considered essential to deal with poisoned content and all content is pro-actively checked before being cached in the CS. According to our simulation study, however, over 90% of the content is evicted from the CS without serving requests, even when the cache-hit ratio is very high. This observation implies that a large amount of computation has been unnecessarily wasting. The proposed scheme sacrifices cache integrity and perform limited verification only to the cache-hit content. This limited verification helps to avoid unnecessary verification and favors already-verified content in the CS. Meanwhile, it effectively restrains poisoned content in the CS from being spread out over the network. Segmented LRU is exploited to make already-verified content remain in the CS for longer, which improves the verification efficiency by minimizing the repeated verification of popular content. Via ns-3 simulation [6], it is shown that the verification overhead is reduced to one thirtieth (when no poison content exists in the network). Alternatively, with poisoned content, the proposed scheme improves the verification efficiency by up to 20×. However, the proposed scheme has several controversial points such as increased access latency and verification attack.

Under the scheme in [1], verification should be done when content is accessed. Hence, verification delay is included...
in the service latency, which may impair service quality. We analyze the effect of verification delay and show that the verification delay is insignificant compared with the delay gain from in-network caching. Additionally, asynchronous verification is introduced to avoid verification delay and spread the computational overhead over an extended period of time. In asynchronous verification, cached content is verified in advance if extra computational resources are available. More importantly, *verification attack* is introduced as a limitation of our scheme. In *verification attack*, attackers load a large amount of unverified content into the CS and then launch requests for this unverified content to overwhelm the router’s authentication system. Obviously, *verification attack* reduces the benefit of the proposed scheme and deteriorates the verification efficiency. To deal with *verification attack*, we present a light-weight solution to detect *verification attack* at routers and identify the vulnerable faces. The proposed solution exploits the relationship between the amount of serving content and the number of cache-hit events. If *verification attack* occurs, routers may observe more serving content than usual because much more unverified content is being accessed by the attackers’ requests. Hence, if the amount of serving content increases above a certain threshold, routers consider that they are under *verification attack* and seek the vulnerable faces. Vulnerable faces are identified simply by counting the number of forwarded pieces of serving content per face. Our simulation study shows that the proposed solution effectively detects *verification attack* as well as the vulnerable faces.

**Contributions** The main messages of this paper are summarized as follows:

- We introduce our preliminary work to mitigate content poisoning attack. According to the simulation study, the proposed scheme reduces the verification overhead by up to 95% and improves the verification efficiency by up to 20× under the LRU replacement policy.
- In terms of mathematical analyses, we show that the increased latency due to verification is much less than the delay gain from in-network caching. Hence, we demonstrate that the proposed solution rarely degrades the QoS.
- We show that our preliminary work is greatly prone to *verification attack*, which exhausts routers’ computational resources by generating massive cache-hits for unverified content in the CS.
- We suggest a solution for *verification attack*, which helps make routers aware of the attack by simply monitoring the amount of serving content and the cache-hit ratio. Furthermore, we effectively block attacking end-hosts by disabling vulnerable faces that are directly connected to them.

To the best of our knowledge, the proposed scheme is a practical solution for *content poisoning attack* due to its simplicity and alignment with the basic architecture of NDN. The rest of this paper is organized as follows. We first review the related work, including how to implement content poisoning attack, in Section II. In Section III, we introduce our solution for *content poisoning attack* and analyze its efficiency. The possible concerns of the proposed solution are discussed in Section IV. In particular, we introduce *verification attack* and its counter measure. In Section V, we present various simulation results to evaluate the proposed scheme. Section VI concludes this paper.

## II. RELATED WORK

### A. Security Attacks in NDN Caching

In NDN, security attack via the network cache is classified into two categories in terms of the purpose of the attack: *cache pollution* and *content poisoning*. *Cache pollution* aims to ruin cache locality. Unpopular content is intentionally requested by attackers, which then occupies most of the limited cache space. As a result, popular content is evicted and the benefit of in-network caching is seriously harmed. Since valid content is used in *cache pollution attack*, it is useless to check the validity of the content itself. The authors in [7] introduced CacheShield, which is a technique used to cache contents selectively according to their popularity. Since CacheShield filters out unpopular content, popular content is successfully protected in the cache. The authors in [8] pointed out that proactive solutions, like CacheShield, are effective under limited attack patterns and small-scale network topologies. They presented a reactive solution in which attacks are detected by analysing the statistics of sampling requests. A simulation study showed that their reactive solution works for more general topologies with more realistic attack scenarios.

