CoMon++: Preventing Cache Pollution in NDN Efficiently and Effectively

Hani Salah* Mohammed Alfatafta* Saed SayedAhmed* Thorsten Strufe‡

*Palestine Polytechnic University, P.O. Box 198, Hebron, Palestine
hani@ppu.edu, {121035, 131008}@ppu.edu.ps

‡TU Dresden, Mommsen Str. 8, 01187 Dresden, Germany
thorsten.strufe@tu-dresden.de

***Abstract*—Defending against cache pollution attacks, highly detrimental attacks that are easy to implement in Named-Data Networking (NDN), currently suffers from the lack of coordination. Solving cache pollution attacks is a prerequisite for the deployment of NDN, which is widely considered to be the basis for the future Internet. We present CoMon++ to this end, a framework for lightweight coordination that protects from cache pollution and further attacks in NDN. Our simulation studies demonstrate that CoMon++ efficiently and effectively prevents cache pollution, remarkably outperforming a very notable state-of-the-art solution.**

***Index Terms*—NDN; Cache Pollution; Locality Disruption; False Locality**

I. Introduction

The basis of the Internet was developed in the 1960s as a network for connecting hosts, each with a unique address. Back then, the number of hosts was relatively very small, and contents were merely static pages and messages. Since then, the use of the Internet, its population, and types of exchanged contents have changed dramatically. In particular, billions of users are connected to the Internet today, mostly distributing and retrieving massive (and ever increasing) amounts of dynamic contents [1]. That is to say, the focus of networking has shifted mostly to the content itself, rather than its producer.

With this change in the way we use the Internet, it would be more efficient if its architecture was information-centric rather than host-centric one. Therefore, several architectures have been proposed in the last years in that direction [2]. Amongst them, Named-Data Networking (NDN) [3] is the most researched one. Furthermore, several academic institutions and major networking manufacturers consider NDN a potential architecture for the future Internet. Such an outstanding position of NDN motivated us to study it, aiming to improve its security before it is implemented in real world.

A key feature of NDN, which promises to significantly improve the performance of content distribution, is in-network caching. Nevertheless, it has been shown that the effectiveness (thus usefulness) of in-network caches can be degraded greatly by the so-called *cache pollution attacks*. These attacks has two main forms: (i) disrupting cache locality by unendingly issuing interest packets for unpopular contents, and (ii) creating a false locality of contents in the cache by requesting the same (small) set of unpopular contents over and over.

Although cache pollution attacks have been widely studied, the proposed mechanisms are either of limited benefit or costly. For instance, [4], [5] are effective in IP-based networks, but are not applicable for NDN. As for the mechanisms that are designed specially for NDN, some of them detect attacks but do not counter them [6]. Others are either not highly effective against realistic attacks [7], [8] or require expensive computations [9]. In addition to the aforesaid limitations and drawbacks, prior mechanisms run detection and reaction algorithms at each node (i.e. router) independently. This makes early defence against distributed attacks hard to achieve. Additionally, reacting to potential attacks at each node without coordination with the others might cause overreactions.

Our main contribution in this paper is CoMon++, a defence mechanism for cache pollution attacks. CoMon++ is an adaptation of CoMon, our framework for lightweight coordination in NDN. This choice is motivated by the success stories of CoMoN in coordinating network-wide decisions in NDN with a low overhead [10]–[12].

CoMon++ greedily selects a few nodes, located relatively close to clients, as monitoring nodes. These nodes jointly are capable to capture a network-wide view of attack-related information, like content request rates and intra-ISP hit ratios. This information is aggregated, and then used to proactively counter against cache pollution attacks in a coordinated, yet lightweight, way.

We evaluate the effectiveness and overhead of CoMon++ through extensive simulations applying challenging attacks and realistic network settings. Furthermore, we compare the effectiveness of CoMon++ to one of the most visible related works. The results show that CoMon++ is highly effective and superior. In particular, CoMon++ is able to improve the overall intra-ISP hit ratio, thus decreasing inter-ISP traffic, significantly. The results also show that the signalling overhead resulting from coordination in CoMon++ is tiny.

The remainder of this paper is organized as follows: we give an overview of NDN and cache pollution attacks in Section II. Next, we review the related work in Section III. Then, we describe CoMon++ in Section IV and evaluate it in Section V. Lastly, we conclude the paper in Section VI.

II. Background

In this section, we give an overview of NDN design and pollution attacks in Subsection II-A and Subsection II-B, respectively.

A. Named-Data Networking

The Named-Data Networking (NDN) architecture [3] was started at Xerox PARC as an attempt to tune the host-centric design of the Internet with its content-centric usage. It is one of the most researched architectures for the future Internet. Furthermore, it is supported by the National Science Foundation's Future Internet Architecture program in the US, several universities as well as some of the leading players in the networking industry.

Design concepts: The communication model in NDN is user-driven. More particularly, each content is identified by a unique hierarchical name (e.g. ***"/ieeelcn.org/papers/salah.pdf"***). End-users use *interest packets* to request contents by their names, rather than locations or host addresses. Each content is delivered inside a *data packet* on the same path through which it was requested, in the opposite way.

Each data packet contains a digital signature or a reference to it. The content producer calculates the signature over the content's name and the content itself, hence linking them to each other. The data packet contains information through which the producer's public key can be retrieved. This way, the integrity and authenticity of data packets can be verified regardless from where they are retrieved.

