Analysis, Detection and Mitigation of Cache Pollution Attack and Interest Flooding Attack in Named Data Networking

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology
in
Computer Science & Engineering

by

Group CS03
Chinmay Pani, Himanshu Bansla, Debasmita Das, Daniel Wintermeyer, Laxmi Kant Yadav, Abhishek

to the

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD,PRAYAGRAJ
May, 2019

UNDERTAKING

I declare that the work presented in this report titled "*Analysis, Detection and Mitigation of Cache Pollution Attack and Interest Flooding Attack in Named Data Networking*", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is my original work. I have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

May, 2019
Allahabad

(Chinmay Pani)

(Himanshu Bansla)

(Debasmita Das)

(Daniel Wintermeyer)

(Laxmi Kant Yadav)

(Abhishek)

CERTIFICATE

Certified that the work contained in the report titled "*Analysis, Detection and Mitigation of Cache Pollution Attack and Interest Flooding Attack in Named Data Networking*", by *Chinmay Pani, Himanshu Bansla, Debasmita Das, Daniel Wintermeyer, Laxmi Kant Yadav, Abhishek*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Shashank Srivastava,
Assistant Professor)
Computer Science and Engineering Dept.
M.N.N.I.T, Allahabad

May, 2019

Preface

The basis of our thesis is to explore the potential of Named Data Networking and its efficiency in delivering data to the consumers. It is written to provide a new approach for detection of Cache Pollution Attack using a new metric based upon the number of unique nodes or users requesting for data with a particular prefix and hence, rating each node with this metric which changes dynamically over time in the network. It also shows the effect of advanced Interest Flooding Attacks and the performance of currently available countermeasures for IFA against them.

The idea for the project was conceived under the guidance of our mentor Dr.Shashank Srivastava. Our project work was decided together by our group and our mentor. We have worked upon the Analysis, Detection and Mitigation of Cache Pollution Attack and Interest Flooding Attack in Named Data Networking.

The result of the thesis can be implemented by someone who has basic idea of how routing algorithms are implemented and how metrics are used for determining the popularity of data and satisfaction of interests.

Acknowledgements

We would like to express our sincere gratitude to **Dr. Shashank Srivastava** for providing us the golden opportunity to do this interesting project on the topic ***Analysis, Detection and Mitigation of Cache Pollution Attack and Interest Flooding Attack in Named Data Networking*** which also helped us to learn and explore more about networking, networking devices, and routing protocols. We would also like to thank all the Ph.D. Scholars working under Dr. Shashank Srivastava for their proper guidance and encouragement in carrying out this project in the limited time frame.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The Internet has changed a lot for the past few decades. How we use it as well as why we use it, has changed significantly. Originally it was intended as a point-to-point system hence, emails, chatting were very common, but as things have changed, services like Netflix, YouTube, Facebook have become popular, showing that multicast uses have taken precedence. With the previous IP architecture, the point-to-point system made sense, but as things have changed. Issues are arising as a result of these changes. NDN seeks to solve some of these issues. Our Project seeks to explore the potential of NDN and find a way to detect and mitigate the problem it faces during the course of its evolution.

1.1 Motivation

Named Data Networking is a major paradigm shift in network architecture. It has the potential to replace IP (Internet Protocol) in the hourglass model. Today's Network is based on a protocol called IP which works by naming endpoints, while the usage is mainly focused on the content of the data rather than its source. Point-to-point network architecture has many limitations when it comes to large-scale content/data distribution like efficiency and security. Another major challenge faced is Privacy where one can manipulate packets and make the user vulnerable to certain attacks by extracting information like its IP address, MAC Address, etc. NDN seeks to solve some of these problems faced by the current Internet, that is content distribution, network congestion, routing of data, etc.

Our main motivation behind the project is to narrow down the architectural mismatch between the current Internet's architecture and its usage.

1.2 NDN Architecture

NDN is a very fast developing architecture that is designed to remove the limitations of the currently working IP based architecture. NDN retains the same **hourglass architecture** which allows upper and lower layers to function and develop independently.

NDN identifies the data by naming them instead of identifying the end-systems, thus emphasizing on the quality of content instead of its source. It classifies the data into hierarchies based on their name prefixes. The user requesting data, i.e., The Consumer, sends out an interest packet which contains the name of the data required. Once the interest packet has been created, it follows a specific path through certain sections namely The Content Store (CS), The Pending Interest Table (PIT) and The Forwarding Information Base (FIB) in a previously configured router. These sections are addressed in a hierarchical order. The Content Store is the first stop for the interest packet which contains the interest packets that have previously been fulfilled or requested for. In the CS, we can find commonly visited websites which were cached by previously fulfilled Interest Packets. If the data is not found in the content store, then the interest packet goes through the Pending Interest Table. The PIT lists those interest packets which have no data packets returned or stored in the CS. Then following the PIT, it goes to the Forwarding Information Base and follows a routing protocol before sending out the packet to the web.

Once the data is found by matching the name prefix, a data packet is generated by the producer and sent back to the consumer retracing the same path that of the interest packet.

1.2.1 Naming Convention of Data

NDN uses hierarchical naming for data packets. For example, a video lecture of NDN on youtube may be named as */youtube/videos/NDN*. The next lecture of the same series may be named as */youtube/videos/NDN/2* and so on. This type of hierarchy allows convenient aggregation of content and efficient interest forwarding based on matching prefixes, thus speeding up the retrieval of content in the network.

1.2.2 Types of Packet

Two types of Packets is used in NDN:-

1. **Interest Packet**
2. **Data Packet**

▮ *Interest Packet*

Whenever a consumer requests some data, it sends a request packet containing the name of the data it requires. This request packet is known as **Interest Packet** in NDN terminology. The interest packet has three parts namely *Content Name, Selector, and Nonce.*

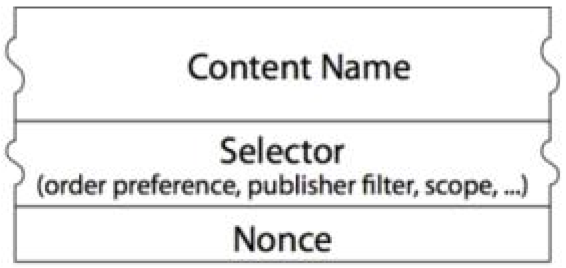


Figure 1: Interest Packet [10]

The Content Name is the name of the requested data which will be used to match the prefix. The Nonce carries a randomly-generated 4-octet long byte-string. The combination of Name and Nonce should uniquely identify an Interest Packet. This is used to detect looping Interests.

- *Data Packet*

After receiving an Interest Packet from a consumer, the Server(Producer) sends back a response packet containing the data which the consumer requested. This response packet is known as **Data Packet** in NDN terminology. The data packet has four parts namely *Content Name, Signature, Signed Information and Data.*

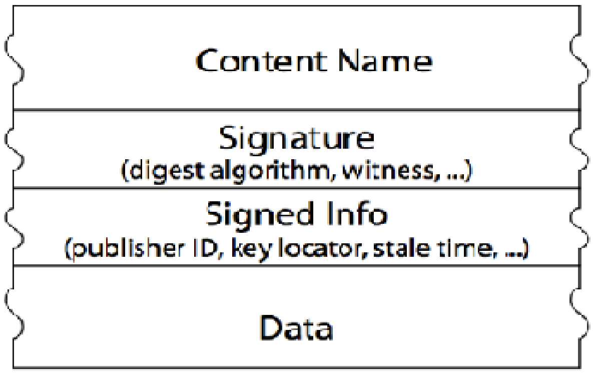

Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

Figure 2: Data Packet [10]

The Signature includes the digest algorithm, witness, etc. The Signed Information contains publisher ID, key locator, stale time, etc. FreshnessPeriod is an optional field which indicates the time for which a node should wait after the arrival of the data before declaring it non-fresh.

1.2.3 Routers

NDN supports multipath routing as it does not need to worry about looping as is the case in IP based routing. The routers in NDN maintain the following data structures to do their job.

1. **Pending Interest Table (PIT)**
2. **Forwarding Information Base (FIB)**
3. **Content Store (CS)**

▮ *Pending Interest Table*

The Pending Interest Table or PIT holds the entry records of the names of all pending interests and the interfaces from which the interests have been received. This allows the router to avoid forwarding the same interest packet individually for each consumer. If the incoming interest packet is already recorded in the PIT, the router simply adds this new incoming interface to the found record instead of uselessly forwarding this interest again.