As opposed to *cache pollution*, in this paper we focus on *content poisoning*. In *content poisoning attacks*, fabricated content is placed in the routers’ CS. The fabricated content has a valid name that can be matched with the corresponding interest, but its payload or signature is faked. When requests for the content pass through the contaminated router, they are served by poisoned content in the CS instead of being forwarded to the content source. While poisoned content is relayed back to the user, it locates in the CS of all intermediate routers. As a result, the CS is filled with useless content. Even worse, subsequent requests for poisoned content cannot reach beyond the contaminated router with a single attempt. After detecting the falsification of the received content, a user retransmits a request with the hash value of the poisoned content in the exclude field; this process incurs additional delays and hash-checking overhead at routers.

To avoid *content poisoning attacks*, the NDN architecture embeds a digital signature in every piece of content and routers perform signature verification. According to [4], however, signature verification is not mandatory at routers due to the large computational overhead that verification generates. To mitigate verification overhead, several approaches have been presented in [3]–[5], [9], and [10]. In [3] and [4], the authors discussed a self-certifying name whose last component is the hash value of the content. In terms of the hash value embedded in the interest, the valid content is identified without performing signature verification. Since the hash value of dynamically-generated content cannot be created and informed a priori, this approach
has a limit application to static content. In [5] and [9], the authors suggested the Interest-Key Binding (IKB) rule, which adds a bond between the content name and the provider’s public key. In the scheme, a user obtains the provider’s public key before issuing an interest for the content, and embeds its digest (PPKD) in the interest. Since each piece of content also carries the public key, routers match the hash value of the public key with PPKD in the PIT entry. If these do not match, the content is discarded. These approaches effectively lessens the burden of routers, but seem to create a sort of “chicken and the egg” situation since the hash value of the content or the provider’s public key must also be securely served [4]. In [9], the authors discussed schemes that allow users to obtain the provider’s public key. In the scheme, a user obtains the provider’s public key before issuing an interest for the content, and embeds its digest (PPKD) in the PIT entry. If these do not match, the content is discarded. These approaches effectively lessens the burden of routers, but seem to create a sort of “chicken and the egg” situation since the hash value of the content or the provider’s public key must also be securely served [4].

In [4], user’s feedback information is used to exclude poisoned content. Whenever user exclusion information is received, routers lower the rank of the content. By serving the content with the highest rank, routers increase the probability of serving valid content. However, user feedback has the potential of excluding valid content, as explained in [11]. In [10] and [12], feedback is secured by the signature signed by network constituents, and valid content is protected from fake feedback. However, attackers may issue a massive amount of feedback, causing routers’ computational resources to be exhausted by verification.

In [13], the authors presented a simple strategy called ‘Lossy Caching.’ In Lossy Caching, content is verified and cached with a certain probability. Routers minimize verification overhead by lowering the probability. However, the probability also affects the hit ratio as well as the recency of network caches. As the probability becomes lower, verification and caching are limited to more popular content, but caches are more likely to be filled with outdated content. Hence, it is challenging to find the optimal probability value. In addition, Lossy Caching is strongly coupled with probabilistic caching, and it is difficult to apply this scheme to different types of cache replacement policies.

Although trust management is outside the scope of this paper, it needs to be discussed for secure distribution of keys. Each piece of content contains a KeyLocator field that specifies either the name or location of the certificate. Since a certificate is a sort of content that carries a set of keys, it could be fetched in the same way that other content is retrieved. With secure encoding the identity of a key into a certificate name, the legitimacy of the signer is determined [14]. In a hierarchical trust model, a chain of keys or authorities form a hierarchy that is rooted at trust anchors, which are trusted by all verifiers. In [9], a global Key Name Server has been introduced. Similar to today’s Domain Name Server, the Key Name Server replies to the interest embedding key names with signed certificates. Here it is noted that the proposed scheme can work with most of the existing key distribution mechanisms. The proposed scheme addresses how to minimize the number of security check (verification) at routers and cut the motivation of content poisoning attacks off.

B. Implementation of Content Poisoning Attack

In this section, we discuss the implementation of content poisoning attack. As mentioned above, content poisoning attack is initiated by placing poisoned content in a router’s CS. One intuitive way to carry out this attack is to compromise routers. However, since routers cannot be easily accessed by end-users, content poisoning via router compromise is not a common threat. The risk of content poisoning attack is related to the fact that it can be implemented by end-hosts only. Fig. 1 describes the first possible scenario, which uses two end-hosts. One of the hosts is a client node that issues an interest and the other is a server node that serves the poisoned content. If the client node requests content, the interest is forwarded to the valid content source by the NDN router. At that moment, the server spills the poisoned content into the network. Since it is not mandatory that NDN routers check the incoming face of the content, the poisoned content could be inserted into the CS and forwarded back to the user by consuming the pending PIT entry. Even though the valid content arrives at the NDN router later, it is simply discarded since no pending interest is matched in the PIT.