An essential design feature in NDN is *in-network caching*. The main point of having a cache is to store the highly popular contents and/or recently requested ones. This way, it is likely that legitimate users get their requested contents from in-network caches rather than going all the way to content producers. Having the requested contents in the cache would tremendously cut the time and cost of content retrieval.

Node model: NDN relies on a node model consisting of three data structures:

1) *Cache Store (CS):* It tentatively stores data packets passing through the node. More precisely, when a data packet is being delivered, a copy of it is cached in the CS of each node along the path between the content producer and the requesting node. In case the cache is full, a replacement algorithm (e.g. LRU or LFU) is used to replace cached data packets.
2) *Pending Interest Table (PIT):* It stores recently received requests in a two-tuple form: one records the content name and the other records the interfaces through which the content is requested. A PIT entry is removed either after satisfying the corresponding interest packet or when it times out.
3) *Forwarding Information Base (FIB):* This is a routing table-like data structure. It holds a list of outgoing interfaces against different content names or their prefixes.

Handling interest and data packets: Every time the node receives an interest packet, it looks for a matching data packet in its CS. If found, the node forwards the corresponding data packet to the same interface from which the interest packet was received.

If no matching data packet is found in the CS, the node looks for the name of the requested content in its PIT. If found but the interface from which the interest packet was received is not listed, the node appends that interface to the same PIT entry. Otherwise, the node does nothing. Doing so, NDN nodes avoid forwarding duplicate copies of identical interest packets. If no matching PIT entry is found, a new one is created (specifying the content name along with the incoming interface). The node subsequently looks for a matching entry in the FIB and forwards the packet accordingly.

As for data packets, the node looks for the content name of each received data packet in its PIT. If found, the node caches the packet, then forwards a copy of it to the interfaces through which it was requested, and finally deletes the corresponding PIT entry. If no matching PIT entry is found, the node simply discards the packet.

Security by design: NDN is robust against several types of traditional DDoS attacks [13]. In particular, prefix hijacking, bandwidth depletion, black-holing, and reflection attacks are eliminated or at least mitigated in NDN by design. This is achieved through four of the above described design features: in-network caching, PIT-based (i.e. stateful) forwarding, name-based routing and forwarding, and content-based security. Furthermore, NDN is not exposed to DNS cache poisoning because name resolution is not needed.

B. Cache Pollution Attacks

With in-network caches being the core of NDN, they are one of the primary targets for attackers. Cache pollution attacks aim at reducing the effectiveness of in-network caches, thus degrading their usefulness. These attacks are classified into two main types [5]:

1) *Locality-disruption attack:* In this type, the attacker continuously generates interest packets for new or unpopular contents, thus ruining the cache locality. This attack type might result in creating a uniform distribution of content requests. Consequently, the usefulness of caching is degraded, as popular contents get replaced very often.
2) *False-locality attack:* In this type, the attacker repeatedly requests a small set of unpopular contents. This would result in creating a false locality of contents in the cache. That is, the selected set of unpopular contents likely will take over the caches (replacing popular contents).

As the sole mission of the attackers is to pollute in-network caches, they would usually request contents at much higher rates compared to legitimate users. It is also worth to mention that locality-disruption attacks and false-locality attack are not mutually exclusive. That is to say, the attacker could perform an attack that somehow combines both attack types [5].

III. Related Work

NDN and other information-centric architectures are vulnerable to several types of attacks. These attacks have been the subject of plenty of studies in the last years (the reader is referred to [14] for a survey). In this section, we focus only on prior studies that are highly related to the subject of this paper – cache pollution attacks and defences designed against them.

Cache pollution attacks have been widely studied long before the introduction of NDN, particularly in proxy caching servers. For instance, the authors in [5] proposed reactive methods for detecting locality-disruption attacks, false-locality attacks, and an attack that combines both. For locality-disruption attacks, the authors analysed the status of caches while such an attack is ongoing. As for false-locality attacks, the authors analysed the behaviour of those who perform such attacks: since attackers repeatedly request contents from the same unpopular set of contents, it is highly likely that the same attacker will be requesting the same content more than once in a short time.

Another study was done by Deng et al. [4]. The authors studied both types of cache pollution attacks and investigated their effects on systems that use proxy caching servers. They showed that a moderate reduction of the cache hit-ratio could lead to an order of magnitude increase in the network traffic.

The proposed methods for mitigating cache pollution attacks prior to the introduction of NDN work well against attacks directed to IP-based networks. However, they are not as effective in NDN. For example, the proposed detection scheme for false-locality attacks in [5] is not applicable for NDN since it is not possible to identify the source of each request (recall that there are no host identifiers in NDN). That is, it is not possible to know when an attacker requests the same content more than once in a short period. Moreover, proxy cache servers are usually scattered in the network, while a cache in NDN is a core component in each node in the network.

In [15], Park et al. proposed an approach for detecting locality-disruption attacks by using randomness checks of the distribution of contents. The authors suggested this detection approach for Internet caching in general. Theoretically, their approach can be extended to NDN as it only depends on the contents being requested. However, it is challenging to do so practically as it will be computationally expensive when applied to many caching nodes (the authors evaluated their approach with only a single caching node in the network!). Furthermore, the authors did not study the effectiveness of their approach against false-locality attacks, and they only provided a detection technique (with no countermeasure).