After filling the PIT entry (or entries), it waits for the data packets, and when data packets arrive, router searches for the matching PIT entries. If the matching entries are found, then router sends the data to all the interfaces which are listed against the PIT name entries. After sending the data to all required interfaces, the router removes that particular PIT entry.

▮ *Forwarding Information Base*

Whenever consumer sends interest packet to the server, the router uses FIB. FIB decides the path of interest packet to the server by using hierarchical naming for data packets.

For example:- */youtube/videos/NDN* , */youtube/videos/NDN/2*

FIB stores two things about any interest packet:

1. **Prefix**

2. **Interface**

The prefix contains the name of the data which is requested by the consumer. The interface contains the interface of the router at which the request arrives. On the basis of its prefix entry, FIB forwards the interest packet towards the data source.

▮ *Content Store*

The router has some fix memory space to save those Data packets which are recently used. When this space is fully occupied, then router replaces some Data packet which is not used for a long time. To replace those Data Packet router uses LRU (Least Recently Used), LFU (Least Frequently Used), etc. algorithms.

There are two components stored in each entry in the content store:

1. **Name of data**

2. **Data**

The *"name"* part contains the content name and the *"data"* part contains the data for the future use. Whenever consumer requests for data in NDN it sends interest packet which first comes to the router and the router look for it in the content store. If data is available in the content store, then it sends back to the interface from where the request has arrived. If the data is not available in the content store, interest packet is forwarded according to Forwarding Information Base towards the data source. When data arrives back, the router caches it in the content store according to the popularity of the data.

1.2.4 Forwarding and Routing

Forwarding strategies in NDN are based not only on routing to calculate a single best path for reaching the destination. Rather, NDN nodes can dynamically select multiple interfaces from the FIB to forward every Interest Packet. Because of this real-time implementation, nodes are capable of fully utilizing their rich connectivity, and it prevents route hijacking attacks. The forwarding strategy layer also implements traffic control by adjusting the number of pending interests permitted.

The basic strategy of forwarding in NDN is to send an Interest in each interface in a FIB entry in a sequence. It can send interests on all interfaces at the same time and note which interface receives the data the earliest. The performance of the interfaces is monitored constantly, and if the performance level depreciates to a certain level, then actions are taken accordingly.

Any Routing algorithm that works for the IP can be used in NDN as well, but instead of using IP Address prefix we are using name prefix, and we update the route using interest and data packets. The number of next hops is also being computed for every name prefix.

1.2.5 Security in NDN

The most basic security feature of the NDN is embedded within its "thin waist," i.e., the name in each data NDN packet is held with the packet content through signature. This helps in providing the data integrity and the authentication of the origin of the packet by mapping between the packet signer and the origin of the packet.

▮ *Cache Pollution Attack*

NDN is primarily oriented towards large-scale content distribution. Due to this, every router provides an arbitrary amount of cache that store forwarded content on subsequent retrieval. However, this caching creates problems related to the caches including the pollution.In a cache pollution attack, the main aim of the attacker is to force the NDN routers to cache the non-popular content. Thus, a successful attack reduces cache hits of content requests from genuine consumers, affecting overall network performance and increase the utilization of the link.

These attacks do not prevent the users from retrieving the content. These attacks have two main forms:

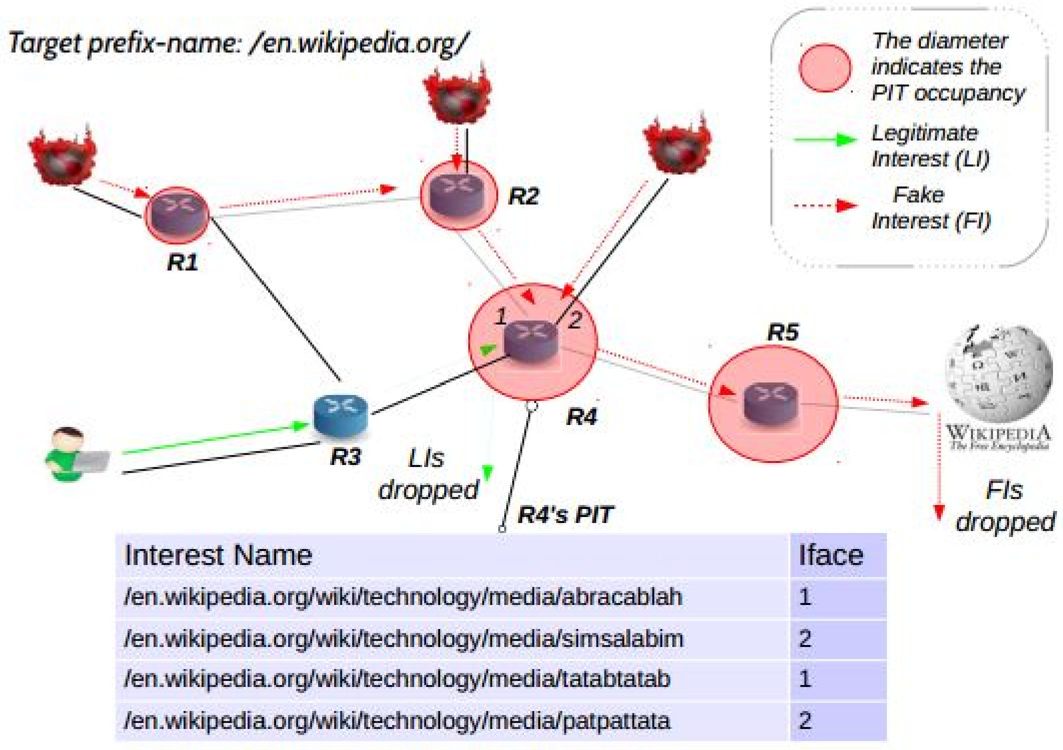
1. ***Disrupting cache locality***: In this, the attacker constantly issuing interest packets for non-popular contents of data, thus ruining the cache locality. Due to this, the adequacy of the cache is degraded, as popular content gets replaced by the non-popular content.

2. ***Creating a false locality***: In this, the attacker continuously and repeatedly requests the same but small set of non-popular content of data. Hence, creating the false locality in the cache.

▮ *Interest Flooding Attack*

Interest Flooding Attack or IFA is largely concerned with issuing a large number of Interests for non-existent content thereby flooding the Pending Interest Table which can severely disrupt the network services by decreasing the throughput delivered to legitimate consumers.

Interests whose aim is to fetch Data for a legitimate purpose are commonly referred to as Legitimate Interests (LIs). While Interests aiming to achieve network or producers service degradation are commonly referred to as Malicious Interests (MIs). Malicious Interests can be satisfied because they sometimes refer to the existing content or to dynamically generated content or can be fake if they refer to the non-existing content.



Interest Name	Iface
/en.wikipedia.org/wiki/technology/media/abracablah	1
/en.wikipedia.org/wiki/technology/media/simsalabim	2
/en.wikipedia.org/wiki/technology/media/tatabtatab	1
/en.wikipedia.org/wiki/technology/media/patpattata	2

Figure 3: Illustration of an IFA targeting a Wikipedia Content Provider [8]

Interest Flooding Attack targeting a specific NDN content provider is called *pure* IFA (pIFA). There are two simple attack variants [8] of pIFA:

- ***blended* IFA (bIFA):** In bIFA, attacker generate interest for both existent content and non-existent content. The main aim of a bIFA is to influence the detection metrics observed by routers, in order to both stay undetected and lower the probability for malicious Interests to be dropped. This causes countermeasures relying on satisfaction ratio as their detection mechanism to fail.

- ***chameleonic* IFA (cIFA):** cIFA includes the attackers which change the target prefix name after a certain time window. It tries to avoid the various countermeasures taken by the routers where few of the prefix names are marked as infected. This causes countermeasures relying on blacklisting certain prefixes used by attackers to fail.

1.3 Some Wonderful Minds

The idea of NDN was first explored by **Ted Nelson** in 1979 and then later by **Brent Baccala** in 2002 [3]. The proposal of avoiding DNS (Domain Name System) lookups was first put forward by the **TRIAD project** in 1999. A content-centric network architecture was first proposed by the **Data Oriented Network Architecture** (DONA) in 2006. It was an improvement of TRIAD by implementing security and authenticity in the architecture. NDN is an instance of a broad network research direction called Information-Centric Networking (ICN) under which lies various other architectural designs.