Another route for content poisoning attack is depicted in Fig. 2. NDN routers could replicate interests and send them via multiple faces to find the closet copies of the content [15], [16]. Or, in terms of the NDN strategy layer, different parts of the content are retrieved from multiple sources in parallel for fast content retrieval as well as load balancing.
In both cases, interests could be forwarded via randomly chosen faces before reaching the attacker’s server. Then, poisoned content flows into the network and contaminates the CS of intermediate routers. The secure routing may be the first guard against these types of attacks. If careful consideration is not taken into account, however, it could limit the benefit from in-network caching in NDN.

III. EFFICIENT CONTENT VERIFICATION SCHEME

A. Basic Approach

We consider the first attack scenario where two end-hosts are used as a client and a server. This type of attack can be prevented by matching the outgoing face of the interest with the incoming face of the corresponding content. If the match fails, the incoming content is considered to be uninvited and is discarded. Here, we note that the initial version of NDN does not support this type of matching [17]. Routers only maintain the incoming face of the interest for a bread-crumb trail. Fortunately, the recent technical report for NFD [18] specifies that the interest-outgoing face is also added to the structure of PIT for implementing various forwarding strategies such as the broadcast strategy and interest retransmission. Hence, when content arrives at the router, it consumes the pending PIT entry if both the content name and faces are successfully matched. It is worth noting that checking the incoming face of content is necessary only at the edge router that is directly connected to the end-hosts. Therefore, we recommend that matching functionality should be provided by a configurable option.

In the second scenario, poisoned content arrives at routers and is stored in the CS in a legitimate way. Hence, poisoned content should either be filtered out before being stored, or properly removed from the CS. In terms of the NDN built-in security mechanism, poisoned content could be checked before being inserted into the CS. Although this approach guarantees the integrity of the CS, it incurs too much computational overhead. According to [3], a system with an Intel Core 2 Duo 2.53 GHz CPU can achieve a throughput of only 150Mbps, even when the most convenient RSA public exponent is used. Surely, performing signature verification for all arriving content is a large burden to routers. Here, we sacrifice the integrity of the CS and propose the latter approach.

Our proposed scheme is born from a simple question: why does the content that is not actually served need to be verified? It is obvious that verifying only the serving content is sufficient to prevent the malicious effects of poisoned content. A large computational overhead for by-passing content is definitely a waste of resources.

To estimate how many resources are wasted by the verification of by-passing content, we perform a simulation using ns-3 ndnSIM [19]. In the simulation, \(10^5\) pieces of content are equally distributed and served by 12 server nodes (Fig. 3). The content popularity follows the Zipf-Mandelbrot distribution function with parameters \(s\) and \(q = 0.7\) [20]. In the Zipf-Mandelbrot distribution, the probability that the \(i\)’th most popular content is requested out of \(N\) pieces of total content is denoted by

\[
P_N(i) = \frac{\Omega}{(i + q)^s} \quad (1)
\]

where

\[
\Omega = \left(\sum_{i=1}^{N} \frac{1}{(i + q)^s}\right)^{-1}
\]

The value of \(s\) varies from 0.7 and 1.3 in order to simulate various traffic patterns including web traffic and video on demand services [21]. Each client node requests 40 pieces of content per second with exponentially-distributed inter-packet delay. The bandwidth of the edge links is set to 10Mbps, while that of the core links is set to 100Mbps. The size of the CS is proportional to the link capacity. Each CS stores the content that arrives for 5 ~ 20 seconds. The simulation is performed for 1500 seconds.

Fig. 4 shows the hit rate and the proportion of serving content in the CS at R2 and R7, respectively. Here, \(r2/r7\) (in the legend) indicates a router index, and the latter part of the legend (0.7/1.0/1.3) indicates the value of \(s\) in the Zipf-distribution. In Fig. 4, it is observed that the proportion of serving content is much smaller than the value of the hit rate. Even though the hit rate is larger than 0.55 in the case of \(r2, 1.3\), the proportion of serving content in the CS is less than 0.1. These results indicate that over 90% of the content in the CS is by-passing content, without regard to the hit ratio; popular content in the CS is accessed repeatedly. If we assume that the congestion control algorithm fills up the residual link bandwidth, the absolute amount of by-passing content is high, regardless of the hit pattern.

Based on this observation, the proposed scheme implements the concept of check on cache-hit. In the proposed scheme,
all arriving content is stored in the CS without signature verification. Later, when a cache-hit occurs in the CS, that serving-content is verified before coming out to the network. This approach saves a large amount of computational resources that have previously been used for by-passing content. Nevertheless, this perfectly prevents the malicious effects of poisoned content. Under the proposed scheme, poisoned content in the CS is either simply evicted from the CS without any adverse effects or verified before influencing the network. Here, it is worth nothing that poisoned content can be diffused without the intervention of the CS by multiple pending interests. In such a case, content that has multiple pending interests can be considered as serving content and is verified before being forwarded. However, this solution may increase the average access delay of popular content. Another option is to let the poisoned content be distributed over the network. Even though this allows the poisoned content to contaminate the network caches, it would immediately be detected and eliminated by re-requests from users.