Recently, there have been some efforts to study cache pollution attacks for NDN in particular. To the best of our knowledge, the first technique for increasing the robustness of caches against cache pollution attacks in NDN was introduced by Xie et al. [7]. The authors proposed a technique called CacheSheild which is a defensive technique that aims to prevent unpopular contents from being cached. The authors also showed that their approach works well against locality-disruption attacks and that it can be implemented with different cache replacement policies. One disadvantage of CacheSheild is that a large volume of statistics needs to be stored by the node, which takes up a large space.

Conti et al. [6] proposed a lightweight mechanism for detecting cache pollution attacks. Their technique works by running a learning phase at first, which defines a random sample set of the requested contents by consumers before the attack happens. After that, it monitors this set to determine if an attack is ongoing. The authors showed that their mechanism is able to detect cache pollution attacks relatively quickly with various network topologies. However, they did not provide a reaction technique. Furthermore, as the distribution of contents popularity might change from time to time in a real-world network, it is not clear how they could modify their algorithm to run the learning phase more than once.

The authors of [8] tried to benefit from the techniques used in [6] to identify a set of prefixes that attackers usually request their contents from. The motivation for their method was to reduce the storage requirements by only storing the prefixes of the contents rather than their full names. Additionally, their method works as a reactive technique as it defines a black-list for the prefixes that are identified to be requested by attackers. We argue that their method may work well only when an attacker requests contents from a small set of prefixes that consumers usually do not use. However, its effectiveness will definitely get degraded when faced by a smart attacker launching attacks specifically for this method. For example, an attacker can request contents from a large set of prefixes. Realistically, the attacker can create a list of unpopular contents that are part of a prefix that has very popular contents. By doing so, the attacker can either bypass their detection technique or degrade the effectiveness of the cache even more if that prefix gets added to the black-list.

Other contributions defined new replacement algorithms for caches designed specifically to mitigate cache pollution attacks. For example, the authors of [9] proposed a new replacement algorithm based on neural networks and fuzzy systems. One of the drawbacks of such a replacement algorithm is that it is computationally very expensive when compared to traditional replacement algorithms [16].

The aforementioned detection and defence mechanisms, in addition to the above discussed limitations and problems, work at node level only. That is to say, each node attempts to detect cache pollution attacks autonomously, without coordination with other nodes. As a consequence, early attack detection and prevention (by the nodes located close to attack sources) is likely not possible, especially for distributed attacks. This is because each node alone has only a constrained view of the features thorough which attacks can be detected (e.g. content request rates, cache usefulness, or content request patterns). Furthermore, mitigating potential attacks autonomously, without coordination among the nodes, might result in overreactions (which could harm legitimate traffic).

IV. Our Solution: CoMon++

Our goal is to defend against cache pollution (i) effectively and (ii) efficiently. On the one hand, effectiveness (as we learned from our experiences in defending against DDoS attacks [11], [12]) can be achieved if attacks are defended early (before they cause a high damage) and in a coordinated way. On the other hand, the defence mechanism is said to be efficient if it is achieving its security goals with minimum wasted effort or expense.

Obviously, the aforementioned two requirements contradict each other. In more details, the first requirement implies that each node should have up-to-date, network-wide level, view of attack-related information. This requires coordination among nodes very frequently. However, coordination at the level of the Internet, or even at the level of an ISP, causes massive volumes of network traffic. Additionally, it demands storage space and processing power which are likely unaffordable by modern routers [17].

We propose to address the aforementioned conflict by adapting CoMon, our framework for **Co**ordination with lightweight **Mon**itoring. This choice is motivated by three published success stories: we presented CoMon initially in [10] to coordinate caching-related decisions in NDN. After that, in [11] and subsequently in [12], we adapted it to defend against two types of *interest flooding attacks* (NDN-tailored, harmful, DDoS attacks). In [10] and [11], we detailed the design of CoMon and evaluated its performance and overhead thoroughly. The evaluations have shown that CoMon is capable to provide network-wide coordination, on ISP level, with a very low signalling overhead.

We call our solution CoMon++. We give an overview of its architecture and design principles in Subsection IV-A. After that, in Subsection IV-B, we detail its defence mechanism.

A. System Overview

CoMon++ works within an ISP consisting of a set V of nodes connected via a set E of edges. Fig. 1 illustrates the system architecture which contains three main types of nodes working together to defend against attacks as follows:

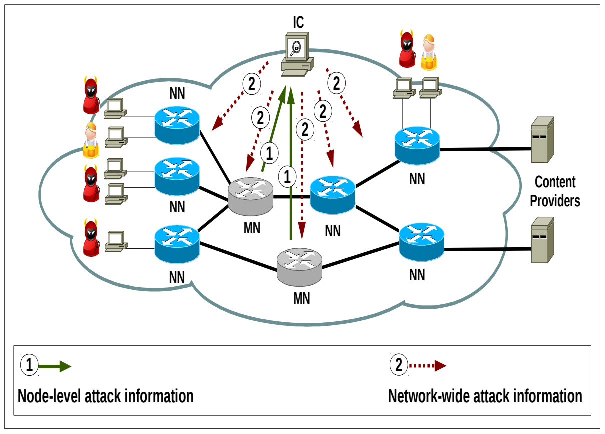

IC
NN
NN
NN
NN
NN
MN
MN
NN
Content Providers
1
2
Node-level attack information
Network-wide attack information

Fig. 1: System architecture (adapted from [12]): "IC" stands for ISP Controller, "NN" for NDN Node, and "MN" for Monitoring Node.