1.4 Major Contributions

Our contributions are as follows :-

1. Formulation and implementation of new approach for detection and mitigation of Cache Pollution Attack.
2. Implementation of Cache Protection Method Based on Prefix Hierarchy [5].
3. Implementation of CacheShield: Enhancing Cache Robustness approach [9].
4. Implementation of Interest Flooding Attack [8]
5. Implementation of Poseidon [1] for mitigating the Interest Flooding Attack [1].

6. Comparison of various results with the results of our approach.

Chapter 2

Related Work

Though the field is relatively new, due to the promising future of NDN, a lot of work has already been done related to it. The fields include routing, security, etc.

2.1 Cache Pollution Attack

The significant ones to detect/mitigate Cache Pollution Attacks are discussed below.

2.1.1 ANFIS-based Cache Replacement

ANFIS stands for Adaptive Neuro-Fuzzy Inference System [6]. ANFIS is an advanced Cache Replacement Policy, which uses Neuro-Fuzzy network to calculate a goodness value of all data packets in the network.

ANFIS works on each router independently by collecting stats about each data packet and then passing the information through 5 layers of a fuzzy network to refine the goodness value. This value is then used in the cache replacement policy decisions.

Though ANFIS can counter most attacks efficiently, we argue that it will fail to counter smartly designed attacks tailored specifically to defeat ANFIS. This is because, in ANFIS, each router is working completely independent of other routers. So, the decision of routers can be smartly manipulated by a nearby attacker. Since the router only has access to the local data, it may wrongly perceive the attacker's request to be legitimate and thus start caching contents requested by the attacker replacing the actual legitimate content from the cache.

2.1.2 CoMon++

CoMon is acronym for **Co**ordination with lightweight **Mon**itoring [7]. CoMon attempts to provide a central co-ordinated attack detection and mitigation while remaining light-weight to process in routers.

It classifies nodes into three components - the ISP Controller (IC), Monitoring Nodes (MNs) and NDN Nodes (NNs).

The MNs send the IC a list of contents they observed in and the corresponding number of requests periodically. The IC then forms a white-list of prefixes based on their popularity. It then distributes the contents of white-list to the monitoring nodes and instructs them to stick with this content.

This ensures that the popular data always remains cached in the network thus reducing the number of costly Inter-ISP data requests.

But, we believe that CoMon has its limitations. Firstly, the white-list ranking is formed primarily using the frequency of the data packet requests, that cannot be a sole deciding factor. Secondly, CoMon does not remember or use the previously calculated ranks of the data packets. Lastly, although CoMon may be able to reduce the number Inter-ISP requests, the requests within the networks still require few hops to get satisfied, since the popular contents are cached only at the monitoring nodes which are roughly 10% of all routers in the network.

2.1.3 Cache Protection Method Based on Prefix Hierarchy for Content-oriented Network

This Method or (CPMH) can defend caches from the malicious users after attack detection. This Method uses the hierarchy of content name prefixes to identify the unpopular contents requested by attackers from all requested contents. It has two purposes. One is reducing a storage cost as much as possible, and Second is to protect the cache without damaging normal users.

▮ *Steps Of CPMH*

CPMH consists of three steps as below.

- **Identifying the attackers' prefixes.** Identify the prefixes of attackers' requested contents and blacklist them.
- **Cache Recovery.** Recover the Cache by removing the blacklisted contents.
- **Protecting Cache.** Protect the Cache by avoiding storing the contents which have prefixes contained in the blacklist.

Identifying the Attackers' prefixes: Routers identify the prefixes which are not much popular among the normal users but are frequently requested by the attackers.The prefix method is used due to two different reasons, One is to reduce the storage cost and second is to extend the range of identifying contents requested by attackers.

To identify the prefixes requested by attackers, CPMH has three steps. First, routers calculate Request rate Variation per Prefix (RVP), but using RVP only causes a problem that routers mistake popular prefixes for attackers prefixes. Hence, routers calculate Weighted RVP (WRVP) in the next step and blacklist using the WRVP.

1. *Request rate Variation per Prefix (RVP)*: CPMH should identify the attackers prefixes without containing prefixes which are popular among normal users. CPMH implements this by using RVP and prefix length on the basis of the name hierarchy.

   RVP δ_f is calculated as :

   $$\delta_f = \sum_{i \in S_p} \delta_i \quad (1 \geq \delta_p \geq 0) \tag{1}$$

   where δ_i is defined as the variation of content and is calculated as:

   $$\delta_i = (\frac{n_r^m(i)}{N_r^m} - p(i)) \tag{2}$$

where m means the past measurement, and N_r^m means the requested count of whole selected contents at the past.

Request rate p(i) of content i is calculated as:

$$p(i) = \frac{n_r(i)}{\sum_{j \in S} n_r(j)} \tag{3}$$

where S means a set of selected monitoring contents, $n_r(i)$ means the number of requested count of content i and $\sum_{j \in S} n_r(j)$ means the total requested count of whole selected contents.

If routers use RVP to identify the prefixes of unpopular contents, there is a probability that they fail to detect popular prefixes as unpopular prefixes.

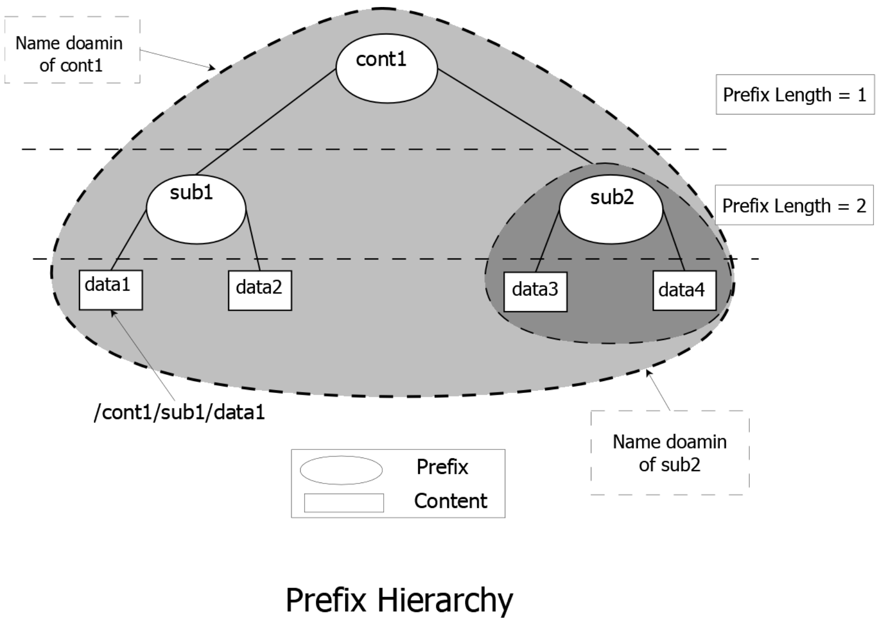


Figure 4: Prefix hierarchy and RVP [5]

Figure 3 depicts an example of the relation between a name prefix hierarchy and RVP. Prefixes tree structure like Hierarchy. We define that the RVPs of */cont1, /sub1* and */sub2* are δ_{p1}, δ_{p2} and δ_{p3} respectively. The RVP of */cont1* is calculated by adding the RVP of */sub2* to the RVP of */sub1*. Hence, δ_{p1} is bigger than δ_{p2} or δ_{p3}, which means that the prefix located near the root of hierarchy has a larger RVP.

2. *Weighted RVP:* CPMH uses length of prefix in order to solve the problem of the RVP. Routers have to identify as far prefix from root as they can, but do not have to leave attackers prefixes as normal prefixes. Prefix length which corresponds to the depth of tree is equal to the number of components which constitute prefixes.

   WRVP δ_{wp} is calculated as:

   $$\delta_{wp} = \delta_p \times \omega_l, \quad (1 \geq \delta_{wp} \geq 0, 1 \geq \omega_l \geq 0) \tag{4}$$

   where l means length of prefix, w_l is the weight which is for reflecting length in variation. The longer prefixes are, the larger w_l becomes, but maximum value is 1. In this way, WRVP reflects both the probability that is used by attackers and the depth of tree structure.