B. Enhancement

In a previous simulation, it was confirmed that popular content is repeatedly accessed in the CS. Hence, in order to avoid redundant verification at subsequent cache-hits, a flag that indicates that content is valid is set in the CS when the content is successfully verified. This flag subsists while the content stays in the CS. Sometimes, it is also observed that popular content is evicted from the CS due to the limited space, but this content is soon re-inserted into the CS. In this case, the content is verified repeatedly at each insertion (more precisely, it is verified when a cache-hit occurs after each insertion). This inefficiency is also observed in the basic NDN security architecture. To deal with this problem, we apply Segmented Least Recently Used (SLRU) to the CS.

SLRU [22] was originally designed for efficient disk systems. SLRU divides a cache into a protected segment and an unprotected segment, and the LRU policy is applied individually to each segment. If an object is accessed on the unprotected segment, it is moved to the protected segment and can stay for a longer period of time than objects in the unprotected segments. By giving preference to frequently-referenced objects, SLRU successfully improves the cache-hit ratio. If SLRU is applied to our scheme, as shown in Fig. 5, verified content is moved to the protected segment and cannot be evicted by by-passing content. If the content moves out of the protected segment, due to LRU replacement, it is first moved into the unprotected segment and then evicted from the cache. Hence, verified content has a higher probability of being re-accessed, which effectively lessens the overhead for the repeated verification of popular content.

C. Efficiency Analysis

In this section, we analyze the efficiency of the proposed scheme in terms of a metric, $\kappa$, which is defined by

$$\kappa = \frac{N_c}{N_v}$$  (2)

Here, $N_c$ is the number of verification performed for poisoned content and $N_v$ is the number of verification performed for the overall content. The value of $\kappa$ in the basic scheme, $\kappa_b$, corresponds to the ratio of the requests for poisoned content to all requests, $e$. The value of $\kappa$ in Lossy Caching, $\kappa_c$, is also equal to $e$ because Lossy Caching simply controls the amount of $N_v$ in the basic scheme by applying a certain probability, $p$.

$$\kappa_c = \frac{pN_c}{N_v} = e$$  (3)

In the proposed scheme, verification is performed only for unverified serving content. If we let $H_p$ be the hit rate for unverified content in the CS, then $N_v$ during the time interval, $\Delta t$, can be represented by $Re\Delta t + H_p(1 - e)\Delta t$, where $R$ is the request arriving rate. We note that a cache-hit occurs when popular content is re-accessed or poisoned content is reported by the re-request. $N_v$ is equal to $Re\Delta t$. Hence, the value of $\kappa$ in the proposed scheme, $\kappa_p$, is

$$\kappa_p = \frac{Re}{Re + RH_p(1 - e)} = \frac{e}{e + H_p(1 - e)}$$  (4)

In order to estimate $H_p$, we assume that the popularity of $N$ pieces of total content follows a Zipf-Mandelbrot distribution presented in (1). $H_p$ is equal to the probability that the content that has been inserted into the CS by a cache-miss generates a cache-hit for the next corresponding request. Therefore, $H_p$ is represented by

$$H_p = \sum_{\forall i} P_N(i)P_h(i)P_m(i)$$  (5)

Here, $P_h(i)$ and $P_m(i)$ are the probabilities of a cache-hit and cache-miss for the content $i$, respectively.

According to Che approximation [23], $P_h(i)$ is represented by

$$P_h(i) = 1 - e^{-P_N(i)t_C}$$  (6)

where $t_C$ is the solution of the following equation:

$$C = \sum_{\forall i}(1 - e^{-P_N(i)t_C})$$  (7)

Here, $C$ is the size of CS.
Fig. 6. The value of $H_p$ with different cache size.

Fig. 7. $H_p$ with SLRU. (a) diff. cache size ($\theta = 0.3$). (b) diff. $\theta$ (1% cache size).

Fig. 8. $\kappa_p/\kappa_b$.

Fig. 9. $\kappa_p$ with different cache size.

IV. ISSUES ON THE SCHEME

A. Access Latency

In the proposed scheme, the first access to the cached content observes a longer latency than subsequent accesses; this is the case because the first access requires a verification process. However, once the content has been verified, it can be repeatedly served without verification. In order to estimate the effect of this increased latency, we analyze the verification delay as well as the delay gain, which denotes the amount of time that will be saved due to caching.