1) *ISP Controller (IC):* Each ISP has a centralized controller that receives statistics from a predetermined subset of nodes (hereafter, Monitoring Nodes (MNs)). The IC aggregates this statistics. It then sends ISP-wide content request information to the MNs and other nodes.
2) *NDN Nodes (NNs):* These are similar to regular NDN nodes [3] with slightly modified routing protocol and cache replacement algorithm.
3) *Monitoring Nodes (MNs):* A set $M \subset V$ (where $|M| \ll |V|$) of nodes are selected as MNs. In addition to the tasks of regular NNs, MNs continuously monitor incoming packets and their caches, and send summaries of their observations to the IC. MNs also receive ISP-wide attack-related information from the IC, and subsequently takes proper actions accordingly.

Avoiding duplication: CoMon++ avoids duplicate packet monitoring (thus inaccurate decisions) as follows: it adds a one-bit field called *Monitored* to each interest packet. The default value of this field is 0 (meaning it has not been monitored yet). Once the packet is received by an MN for the first time, that MN sets the *Monitored* field to 1. *Only* that MN updates its local attack-related statistics accordingly (to be reported to the IC).

Placement of MNs: CoMon++ selects MNs using a greedy algorithm called *PRCS* (Placement based on covered Routes and Closeness to Sources). PRCS takes as input the cardinality of M. It then selects MNs such that the number of unique routes passing through at least one MN is maximized, and simultaneously gives preference to the nodes located close to clients (so that attacks could be defended by MNs at an early stage).

More precisely, for each route r of length $l(r)$, the algorithm weights each node n located on r as follows: $w(n,r) = 1 + \frac{P(n,r)}{l(r)}$, where $P(n,r)$ is n's position on r. In particular, $P(n,r) = 0$ for the gateway node (i.e. the closest to the content producer), and incremented by one for each hop towards the client.

Maximizing coverage: Although PRCS enables for a high traffic coverage [11], it does not guarantee a complete one. Furthermore, interest packets can be filtered by caches and PITs before they encounter MNs. To achieve a complete traffic coverage, the following two techniques are implemented:

1) *Forward-Till-Be-Monitored (FTBM):* This technique aims to deal with the case when an NN receives an interest packet with $Monitored = 0$ (i.e. not monitored) and that NN can consume the packet (either has a matching data packet in the CS or a matching PIT entry). In that case, the NN sets a one-bit *Served* field (default value is 0), and then forwards the packet to the closest MN.
NNs in the way simply forwards the packet towards the designated MN. When an MN receives an interest packet with $Served = 1$, it updates attack-related statistics accordingly and drops the packet afterwards.

While adding new fields to the original packets in general is not desirable, the two fields that CoMon++ adds to interest packets (*Monitored* and *Served*) neither significantly change the structure of the packets (only one bit each) nor violate NDN's design principles (Subsection II-A). We argue that their intended benefit worth their tiny overhead.

2) *MN-Aware Routing (MAR):* This technique enforces each interest packet (thus the corresponding data packet) to pass through an MN. This is done by modifying the original routing protocol. More precisely, with MAR being enabled, each interest packets is first forwarded from the source to some (e.g. the closest) MN. After that, the packet is forwarded from the designated MN to its original destination.

NDN's name-based routing and forwarding is preserved in FTBM and MAR by treating IDs of MNs as content names. The algorithms of PRCS, FTBM, and MAR along with their evaluations are detailed in [10], [11], [18].

B. Defence Mechanism

Design requirement: The design of the mechanism is guided by one requirement: *maximizing the usefulness of in-network caches, even when they are under pollution attacks*.

We measure the usefulness of in-network caches by the *Intra-ISP Hit Ratio (IHR)* of legitimate interests, where

$$IHR = \frac{\#Interests\ satisfied\ inside\ the\ ISP}{\#Interests\ seen\ inside\ the\ ISP} \quad (1)$$

The higher the IHR of legitimate interests the lower the fraction of interests that are satisfied from the content producers, thus the lower the (costly) inter-ISP traffic. That is, the defence mechanism is said to be effective when it succeeds to either eliminate the attacks' impact on the IHR of legitimate interests or at least keep that impact very low.

Idea: Enhancing the cache robustness against pollution can be done by either proactively preventing the attacks from affecting the cache or being able to detect and then mitigate them. Our defence mechanism follows the first approach. More precisely, instead of detecting potential attacks, it attempts to identify a small set of highly popular contents, and then *glues* them in the caches of MNs as long as they remain popular.

We chose to use the caches of MNs for that purpose for two reasons: First, non-served interests are enforced to cross MNs anyway (via MAR). Second, MNs by design (via PRCS) are located close to clients. Altogether, this means that no additional hop count overhead (over that of MAR, which has been shown to be low)[1] is incurred.

[1] The hop count overheads of MAR and FTBM are small [18]: in three real ISP technologies [19], MAR and FTBM, respectively, increased the average hop count from 3.65 to 4.26 and 4.32, from 4.54 to 5.38 and 5.4, and from 3.83 to 4.16 and 3.86.