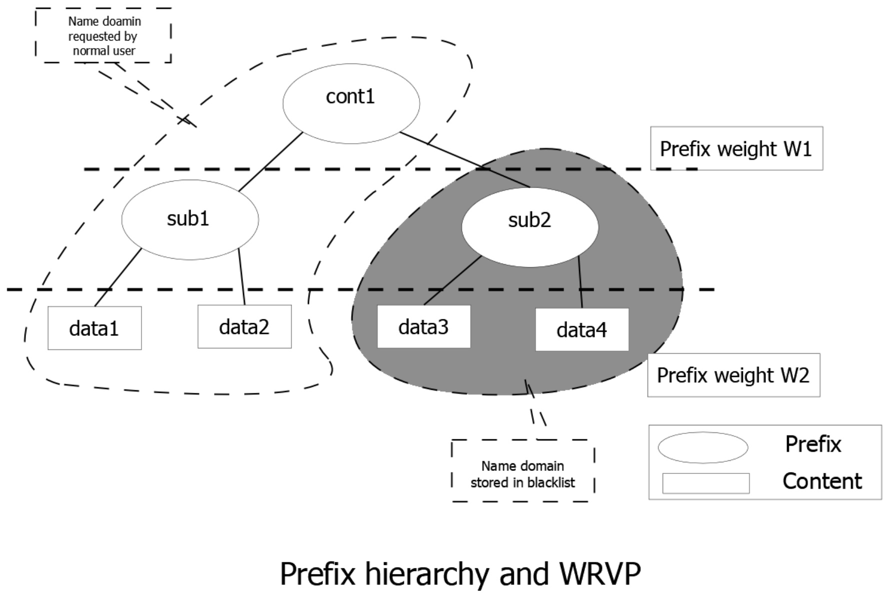

Name doamin requested by normal user
cont1
Prefix weight W1
sub1
sub2
data1
data2
data3
data4
Prefix weight W2
Name domain stored in blacklist
Prefix
Content
Prefix hierarchy and WRVP

Figure 5: Prefix hierarchy and WRVP [5]

Figure 4 depicts the relation between a name prefix hierarchy and WRVP. The weight multiplied to RVP is allocated to prefix length. The weight of prefix whose length is one is ω_1, and whose length is two is ω_2. ω_2 is larger than ω_1. δ_{p1} is larger than δ_{p2} because of name hierarchy, but the magnitude correlation between $\delta\omega_{p2}$ and $\delta\omega_{p2}$ does not depend on the hierarchy.

3. *Making a Blacklist:* After calculating WRVP, prefixes which have high WRVP are selected because they have higher probability that they are requested by malicious users. Blacklist threshold(τ) is defined using the WRVP of candidates. Prefixes with higher WRVP than threshold(τ) are marked as blacklisted. (τ) is calculated dynamically.

Cache Recovery: The cleansing of the Cache takes place using the blacklist. Routers erase the contents which have prefixes contained in blacklist.

Cache Protection: There are two methods to protect caches. One method is to avoid storing unpopular contents requested by attackers, and the other is discarding malicious Interests which request the unpopular contents. CPMH adopts first method. If a prefix of arrival content is included in the blacklist, the router does not store the content.

2.1.4 Enhancing Cache Robustness: CacheShield

Cache-Shield is a generic mechanism to deal effectively against hard-to-detect locality-disruption pollution attacks. Unlike other approaches CacheShield is a proactive mechanism, it takes measures prior to the attack rather than first letting the attack happen and then taking countermeasures.

CacheShield favors those objects that are more frequently requested for caching. CacheShield employs a shielding function to identify relatively popular content objects and filter out unpopular ones.As unpopular content objects are more likely to be kept out of CS, the impact of pollution attack can be greatly reduced. More importantly, the performance of caching can also be improved under normal circumstances due to the denial of entrance of unpopular content objects into the cache.

▮ *CacheShield and its Structure*

CacheShield can be seen as an add-on to the CS and is compatible with any cache replacement algorithm. There are two essential components in CacheShield:

- **A shielding function**

- **A record of content names**

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

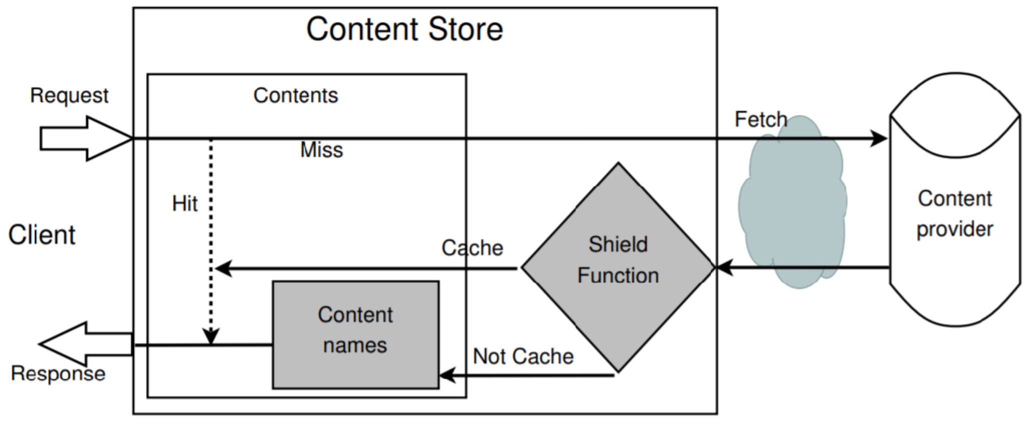


Figure 6: CacheShield Structure [9]

▪ *Working of CacheShield*

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

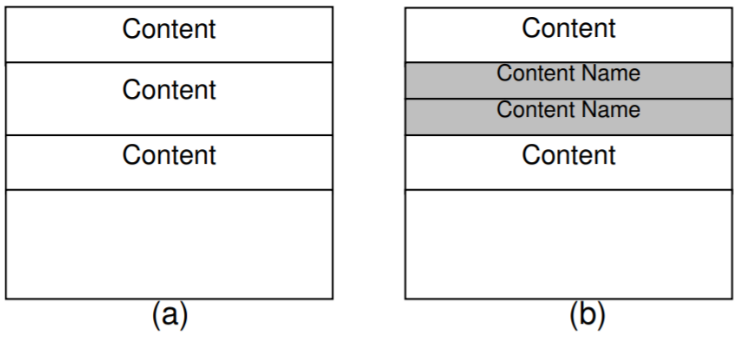


Figure 7: CacheShield CS (a) Without CacheShield (b) With CacheShield [9]

A content name is placed in a position of the stack exactly like a content object according to the caching algorithm.For example, the content name of a newly requested content object will be placed at the top of the stack if CacheShield decides not to cache the content object. Although content names consume additional storage, they are much smaller than typical content objects. When a request finds a matching content name but CacheShield decides not to cache the content object, it will record the number of attempted requests attached to the content name. CacheShield uses this information to make future decision. When a request finds a matching content name and CacheShield decides to cache the content object, the content name will be replaced by the content object.

- *Shielding Function*

CacheShield deals more efficiently with locality-disruption attack and works with a variety of cache replacement algorithms. When a request arrives at CS and the requested content object is already in CS, CS uses the original caching algorithm. Otherwise, CacheShield runs a shielding function.

The objective of the shielding function is to minimize unpopular content objects from being cached in CS. A probabilistic function as the shielding function. The function calculates a caching probability for each requested content object. The more requests a content object receives, the higher the corresponding probability will be and so it is more likely to be cached. An example of shielding function can be cited by the following logistic function:

$$\psi(t) = \frac{1}{1 + e^{(p-t)/q}}, t = 1, 2, .. \tag{5}$$

where t denotes the t^{th} request for a given content object in CS, and p and q are the parameters of the function. With probability ψ, a new content object is fetched into CS for possible future use. Otherwise, it is not placed in CS. Rather, the content name and its number of requests, t,is recorded in CS. As the ψ value of requests for unpopular content objects is small, unpopular content objects are less likely to be cached and cause relatively popular content objects in the cache to be removed. Thus, the logistic function can shield CS from pollution attacks in which requests are for very unpopular content objects. The parameters p and q provide each deployment site with fine control of the probability for popular requests. A shielding function is not necessarily to be the logistic function. Any function that can effectively differentiate requests for unpopular content objects from those for relatively popular content objects is a good candidate of the shielding function.