Let $\tau_i$ and $\tau_i'$ be the RTTs of content $i$ from a user to the content source and from a user to the router, respectively. If the CS is not used, the average access delay, $D$, can be represented by

$$D = \frac{\sum_{\forall i} P_N(i) R \tau_i}{R} = \sum_{\forall i} P_N(i) \tau_i$$  \hspace{1cm} (10)$$

where $R$ is the request arriving rate.

If the CS is employed at the router, the average access delay, $D'$, becomes

$$D' = \sum_{\forall i} P_N(i) (\tau_i P_m(i) + \tau_i' P_h(i))$$

$$D' = D - G_d$$  \hspace{1cm} (11)
where the delay gain in terms of the CS, $G_d$, is given by

$$G_d = \sum_{i} \frac{P_N(i) P_h(i)}{\sigma} (\tau - \tau')$$

(12)

where $\sigma$ is a verification delay. $H_p$ is less than $\varphi$ since $P_m(i) < 1$ for all $i$ in (5). According to [3], $\sigma$ is about 80 $\mu$s when a 1500-byte piece of content is verified by a system with an Intel Core 2 Duo 2.53 GHz CPU with optimized software implementation of RSA-1024 signature verification. Hence, $L_d \ll G_d$ because $\sigma H_p \ll \frac{\varphi}{2}$. From the viewpoint of a single user, the access delay is increased by as much as $\sigma$, but this is still much less than $\tau - \tau'$; $\sigma$ is on the order of a microsecond while $\tau - \tau'$ is on the order of a millisecond. Hence, it is concluded that the effect of verification on the access latency is marginal under the proposed scheme.

**B. Asynchronous Verification**

Unlike traditional schemes, where verification is coupled with the caching operation, the proposed scheme defers verification until the content is actually served. If routers have extra computational resources under the proposed scheme, they can also verify cached content in advance. This strategy minimizes verification delay and spreads the computation over an extended period of time without degrading the other functionalities of routers.

Our asynchronous (anytime) verification starts from the top of the queue. For unverified content, a longer residence time in the CS indicates a larger inter-arrival time between two requests for the content, which may be translated into “less popular.” Hence, the proposed approach favors to popular content. Without consideration of the popularity of cached content, the proposed scheme increases the residence time of the verified content. Since the probability that the verified content is served for future request is proportional to the time for which the content stays in the CS, the proposed scheme maximizes the effectiveness of verification.

This approach looks quite similar to the convention where content is cached after being verified. In the proposed scheme, however, asynchronous verification for the content on the fly or ‘already cached but not verified’ content is optionally performed within the residual computational capacity.

**C. Verification Attack**

Although the proposed scheme effectively minimizes the verification overhead, it is still vulnerable to attacks using unverified content. Attackers may load a large amount of unverified content in the CS and then request this content to overwhelm the router’s authentication system. We refer to this type of DoS/DDoS attack verification attack. We look into the impact of verification attack via a simulation study in the topology of Fig. 10. To begin with, 10 clients request content whose popularity follows the Zipf-Mandelbrot distribution function. From 700th second to 800th second, two clients launch verification attack by issuing the same interest five times consecutively for each piece of content. Here, it is noted that in order to implement verification attack, the inter-arrival time between two identical interests should be within the range of $(\tau - \tau', \mathcal{V})$, where $\tau - \tau'$ is an RTT from the router to the content source and $\mathcal{V}$ is the dwell time of the by-passing content in the CS. If the inter-arrival time is smaller than $\tau - \tau'$, the latter interest is simply aggregated by the PIT. Alternatively, if the inter-arrival time is larger than $\mathcal{V}$, the content has already been evicted when the latter interest arrives at the router. Without any knowledge of $\tau$, $\tau'$, and $\mathcal{V}$, attackers may issue a few replicated requests in series. Even though some of these are aggregated by the PIT, it is highly probable that a cache-hit event occurs by two of them to satisfy the conditions of the inter-arrival time. Fig. 9 plots the number of verifications at R1 every 5 seconds, when the size of the CS is 1000. During the attack period, the verification overhead soars to about 400% higher than usual. If more infected nodes are used for verification attack, these will obviously exhaust the computational resources of the victim router.

To deal with verification attack, routers should be made aware of whether they are under verification attack or not. In the previous section, we demonstrated that most cache-hits are generated by a few pieces of popular content when a router is not under attack. Alternatively, during verification attack, a large amount of unverified content is intentionally injected.
to the CS to generate cache-hit events. From our observations, verification attack might be detected by using the correlation between the number of verifications that have been performed and the number of cache-hit events.