The idea of gluing a small set of contents inside the ISP is based on widely accepted published results. In particular, measurement studies (e.g. [20]–[22]) showed that popularity of contents in the Internet has a Zipf-like distribution: Zipf distribution models the probability of accessing an item at rank i out of P items as $p(i) = K/i^{\alpha}$, where

$$K = \sum_{i=1}^{P} 1/i^{\alpha}$$

and the exponent α represents the distribution skewness. With Zipf, usefulness of in-network caches is high when $\alpha \geqslant 1$, and very high when $\alpha \geqslant 2$ [10], [23]. With such values of α, majority of interests request a small set of popular contents. Our mechanism attempts to identify and glue such a set.

Workflow: The workflow can be summarized as follows:

1) The MNs continuously monitor packets passing through them. At the end of each observation period, each MN sends to the IC a list of contents it observed along with the corresponding number of requests.
2) The IC aggregates the received information.
3) The IC uses the aggregate information to identify a *white-list* of $|M| \times C$ contents, where C is the cache capacity per node. Selected contents are likely to be requested very frequently during the upcoming observation period (see the selection algorithm below).
4) The IC assigns C unique contents from the white-list to each MN. It also informs each MN about the contents assigned to other MNs.
5) The IC sends a copy of the white-list to the NNs.
6) Each MN dedicates its cache to the assigned C contents over the next observation period (without replacement).
7) When an MN receives an interest for a content cached by another MN, it *reroutes* the interest towards that MN.
8) The NNs cache contents similar to ordinary NDN nodes. The only difference is that they exclude (i.e. do not cache) the contents registered in the white-list.

Selecting the contents of the white-list: One straightforward way for selecting the contents of the white-list is to opt the most $|M| \times C$ requested contents at the end of each observation period (Algorithm 1).

While the selection strategy outlined in Algorithm 1 is simple, it is not robust against locality-disruption attacks. More precisely, under a locality-disruption attack, it is likely that the white-list will contain contents requested frequently by attackers to disrupt the locality of popular contents. Such contents should not be selected!

We propose to address the aforementioned problem by restricting the entrance to the white-list. In particular, the IC runs Algorithm 2 at the end of each observation period. In this algorithm, the IC calculates a score ($score_c$) for each requested content c (Eq. 3), using an aging-like technique.

Algorithm 1 Basic Selection Algorithm

```
1: $L \leftarrow \varnothing$                                  ▷ White-list
2: for each requested named-content $c$ do
3:     Calculate the ISP-wide request rate:
           $$RR_c = \frac{\#Interests\ requesting\ c}{observation\ period\ length\ (sec.)} \quad (2)$$
4: end for
5: Sort the contents by their request rates (highest to lowest)
6: $L \leftarrow$ top $|M| \times C$ contents
```

More precisely, it sums over x consecutive observation periods the content's RR-rank multiplied by $\frac{a}{n}$, where a is a constant and n is the index of the observation period ($n = 1$ for the current observation period, and incremented by 1 each step backward)[2]. Lastly, the IC sorts the observed contents by their scores, and registers the top $|M| \times C$ contents in the white-list.

By taking into consideration contents' ranks during former observation periods, it is likely that popular contents achieve high scores. In contrast, contents that got advance RR-ranks recently only because they have been requested frequently by locality-disruption attackers will likely achieve lower scores.

Algorithm 2 Advanced Selection Algorithm

```
1: $L \leftarrow \varnothing$                                  ▷ White-list
2: for each requested named-content $c$ do
3:     Calculate the ISP-wide request rate $RR_c$ using Eq. 2
4: end for
5: Sort the contents by their request rates (lowest to highest)
6: for each requested named-content $c$ do
7:     Calculate a score:
           $$score_c = \sum_{n=1}^{x} rank_c \times \frac{a}{n} \quad (3)$$
8: end for
9: Sort the contents by their scores (highest to lowest)
10: $L \leftarrow$ top $|M| \times C$ contents
```

Robustness against cache pollution attacks: By carefully selecting contents of the white-list and subsequently gluing them inside the ISP, the impact of locality-disruption attacks implicitly is eliminated or at least significantly mitigated.

However, since our mechanism attempts to glue popular contents in the caches of MNs, one might think that it boosts false-locality attacks (rather than countering them)! This is not true. More precisely, these attacks request a very small set κ of contents (very likely $|\kappa| \leqslant |M| \times C$). So, even though the mechanism might result in gluing *fake* popular contents in the caches of MNs, it implicitly protects the remaining much larger cache space (recall that $|V| \gg |M|$). In particular, assuming $|\kappa| \leqslant |M| \times C$, at least $(|V| - |M|) \times C$ slots remain unaffected, i.e. can be used to cache *real* popular contents.

[2] To reduce memory requirements, we use a small value of x ($x = 3$).

V. Evaluation

In this section, we present the simulation study that we performed to evaluate the effectiveness of our solution as well as its signalling overhead. In Subsection V-A, we introduce the evaluation parameters. Next, in Subsection V-B, we describe the evaluation setup. Lastly, we present and discuss the results in Subsection V-C.

A. Evaluation Parameters

We use the following two metrics in our evaluation:

1) *Intra-ISP Hit Ratio (IHR) of legitimate interests:* This metric, as defined and discussed in Section IV-B (Eq. 1), is used to measure the impact of cache pollution attacks as well as the effectiveness of our defence mechanism and CacheSheild. The attack is considered effective if low IHR of legitimate interests is achieved. The opposite is true for defence mechanisms.
2) *Signalling overhead:* The value of the signalling overhead is calculated by normalizing the total size of control messages exchanged between the nodes and the IC (messages marked "1" and "2" in Fig. 1) by the total size of regular data packets.