2.2 Interest Flooding Attack

The currently used methods against Interest flooding attacks should fail to improve performance against bIFA and cIFA attacks. To analyse these attacks and the performance of countermeasures against them, we studied and implemented the following works.

2.2.1 Poseidon

Poseidon[1] is used for detecting and mitigating interest flooding attacks. It consists of a detection and a reaction phase. Detection is either local or distributed. In this paper, a collaborative approach called push-back is used to counter interest flooding. Poseidon is a set of algorithms that execute on routers to identify traffic anomalies (especially, interest flooding) and mitigate their effects. It continuously monitors per-interface rates of unsatisfied interests with respect to overall traffic. If the rates change significantly between two consecutive time intervals, it sets a filter on the offending interface(s) that reduces the number of incoming interests. Also, it can generate a push-back alert message to the those interfaces, to signal that an interest flooding attack is in progress.

▮ *Detection Phase*

Attacks are detected using two parameters: $\omega(r_i^j, t_k)$ and $\rho(r_i^j, t_k)$. $\omega(r_i^j, t_k)$ is the number of incoming interests divided by the number of outgoing content packets, observed by a router r_i on its interface r_i^j within time interval t_k :

$$\omega(r_i^j, t_k) = \frac{number of interests from r_i^j at interval t_k}{number of content packets from r_i^j at interval t_k} \tag{6}$$

$\rho(r_i^j, t_k)$ is the number of bytes used to store interests in PIT, coming from interface r_i^j within time interval t_k.

Poseidon detects an attack when both $\omega(r_i^j, t_k)$ and $\rho(r_i^j, t_k)$ exceed their respective thresholds $\omega(r_i^j)$ and $P(r_i^j)$. The detection algorithm is run at fixed time intervals, for example every 60 ms, and in the presence of push-back messages. $\omega(r_i^j, t_k) > 1$ shows that the number of content packets forwarded to r_i^j is smaller than the number of interests coming from the same interface. But a small bursts of regular or non-satisfiable interests may not be caused by an attack. So taking into account only $\omega(r_i^j, t_k)$ and not considering $\rho(r_i^j, t_k)$ may cause the detection algorithm to report a large number of false positives. If countermeasures are applied, in this case, it'll produce negative effects to the overall network performance.

To improve detection accuracy, Poseidon takes into account $\rho(r_i^j, t_k)$ also. It measures the PIT space used by interests coming from a particular interface. This maintains the number of false positives to be low when compared to just using $\omega(r_i^j, t_k)$ while detecting low-rate interest flooding. In a low-rate interest flooding attack, the adversary limits the rate of fake interests to keep $\omega(r_i^j, t_k)$ below the thresholds. Monitoring the content of the PIT allows Poseidon to keep an eye on the effects of the attack, rather than just its causes, allowing for early detection.

▮ *Reaction Phase*

As soon as the interest flooding attack from any interface of router is detected, the rate of incoming interests from that interface is limited until all the detection parameters do not fall under their predefined thresholds. Once detected, the router issues an alert message on each of the interfaces to convey information about the ongoing interest flooding attack.

An alert message is a content packet rather than being an interest packet so as to take the advantage of content packets being signed while interests packets are not signed. This helps in finding verifying the legitimacy of the alert message. Routers running Poseidon do not process alert messages as regular content: alerts are not checked against PIT content and are not forwarded any further.

This push-back mechanism of issuing the alert messages allows routers that are not the target of the attack, but are unknowingly forwarding malicious interests, to detect interest flooding in earlier phase. In particular, alert messages allow routers to detect interest flooding even when they are far away from the intended victim i.e., close to nodes controlled by the adversary, where countermeasures are most effective.

Chapter 3

Proposed Work

3.1 Cache Pollution Attack

Our primary goal is to detect and mitigate the Cache Pollution Attack and reduce its negative impact on the network. We have made an offline Machine Learning based Classifier and have implemented an idea to mitigate the attack in an effective and systematic manner.

We have learned from the works of previous authors that taking caching decisions on the basis of the popularity of contents calculated by the local router is highly ineffective against smart, manipulative attacks from nearby attackers. Also, ignoring the previously calculated popularity ratings for data may lead to wide fluctuations in hit ratios and may fail against sudden high-frequency attacks.

Our method has been designed to defeat these limitations. The main assumption that we used is that the fraction of attackers in a network is less as compared to the consumers. This assumption is quite general in nearly all realistic attack scenarios. Due to this observation, the number of distinct users requesting a name prefix becomes a significant factor in deciding the popularity of the name prefix.

Hence, we collect global data about the number of distinct users for each prefix periodically over time and use this data to calculate the popularity rating of each prefix in the current period. This newly calculated rating is passed through a method called **exponentially weighted moving average (EWMA)** [4] which is used to calculate the subsequent popularity for each name prefix.

Algorithm 1 OnInterest (Trigger on interest packet retrieval)

```
INPUT: interest, inFace
 1: $hopCount = interest.hopCountTag$
 2: if $hopCount = 1$ then
 3:     $name = interest.name$
 4:     $DRS.store(name, inFace)$                ▷ DRS : Distinct Request Set
 5: Usual interest process
```

Let the total numbers of users (legitimate users and attackers together) in the network be n, the number of distinct users requesting a name prefix in the current time period T_i be x_i, then the new rating R_i of the prefix for this interval is calculated as-

$$R_i = \frac{x_i}{n} \tag{7}$$

The initial EWMA of each prefix A_0 is set to 1.0. The later values for each time interval T_i $(i = 1...n)$ is calculated as-

$$A_i = \alpha.R_i + (1 - \alpha).A_{i-1} \tag{8}$$

Where, α is the smoothing factor which lies in the open interval $(0, 1)$.

This updated EWMA value is then used for making caching decisions in the next time interval. We have to decide upon a threshold value Th. This is selected by hit and trial for a network and usually lies between 0.1 to 0.4 for most networks.

Algorithm 2 PeriodicPopularityUpdation (Trigger with a time period T_i)

```
 1: for each $d \in DRS$ do
 2:     $name = d.name$
 3:     increment $NDR[name]$ by 1           ▷ NDR : Number of Distinct Requests
 4: $clearDRS$
 5: for each $name \in NDR$ do
 6:     $currentRating = NDR[name]/userCount$
 7:     if $name \in PRD$ then               ▷ PRD : Popularity Rating Data
 8:         $pastRating = PRD[name]$
 9:     else
10:         $pastRating = 1.0$
11:     $PRD[name] = \alpha * currentRating + (1 - \alpha) * pastRating$
```

Now, whenever a router receives a data packet -

- if the cache is not full, the data is cached irrespective of its EWMA popularity rating.
- If the cache is full, the decision of whether to cache the data is taken on the basis of whether the EWMA popularity of the data lies above or below the threshold Th value.

Algorithm 3 OnData (Trigger on data packet retrieval, implies a cache miss)

```
INPUT: data, inFace
1: $name = data.name$
2: if $name \notin PRD$ then
3:     $PRD[name] = 1.0$
4: if $PRD[name] > Th$ then
5:     Add data to CS                                  ▷ CS : Content Store
6: Usual data process
```

This idea eliminates the limitations of previous work up to a great extent. Since we are using global data of the network, it will not be possible for an attacker to manipulate our method into caching its data. Also, using EWMA, we are using our previously calculated ratings in each iteration. This helps to smooth out the hit ratios over time and removes wide fluctuations. Since we are taking into account the number of distinct users issuing a prefix instead of the frequency of the prefix, this method easily counters high-frequency attacks.

In addition to this idea, we have implemented an offline attack detector using Machine Learning. In particular, we have used the Support Vector Machine (SVM) classifier which detects if an attack is ongoing on a router interface. This classifier uses features such as the number of incoming and outgoing interest and data packets on the interface along with the number of cache hits and misses of the incoming interest requests.

3.2 Interest Flooding Attack

Our primary goal is simulate the advanced interest flooding attacks namely-

- The blended IFA (bIFA)
- The chameleonic IFA (cIFA)

and analyse their effects on realistic networks. We also aim to analyse how the currently used countermeasures against interest flooding attacks fare against the advanced ones.