If a distribution function of the content popularity is given, the expected number of cache-hit events is expressed by $HRT$, where $R$ is the arriving rate of requests and $T$ is the monitoring interval. $H$ is the cache-hit probability of all content, which is estimated by

$$H = \sum_{i} P_N(i) P_h(i)$$

(15)

The expected number of verifications is $H_pRT$, where $H_p$ is from (5). Hence, the ratio of verifications to cache-hit events corresponds to the value of $\frac{H_p}{RT}$, which should be stable even when the popularity ranking of individual pieces of content is dynamically changing. However, if cache-hit events are manufactured under verification attack, the given content popularity distribution is distorted and the ratio of verifications to cache-hit events increases abnormally. If the ratio increases by a certain threshold, routers recognize that they are under verification attack.

If the distribution function of the content popularity itself changes (i.e., a degree of skewness), a static threshold value may not work properly. This would cause a false positive error in the detection of verification attack. Here, it is noted that verification attack increases the value of $\frac{H_p}{RT}$ much more radically than changes in the popularity distribution function. Hence, false negative errors are expected to be rare. To avoid false positive errors, the threshold value, $T$, is set as

$$T = \min(1, \frac{W(H_p)}{W(H)})$$

(16)

Here, $W$ is an exponentially-weighted moving average function and $\zeta$ is a constant larger than 1. The detection procedure is arranged in Algorithm 1.

Once verification attack is detected, routers should be able to block the attack. In terms of the proposed detection scheme, verification attack is first sensed at the router that is directly connected to the attacking node. After detecting the attack, the router moves into the identification phase to determine the vulnerable faces. In the identification phase, the victim router keeps track of the amount of verified content to be forwarded per face. Here, it is noted that a flag is used to mark ‘already-verified’ content (as described in Section III-B) in order to avoid redundant verification. Hence, routers simply count the pieces of content that has been flagged. If an excessive amount of verified content is forwarded via a specific face, that face is considered to be vulnerable. The detailed implementation is described in Algorithm 2.

If the attacking nodes are end-hosts, the router can simply disable the vulnerable faces. However, if an attack is initiated by a compromised router, disabling faces may cause legitimate requests to be detoured or served by other content sources that are farther away. Here, we observe that verification attack is similar to cache pollution attack (addressed in Section II) in the sense that legitimate but unpopular content is used to fill up the CS. The difference is that more than two identical requests must be issued to impose verification overhead in verification attack, while a single request is enough to ruin cache locality in cache pollution attack. Therefore, if the existing solutions for cache pollution, such as CacheShield [7], are applied at the victim router after the identification phase, verification attack is effectively mitigated.

---

**Algorithm 1 Detection of Verification Attack**

```c
int req /* number of incoming requests*/
int hit /* number of cache-hits */
int ver /* number of verifications */
float mTimer /* monitoring interval is set */
float T /* threshold used to detect verification attack */

****** On receiving an interest ******
req = req + 1
if cache-hit occurs then
hit = hit + 1
if first cache-hit for unverified content then
ver = ver + 1
end if
end if

****** When mTimer expires ******
calculate \(ver/\text{hit}\)
if \(ver/\text{hit} < T\) then
update \(T\) based on (16)
else
go to the Identification phase
end if
initialize req, hit, ver
reset mTimer
```

**Algorithm 2 Identification Phase**

```c
typedef struct {int cnt; char* content;} csEntry
int faces[4] /* router has 4 faces, for example */

/****** On receiving content *******/
a new csEntry, c, is created
c.cnt = 0
c.content = content
c is inserted into the CS

/****** On receiving an interest *******/
if cache-hit occurs then
**** an example case for the entry c ****
c.cnt = c.cnt + 1
face = look up the PIT
i = index of face
if c.cnt == 1 then
faces[i] = faces[i] + 1
end if
end if
```
Fig. 11. Results without poisoned content. (a) Verification Overhead. (b) Hit rate.

Fig. 12. Results with poisoned content. (a) Verification Overhead. (b) \( \kappa \).

V. EVALUATION

In this section, we evaluate the performance of the proposed scheme by using the ns-3 ndnSIM simulator. The simulation topology is shown in Fig. 10. 10^6 pieces of content are served by the server node, and their popularity follows the Zipf-Mandelbrot distribution function with the parameter values, \( s \). 10 clients request content at a rate of 40 pieces per second. Inter-request delays at each client follow the exponential distribution with \( \lambda = 0.025 \). The link bandwidth is set such that it is large enough to exclude the congestion effect. The CS is installed only at the routers (R1 and R2) and its size is varied from 100 to 2000. Except where otherwise noted, all schemes are evaluated using the simple LRU cache. Here, we note that this paper focuses on how NDN routers detect poisoned content and delete it from the CS efficiently. Hence, we simply generate poisoned content at the server with a given error probability. It is beyond the scope of this paper to address how routers isolate malicious nodes and make alternative paths to valid content sources. When clients receive poisoned content, they immediately re-request the content by issuing a new interest. Simulation is performed for 1000 seconds, and records for the last 500 seconds are analyzed.