B. Evaluation Setup

We implemented CoMon++ and the attacks in ndnSIM [24], a widely used simulator in the NDN community. For comparison purposes, we also implemented CacheShield [7], one of the most notable related works.[3]

We experimented with three real ISP network topologies measured by the Rocketfuel project [19]. The results of the three topologies are very similar and lead to equal conclusions. Due to space constraints, we will discuss the results of one network topology only: the AS-3967 topology (Fig. 2). This network consists of 79 nodes, connected via 147 edges. It has an average degree of 3.72 and a diameter of 10.

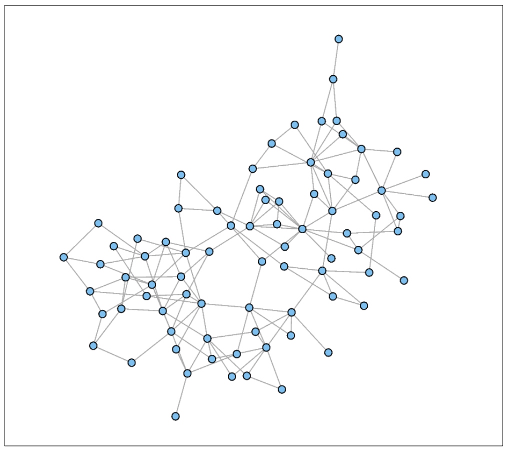


Fig. 2: AS-3967 network topology

[3] The source codes along with the configuration files and network topologies can be found here: https://github.com/fataftamohammad/CoMon-/

At the beginning of each experiment, the simulator randomly selected 70% of the nodes as access nodes (through which clients connect to the ISP) and three of the rest as gateway nodes (through which content providers, outside the ISP, are accessed). The simulator also randomly selected 25% of the clients as attackers (unless otherwise mentioned). Only top $\lceil 10\% \rceil$ PRCS-ranked nodes acted as MNs.

Each legitimate client issued interest packets at a relatively slow rate: 0.05 interest packets per second (ipps). Each attacker issued interest packets nine times faster, i.e. at a rate of 0.45 ipps. We also experimented with faster attack rates. However, the impact on the cache effectiveness was marginal.

We fed the servers with $30,000$ contents (i.e. catalogue size $S = 30,000$). Attackers in the locality-disruption attack and in the false-locality attack sent requests for $0.05 \times S$ contents and $0.0005 \times S$ contents, respectively. These values were selected carefully such that the attacks (when no defence is used) remarkably degrade the usefulness of caches.

In order to increase the competition among the contents on the available cache space, and simultaneously increase the effectiveness of the attacks, we used small caches with respect to the catalogue size. In particular, we used a uniform cache size of $C = 3$ contents, i.e. $C = 0.0001 \times S$. The impact of attacks with larger cache sizes (we also experimented with $C = \{9, 45, 145\}$) was low, thus is not discussed below.

Following the aforementioned measurement results (see Section IV-B), we modelled the popularity of contents by Zipf distribution. Since there is no agreement in the literature on the skewness parameter (α), we performed simulations with varying values of $\alpha = \{0.9, 1.5, 2.5\}$.

To avoid accidental results, each experiment was repeated 20 times. In addition, each experiment lasted for a long time, particularly for $1,000,000$ simulation seconds. The attacks started at second $260,000$ and stopped at $860,000$. The length of the observation period was set to $20,000$ seconds. These time durations were long enough to evaluate both the impact of the attacks and the effectiveness of the defence mechanism.

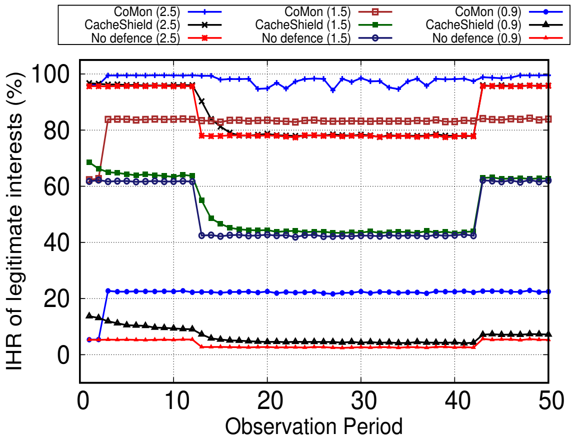
We used a small data packet size of 1100 bytes (smaller than the Ethernet's MTU) to avoid biasing the signalling overhead.

C. Results

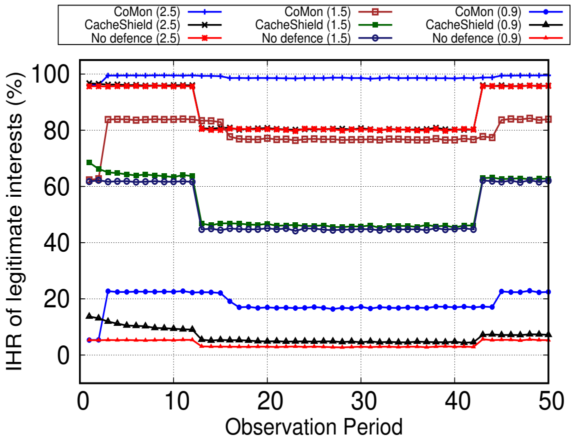
Effectiveness of the attacks and defence mechanisms: We compare the IHR of legitimate interest packets for three systems: (i) a system with no defence mechanism, (ii) a system enabling CacheShield, and (iii) a system enabling CoMon++. We report the IHR of legitimate interests before the start of the attack, during the attack, and after disabling the attack. We do so separately for the locality-disruption attack and for the false-locality attack, once with $\alpha = 0.9$, once with $\alpha = 1.5$, and once with $\alpha = 2.5$. Altogether, we have 18 different cases.