For performing interest flooding attack, we needed a stream of distinct non-existing interest packets. So we made i^{th} producer listen to prefix $/producer_i$, but only reply to interest packets with prefix $/producer_i/data$. This ensured that the only the first part of the prefix i.e., $/producer_i$ is stored in the forwarding information base (FIB) of routers which results in any interest starting with the first part of the prefix to be routed all the way to the respective producers. This results in new entries in PITs of all routers along the way, thus maximizing the impact of the attack.

Algorithm 4 SendPacket (Triggers based on attacker request frequency)

```
1: $prefix = "/producer"$
2: $prefixNo = getRandomInt(1, np)$                ▷ np: number of producers
3: $prefix.append(prefixNo)$
4: $decisionRatio = getRandomDouble(0.0, 1.0)$
5: if $decisionRatio < existingPrefixRequestRatio$ then      ▷ Set to 0.0 for cIFA
6:     $prefix.append("/data")$
7: else
8:     $prefix.append("/notdata")$
9: $counter[prefixNo] = counter[prefixNo] + 1$
10: $seqNo = counter[prefixNo]$
11: $prefix.append(seqNo)$
12: Usual packet creation process
13: Usual packet sending process
```

To make the non-existing interest packets unique, so that new PIT entries are created each request, we have used a non existing prefix of the format $/producer_i/notdata/j_i$, where j_i is a ever increasing counter per producer prefix.

Most of the currently used countermeasures rely on satisfaction ratio or prefix based blacklisting or a combination of both. While these work well against the traditional IFA, but they will fail against bIFA or cIFA.

To demonstrate this, we have implemented one such popular countermeasure, Poseidon [1]. As stated earlier Poseidon relies on two parameters to detect an ongoing attack-

- Satisfaction Ratio per interface
- PIT usage per interface

Now since in bIFA the attacker requests a mix of both existing and non-existing interests, this would up the satisfaction ratio and hence, Poseidon should fail to counter this attack in realistic networks. As far as cIFA is concerned, Poseidon may be able to detect and mitigate it in some cases.

Chapter 4

Experimental Setup and Results Analysis

4.1 Cache Pollution Attack

We have used **ns-3** simulator for testing out our idea. First, we implemented attacks and then have used our method to counter it. We will refer to our mitigation process as **Shield** from now on. We have considered two popular topologies for our simulation-

- the Xie Complex (XC) topology
- the DFN topology

Both of these topologies have been used quite a lot in earlier works and has proved to be a realistic indicator of general NDN networks.

The First Scenario is the **Xie Complex** topology (XC topology). This is a network with several consumers and just one attacker. In this scenario, we have taken the sole attacker's frequency as 20 times of the average consumer frequency. This has been done to demonstrate the maximum impact a single attacker can have on a network.

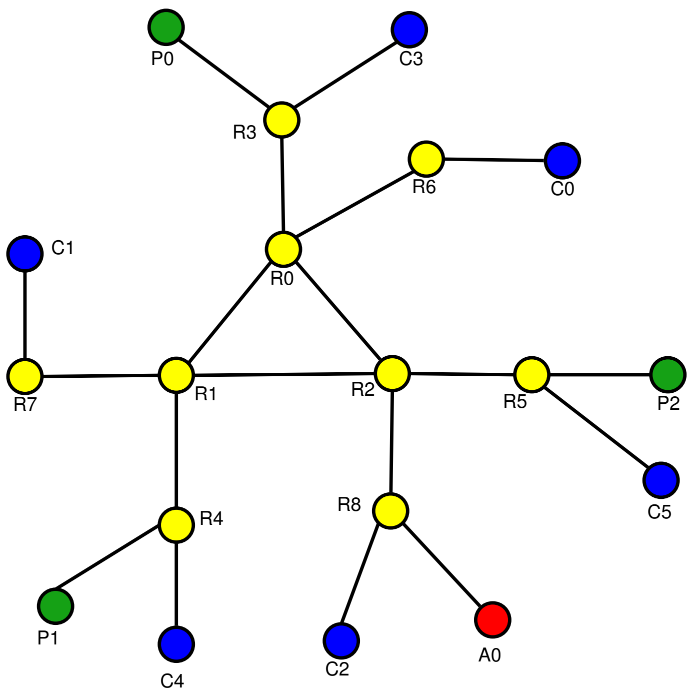
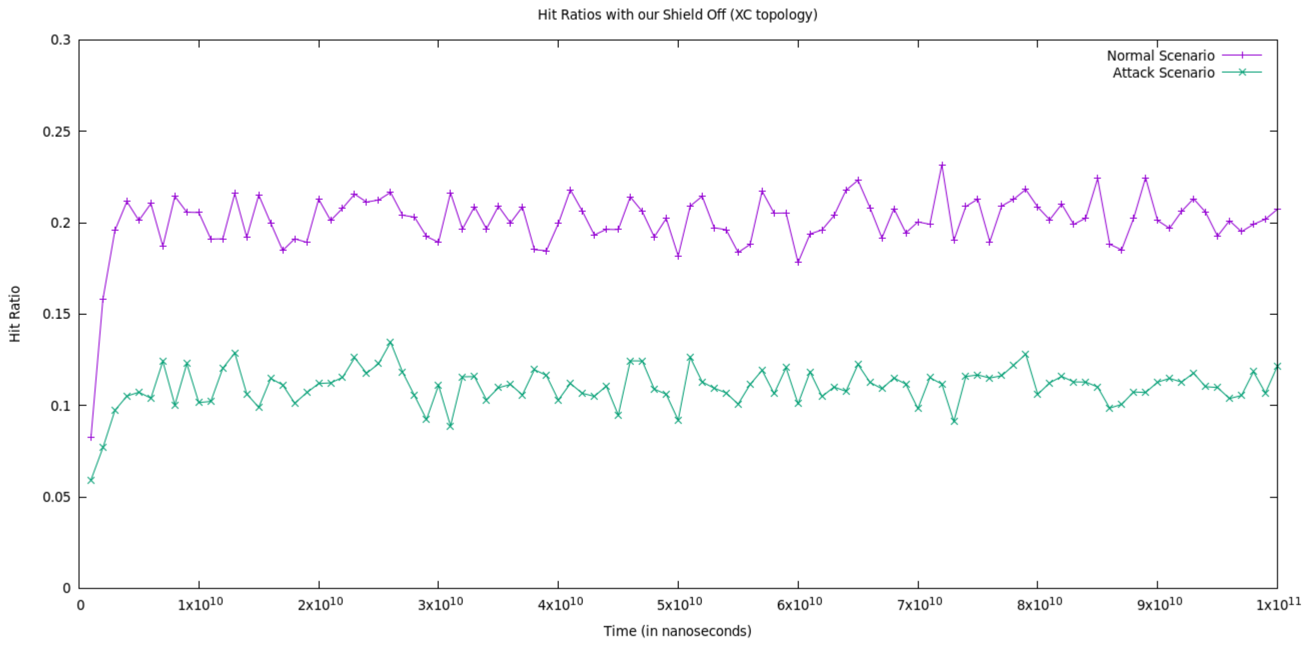
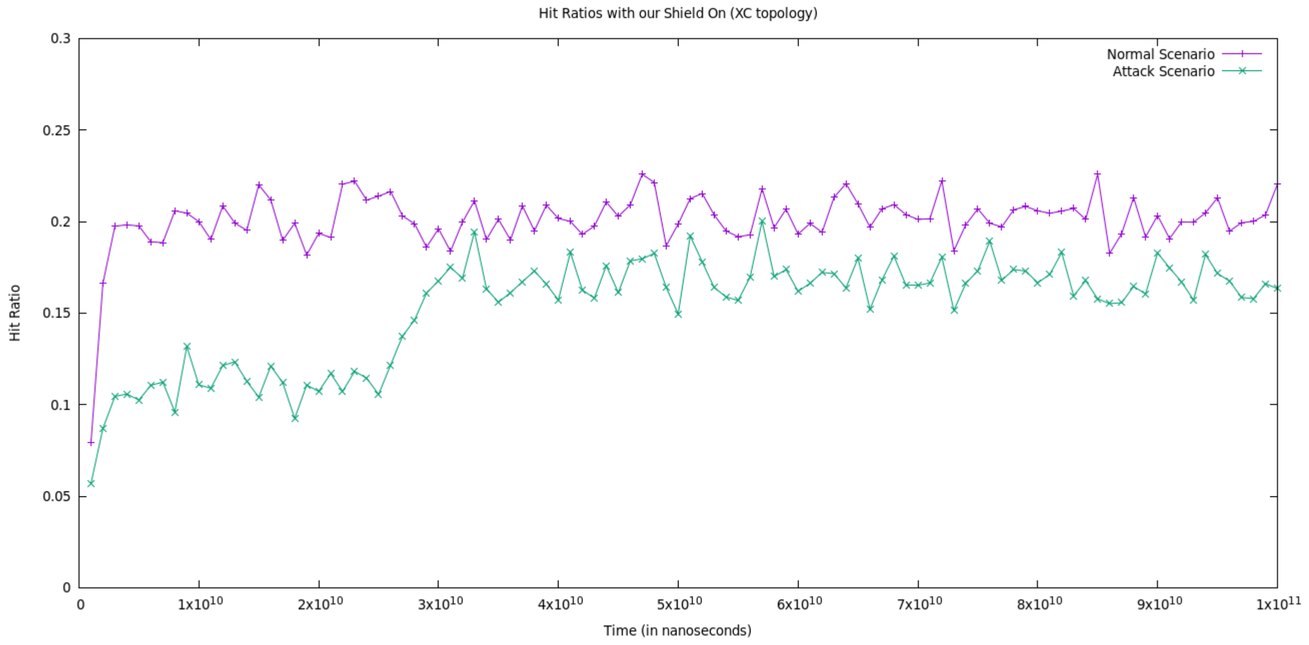