A. Performance Without Poisoned Content

In the first simulation, we look at the results of the verification overhead (\( \frac{N_v}{N_r} \)) and hit rate at R1 when there is no poisoned content in the network. \( N_r \) is the number of arriving requests. The caching probability in Lossy Caching, \( p \), is set to 0.1. In Fig. 11, the basic scheme shows much larger verification overhead than the other schemes. When \( s = 0.7 \), most interests are served from the content source rather than from the CS, which results in a low hit rate. Nevertheless, routers verify every piece of arriving content, which makes \( \frac{N_v}{N_r} \) stay close to 1 in the basic scheme. In Lossy Caching, the caching probability dominantly affects the value of \( N_v \). With \( p = 0.1 \), \( N_v \) is reduced to 10%, and so does the verification overhead. However, the caching probability also affects the cache hit rate. When the content popularity becomes less skewed (\( s = 0.7 \)), Lossy Caching with \( p = 0.1 \) achieves a higher hit rate than the other schemes. Alternatively, when \( s = 1.0 \), the smallest hit rate is observed. Since content popularity in the Internet is not fixed but changes over time, determining the proper value of \( p \) is not trivial. The proposed scheme achieves the minimum overhead but does not influence the cache hit rate. Compared with the basic scheme, the verification overhead is reduced to up to one thirtieth. This result implies that the proposed scheme effectively minimizes the verification overhead when there is no poisoned content.

B. Performance With Poisoned Content

The proposed scheme is also evaluated with poisoned content. With a fixed CS size (1000), we look at values of the verification overhead and \( \kappa \) by varying the amount of poisoned content in the network. The caching probability in Lossy Caching is set to 0.1. Even with different amounts of poisoned content, the verification overhead does not change in either the basic scheme or Lossy Caching; this is caused by the fact that verification is performed regardless of the content’s state (see Fig. 12). Similar to the previous simulation, the largest overhead is shown in the basic scheme, and the verification overhead in Lossy Caching is determined by the caching probability. In the proposed scheme, however, the verification overhead increases in proportion to the amount of poisoned content. This is because more cache-hits occur in the CS due to re-request messages from clients that received poison content. Here, we emphasize that despite the increased overhead, the proposed scheme maintains a high value of \( \kappa \), which indicates that unnecessary verifications are effectively minimized in the proposed scheme.

C. Performance With a Different Caching Policy

In order to determine the effectiveness of the proposed scheme with other caching policies, we repeat the simulation in Section V-B with Least Frequently Used with Dynamic Aging (LFU-DA), which has been widely researched for the web proxy cache [24]. Since LFU-DA keeps a large amount of popular content in the CS, it results in a higher hit rate compared to LRU. This is especially true when the content popularity becomes more skewed (refer to the hit rate of Fig. 11(b) when the size of the CS is 1000 and Fig. 13(c)). Even with the increased hit rate, a large amount of content is evicted from the CS without serving requests. As a result, the verification overhead and \( \kappa \) are very similar to those in Fig. 12 in the basic scheme and Lossy Caching.

Since LFU-DA leads to a greater amount of serving content in the CS, the proposed scheme performs more verification in LFU-DA than in LRU, with no regard to content pollution. Here, it is noted that the number of additional cache-hits due to re-requests for poisoned content is estimated by the difference
in the hit rates between the proposed scheme and the other comparative schemes in Fig. 13(c). As the amount of poisoned content increases, the difference in the hit rates increases. Since the proportion of additional cache-hits used to detect poisoned content is relatively small when $s = 1.0$, a higher verification overhead and a lower value of $\kappa$ are observed in LFU-DA, as compared with the LRU case. However, the proposed scheme still outperforms the basic scheme and Lossy Caching in terms of overhead and efficiency.

### D. Performance With Dynamic Content Popularity

Here, we compare the proposed scheme with Lossy Caching under dynamic content popularity; this comparison is made in terms of the verification overhead, hit rate, and $\kappa$ value. We change the ranks of the content popularity every 100 seconds. If a random number is selected for $\epsilon$, the $i$-th ranked content becomes the $(i+\epsilon)$-th ranked content for the next 100 seconds. If $i + \epsilon > N$, the rank would be $i + \epsilon - N$ by modular arithmetic. ‘s0.7/s1.0’ in Fig. 14 indicates the value of $s$ in the Zipf-Mandelbrot distribution, and ‘p0.05/p0.1/p0.2’ represents the caching probability, $p$, in Lossy Caching. Similar to previous simulations, the verification overhead in Lossy Caching is managed by the value of $p$, while the overhead in the proposed scheme increases in proportion to the amount of poisoned content. However, the proposed scheme obtains a much larger value of $\kappa$, which indicates that additional verifications in the proposed scheme result in the detection of poisoned content. Here, we note that the caching probability $p$ does not influence the value of $\kappa$ in Lossy Caching; this is the case because probabilistic selection for caching is performed regardless of the content’s state. For the cache hit rate, a smaller value of $p$ results in a higher hit rate when $s = 0.7$. However, as the content popularity becomes more skewed, the benefit of probabilistic caching disappears, and Lossy Caching achieves lower hit rates than the proposed scheme.