To not bias to CoMon++, we show the results of the run in which we obtained the minimum IHR of legitimate interest packets with CoMon++, and compare them with results of CacheShield and the No-Defence scenario when each of them achieved the maximum IHR of legitimate interest packets. It worth also to mention that the results of repeated runs were nearly identical. Therefore, we argue that the results are not only unbiased, but also are representative for all the runs.

We depict the results in Fig. 3: under the locality-disruption attack (Fig. 3a) and under the false-locality attack (Fig. 3b).



(a) Under locality-disruption attack



(b) Under false-locality attack

Fig. 3: IHR of legitimate interest packets

The results can be summarized as follows:

1) Both attacks had a remarkable negative impact on IHR, thus on the usefulness of in-network caching. In particular, in the case of $\alpha = 0.9$, as to be expected, the usefulness of caching was low (IHR of legitimate interests was about 5.5%), and it was decreased by both attacks to about 2.5%. With $\alpha = 1.5$, the locality-disruption attack and the false-locality attack lowered the IHR from about 62% to 42% and 45%, respectively. In the case of $\alpha = 2.5$, the IHR was decreased from about 96% to about 78% by the locality-disruption attack and to about 80% by the false-locality attack.
2) CacheShield had almost no impact on the two attacks in the case of $\alpha = 2.5$, and it had a marginal positive impact in the cases of $\alpha = \{0.9, 1.5\}$. That is, CacheShield was ineffective against such realistic cache pollution attacks.

3) CoMon++ had a notable positive impact on the IHR results, thus on the cache effectiveness and usefulness. This is true not only when the network is under attacks, but also in the absence of attacks. In particular, with $\alpha = 0.9$, the IHR of legitimate interests was improved by CoMon++ in the absence of attacks from about 5.5% to about 22%. In the presence of locality-disruption, the value remained around 22%. Under false-locality, the achieved IHR was about 17%.
With $\alpha = 1.5$, CoMon++ increased the IHR of legitimate interests in the absence of attacks from about 62% to more than 83%. During attacks, it increased the IHR of legitimate interests under the locality-disruption attack and the false-locality attack from about 42% to above 82% and from about 45% to above 77%, respectively.
With $\alpha = 2.5$, CoMon++ raised the IHR of legitimate interests when no attack exists from about 96% to more than 99%. As for its effectiveness against cache pollution attacks, it was able to raise the IHR of legitimate interests under the locality-disruption attack from about 78% to 94% and more. The obtained improvement under false-locality attack was even higher: raised from about 80% to more than 98%.

Results under highly aggressive attacks: The results above showed that the system was simulated under effective attacks, and also that CoMon++ is effective against them. We performed further experiments to test the effectiveness of CoMon++ against much more aggressive attacks. In particular, we increased the ratio of malicious clients from 25% to 90%.

We plot the results in Fig. 4. They can be summarized as follows: [4]

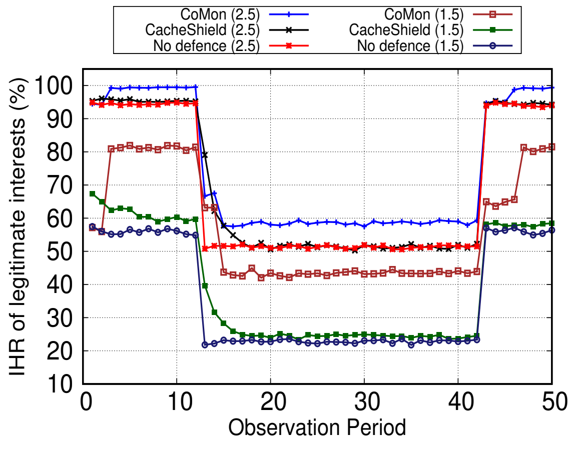
1) As to be expected, the new attacks had larger negative impacts on the IHR of legitimate interests. In particular, in the case of $\alpha = 1.5$, both the locality-disruption attack and the false-locality attack decreased the IHR from about 57% to about 23%. With $\alpha = 2.5$, the two attacks lowered the value from about 95% to roughly above 50%.

2) Similar to the results of less aggressive attacks, CacheShield had almost no impact on the two new attacks except a very small positive impact in the case of $\alpha = 1.5$.

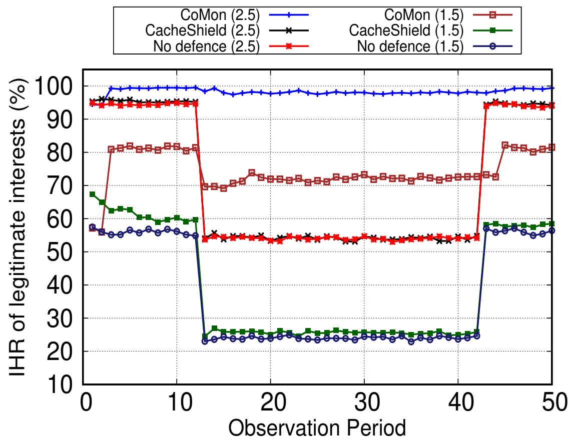
3) CoMon++ improved the usefulness of caching under such aggressive attacks remarkably: with $\alpha = 1.5$, the IHR of legitimate interests was raised by about 20% and 46% under locality-disruption and false-locality, respectively. In the case of $\alpha = 2.5$, the improvements under the two attacks were about 8% and 45%.