Figure 8: XC Topology [2]

In the simulation, the average hit ratio is 0.21 which drops to 0.12 under attack. For our Shield, we have got best results for parameters $\alpha = 0.1$ and $Th = 0.2$. As is evident from the graphs in Fig 9, the Shield can reduce the impact of the attack to a great extent. It takes around $25s$ to train, during which the hit ratios remain low. However, soon after the hit ratios rise close to the normal scenario.



(a) With Shield Off



(b) With Shield On

Figure 9: Hit Ratios in XC Topology

The Second scenario is **Deutsches Forschungsnetz (DFN)** topology. This topology is noted for its complexity which is close to a realistic scenario. This has numerous consumers with as many as 4 attackers. The attackers are also spread strategically to have maximum impact on the network. In our experiment, we have set the attacker frequency to $5 times$ that of the average consumer frequency.

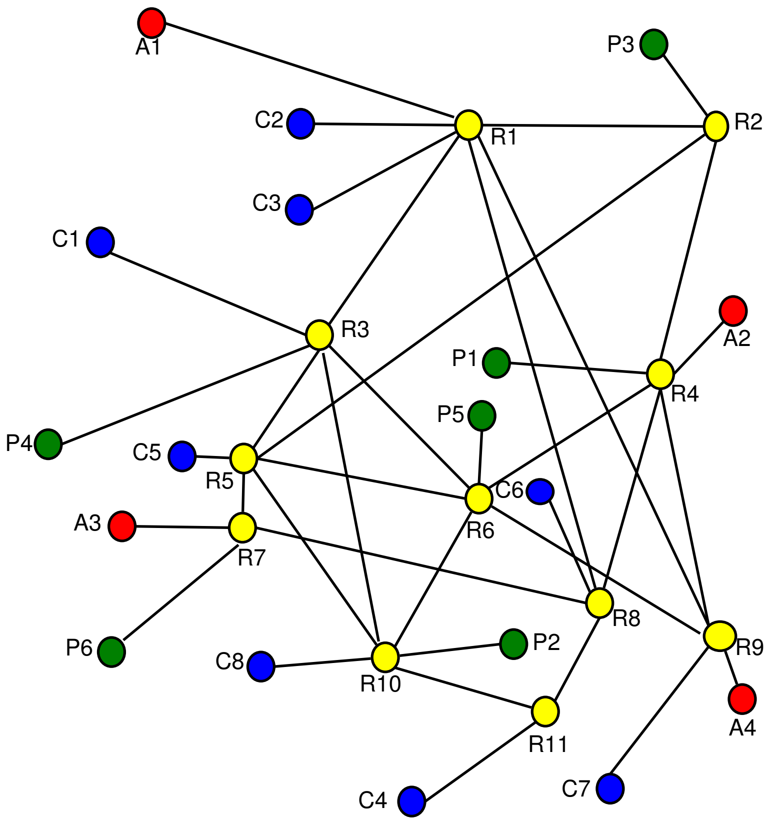
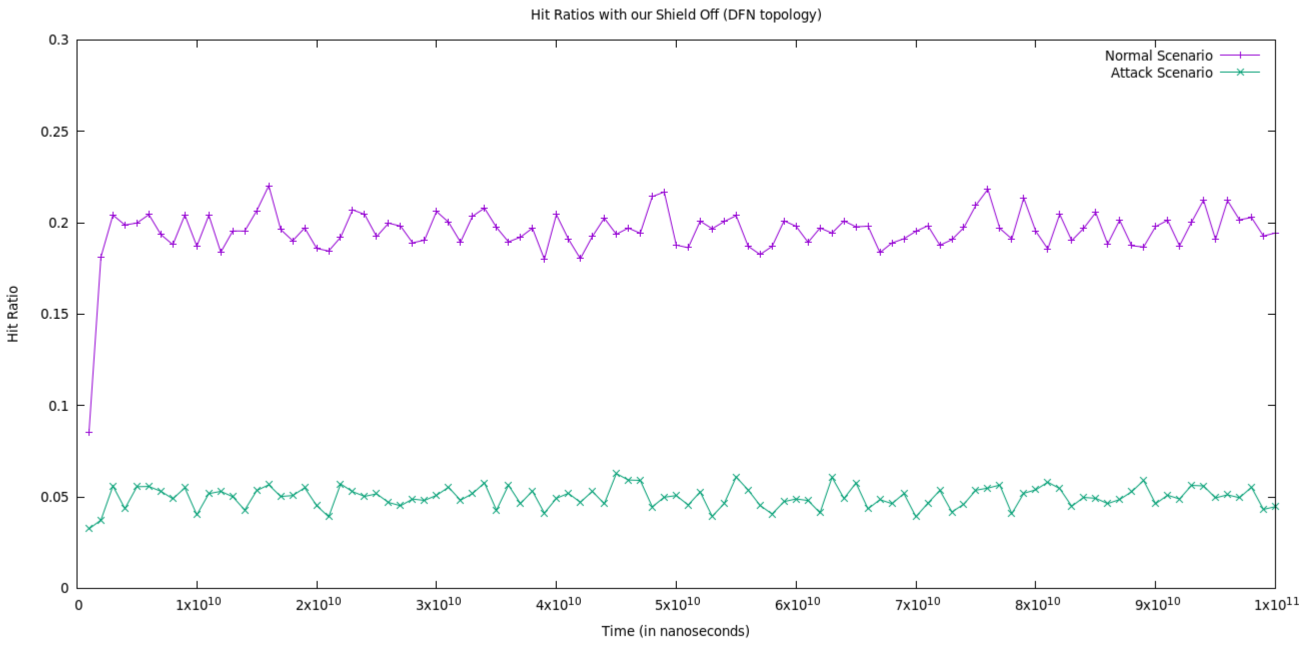
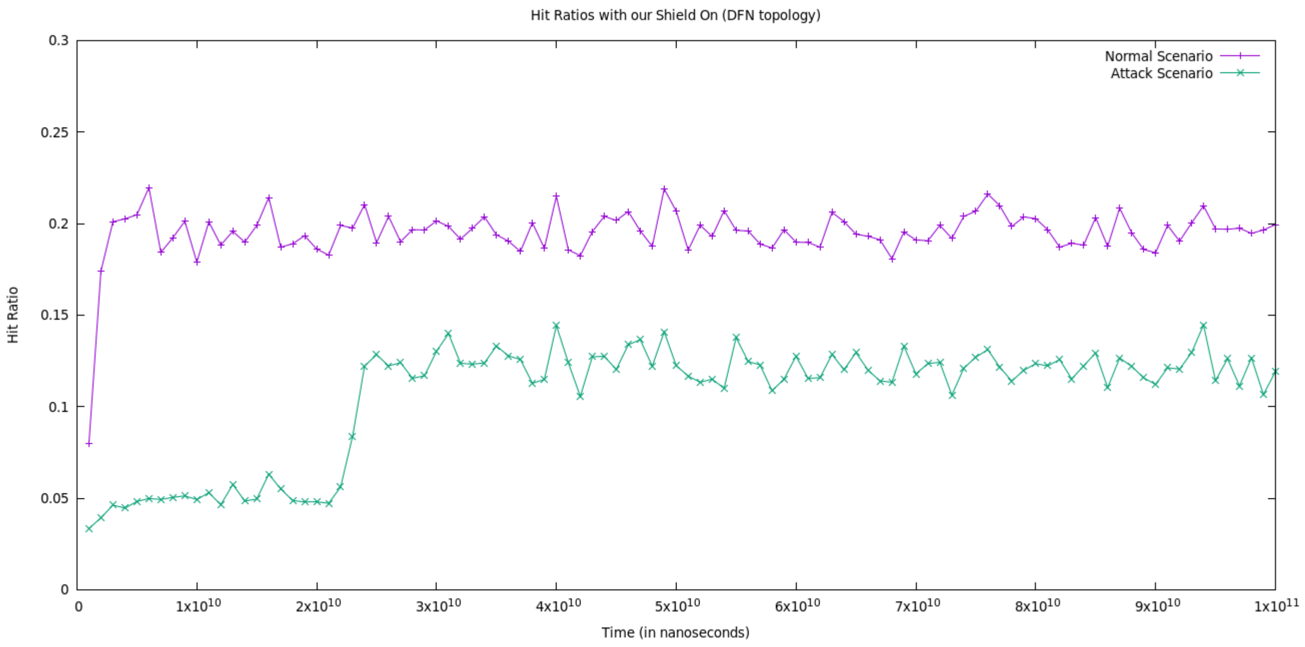