### E. Performance Under Verification Attack

The proposed scheme is evaluated under verification attack. We perform a simulation with the same scenario that was used in Section IV-C. Fig. 15 plots the values of $H_p/H_T$ every 5 seconds with different sizes of the CS. Without verification attack, the values of $H_p/H_T$ stay around 0.6-0.7 (Fig. 15(a)) and 0.2 (Fig. 15(b)). Here, it is noted that the values in Fig. 15(b) are smaller than those in Fig. 15(a) because popular content is more frequently accessed when $s = 1.0$. After 700 seconds, the value of $H_p/H_T$ becomes distinctively larger for a period of 100 seconds. As previously explained, the traffic that is manipulated by the attacker changes the original popularity distribution of the content; this is successfully sensed by the value of $H_p/H_T$. If the value of $H_p/H_T$ increases above the threshold, as presented in (16) where the $\zeta$ value is set as 1.5, the routers move into the identification phase to find the vulnerable faces.

Fig. 16 shows how much serving content is delivered via each face every 5 seconds. Before verification attack, an average of 6~9 and 11~14 pieces of content are forwarded per face, when $s=0.7$ and $s=1.0$, respectively. After the attack is launched, however, 130 180 pieces of content are served via vulnerable faces, while the other faces still forward a similar amount of serving content. Therefore, the attacker is
effectively blocked when the vulnerable faces are disabled. Here, it is noted that operations in the identification phase are triggered after verification attack is sensed in order to minimize the overhead.

F. Performance With SLRU Enhancement

In this section, a simulation is performed to examine the effect of SLRU in the proposed scheme. We set the proportion of poisoned content to 0.1. The overall size of the CS is fixed as 1000, and the proportion of the protected segment in the CS is varied from 0 to 0.5. Fig. 17 shows the resultant hit rate and $\kappa$ values. As the size of the protected segment increases, the time during which a content stays in the unprotected segment before eviction becomes shorter. As a result, the cache hit rate decreases. However, the verification efficiency, $\kappa$, grows in proportion to the size of the protected segment, which indicates that verified content is re-used more frequently.

G. Performance With YouTube Traffic

In the last simulation, the proposed scheme is evaluated with a YouTube trace from the UMASS campus from March 11-17, 2008 [25]. The overall number of pieces of content in the trace is 158974, and we assume that all content is the same size. The temporal locality of the traffic is measured by using the stack distance [26], [27], as shown in Fig. 18. The resulting graph shows a very similar shape to the empirical stack distances obtained from other traces in [27]. In particular, about 40% of the requests show a stack distance that is less than or equal to 1000, which implies that the traffic displays highly-temporal behavior. In Fig. 19(a), it is confirmed that the verification overhead is minimized in the proposed scheme (as low as 25% of the overhead in the basic scheme) when the size of the CS is 100. As the size of the CS increases, the verification overhead increases due to the high hit ratio; however, it still remains less than that of the basic scheme. In terms of the hit rate, the proposed scheme shows a similar performance with the basic scheme (Fig. 19(b)). Fig. 20 shows the value of $\kappa$ when the proportion of poisoned content is set to 0.1. When the size of the CS is 100, the efficiency of the proposed scheme is more than five times larger than those of the comparative schemes. With a larger CS, more content is served from the CS, which increases the number of verifications in the proposed scheme. Hence, the value of $\kappa$ decreases. However, the value of $\kappa$ is still more than two times larger than those of the comparative schemes when the size of the CS is 1000.

VI. CONCLUSION

In this paper, we address the problem of content poisoning attack in NDN and present an effective solution to this problem. In terms of mathematical analyses and simulation studies, we show that the proposed scheme saves a large amount of computational resources by avoiding meaningless
verification for by-passing content and by favoring ‘already-verified content.’ Nevertheless, malicious effects from poisoned content in the CS are perfectly prevented. Additionally, we consider the potential drawbacks of the proposed scheme such as the increased access latency and verification attack. Via mathematical analyses, we show that the verification delay has a minimal effect on the service quality. We also present a simple but effective solution for verification attack.

REFERENCES