Over all, the results above demonstrate the effectiveness of CoMon++. In particular, the impact of cache pollution attacks (even highly aggressive ones) can be remarkably mitigated when CoMon++ is enabled. Furthermore, CoMon++ (by gluing the most popular contents inside the network) can enhance the effectiveness of in-network caches even in attack-free scenarios.

[4] We omit the results of $\alpha = 0.9$ because the corresponding attack-free cache usefulness is very low, as discussed above.



(a) Under highly aggressive locality-disruption attack



(b) Under highly aggressive false-locality attack

Fig. 4: IHR of legitimate interests under highly aggressive attacks

Signalling overhead: Lastly, we look at the amount of signalling overhead caused by CoMon++ for coordination (i.e. for exchanging control messages between the IC and the nodes at the end of observation periods). The results of all experiments show that the signalling overhead is tiny. In numbers, in all the experiments that we performed, the signalling overhead was always below 0.003%.

The signalling overhead was similar with the two attack types. This is to be expected because the exchanged information is the same.

VI. Summary and Future Work

The main contribution that we presented in this paper is CoMon++, a coordinated solution for cache pollution attacks in NDN. The main feature of CoMon++, which distinguishes it from state-of-the-art solutions, is relying on a network-wide (rather than node-level) view of attack-related information. CoMon++ is designed to capture such a view by a small fraction of the network nodes.

We evaluated CoMon++ through an intensive simulative study. The results showed that CoMon++ is highly effective and remarkably outperforms a notable state-of-the-art solution. The results also showed that the signalling overhead incurred by CoMon++ for coordination is very low.

The current design of CoMon++ can be improved in several ways. For instance, fault tolerance and load balancing can be enhanced by redesigning the ISP controller in a distributed way. Furthermore, multiple ISPs could cooperate in collecting attack-related information and subsequently in defending against potential attacks.

Acknowledgement

This work was supported by Palestine Polytechnic University (through the Deanship of Scientific Research) and the Palestinian Higher Council for Innovation and Excellence.

References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2014–2019," *CISCO White paper*, 2015.

[2] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Communication Magazine*, 2013.

[3] V. Jacobson *et al.*, "Networking named content," in *CoNEXT*, 2009.

[4] L. Deng *et al.*, "Pollution attacks and defenses for internet caching systems," *Computer Networks*, 2008.

[5] Y. Gao *et al.*, "Internet cache pollution attacks and countermeasures," in *IEEE ICNP*, 2006.

[6] M. Conti *et al.*, "A lightweight mechanism for detection of cache pollution attacks in named data networking," *Computer Networks*, 2013.

[7] M. Xie *et al.*, "Enhancing cache robustness for content-centric networking," in *IEEE INFOCOM*, 2012.

[8] T. Kamimoto *et al.*, "Cache protection method based on prefix hierarchy for content-oriented network," in *IEEE CCNC*, 2016.

[9] A. Karami and M. Guerrero-Zapata, "An anfis-based cache replacement method for mitigating cache pollution attacks in named data networking," *Computer Networks*, 2015.

[10] H. Salah and T. Strufe, "CoMon: An Architecture for Coordinated Caching and Cache-Aware Routing in CCN," in *IEEE CCNC*, 2015.

[11] H. Salah, J. Wulfheide, and T. Strufe, "Coordination supports security: A new defence mechanism against interest flooding in NDN," in *IEEE LCN*, 2015.

[12] H. Salah and T. Strufe, "Evaluating and mitigating a collusive version of the interest flooding attack in ndn," in *IEEE ISCC*, 2016.

[13] P. Gasti *et al.*, "DoS and DDoS in Named Data Networking," in *IEEE ICCCN*, 2013.

[14] E. G. AbdAllah, H. S. Hassanein, and M. Zulkernine, "A Survey of Security Attacks in Information-Centric Networking," *IEEE Communications Surveys & Tutorials*, 2015.

[15] H. Park *et al.*, "Detection of cache pollution attacks using randomness checks," in *IEEE ICC*, 2012.

[16] Z. Xu *et al.*, "Elda: Towards efficient and lightweight detection of cache pollution attacks in ndn," in *IEEE LCN*, 2015.

[17] D. Perino and M. Varvello, "A reality check for content centric networking," in *ACM SIGCOMM workshop on Information-centric networking*, 2011.

[18] H. Salah, "Measuring, understanding, and improving content distribution technologies," Ph.D. dissertation, Technische Universität Darmstadt, 2016.

[19] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *SIGCOMM*, 2002.

[20] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *ACM IMC*, 2007.

[21] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Transactions on Networking*, 2008.

[22] M. Busari and C. Williamson, "ProWGen: a synthetic workload generation tool for simulation evaluation of web proxy caches," *Elsevier Computer Networks*, 2002.

[23] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," Telecom ParisTech, Tech. Rep., 2011.

[24] A. Afanasyev *et al.*, "ndnSIM: NDN simulator for NS-3," *University of California, Los Angeles, Tech. Rep*, 2012.