Figure 10: DFN Topology [2]

Here, the average hit ratio under normal scenario is 0.20 which reduces to as low as 0.06 when under attack (see Fig 11), due to the larger number of attackers. However, our Shield is still able to counter the attack up to a great extent and can increase back the hit ratios to around 0.13, after a training time of around $21s$.



(a) With Shield Off

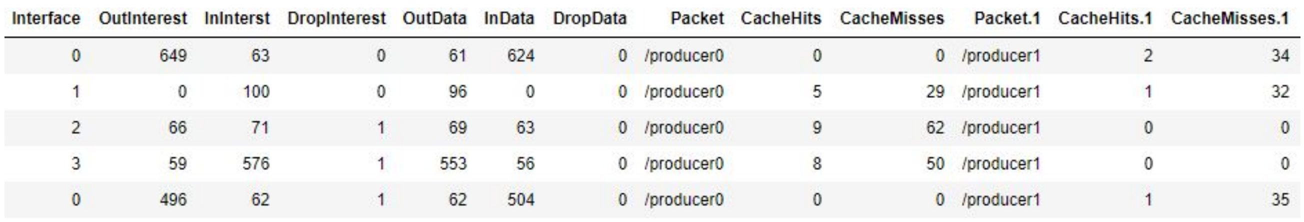


(b) With Shield On

Figure 11: Hit Ratios in DFN Topology

Hence, the relative comparison in the graphs (Fig 9 and Fig 11) shows that the with the help of our mitigation strategy(Shield), we can narrow down the gap between both the curves under two very different topologies, thus exhibiting the ability of our method to counter attacks under different scenarios.

We have also implemented an offline attack detector using Machine Learning approach. *Table 1* below shows the table of features used to train the model to detect whether the attack is being done on an interface of a router.



Interface	OutInterest	InInterst	DropInterest	OutData	InData	DropData	Packet	CacheHits	CacheMisses	Packet.1	CacheHits.1	CacheMisses.1
0	649	63	0	61	624	0	/producer0	0	0	/producer1	2	34
1	0	100	0	96	0	0	/producer0	5	29	/producer1	1	32
2	66	71	1	69	63	0	/producer0	9	62	/producer1	0	0
3	59	576	1	553	56	0	/producer0	8	50	/producer1	0	0
0	496	62	1	62	504	0	/producer0	0	0	/producer1	1	35

Table 1: Features used for training

The fitting of SVM is shown below:

```
form sklearn import svm
clf = svm.LinearSVC()
print(clf)
a = clf.fit(B, A)
```

After training the model with above features, we predicted whether the attack was executed or not.

```
ans = clf.predict(D)
```

After prediction, accuracy obtained obtained by the model is shown below:

```
clf.score(D, C)
0.6750000000000004
```

The Confusion matrix for the accuracy of the SVM classifier used is shown below:

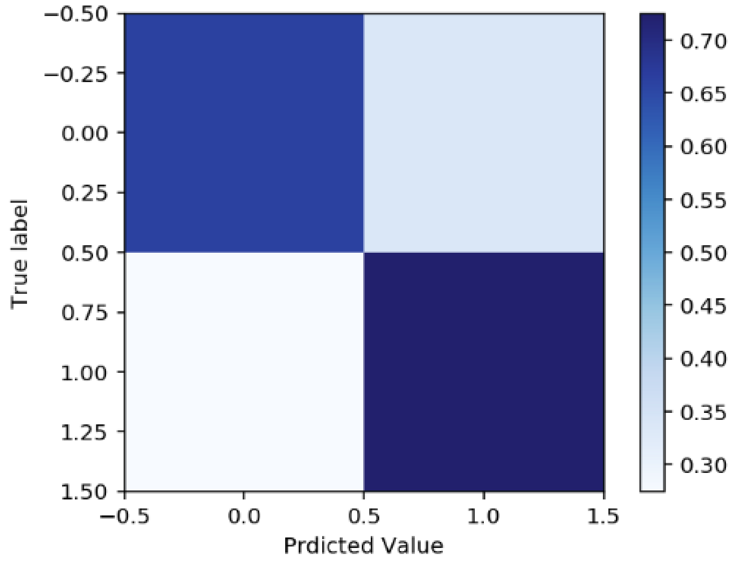


Figure 12: Confusion Matrix of model

4.2 Interest Flooding Attack

We have used **ns-3** simulator for testing out the idea that the existing countermeasures against Interest Flooding Attack are not effective against the newly discovered variations namely, bIFA and cIFA. First, we implemented the attacks and then have used the method described in Poseidon to counter it. We have considered three popular topologies for our simulation-

- the Linear topology
- the Xie Complex (XC) topology
- the DFN topology

All of these topologies have been used quite a lot in earlier works and has proved to be a realistic indicator of general NDN networks.

The First Scenario is the **Linear** topology. It is a very simple topology consisting of a single consumer, a single producer and a single attacker. We have used this simple topology to analyse the efficacy of the IFA attacks and the used countermeasure, i.e., Poseidon.

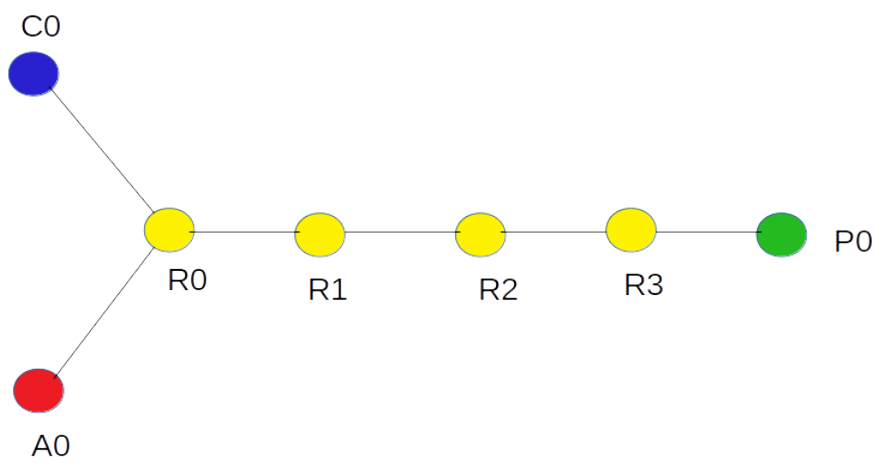


Figure 13: Linear Topology

In the simulation, the average satisfaction ratio is 1.00 which drops to 0.72 under attack. For Poseidon, we have got best results for parameters $satisfactionRateThreshold = 2.0$ and $pitSpaceUsedThreshold = 0.6$. As is evident from the graphs in Fig 14, it reduces the impact of the attack to a great extent increasing the average satisfaction ratio to around 0.90. Due to the simple structure of the topology the attacker interface is easily identified and hence the effectiveness.

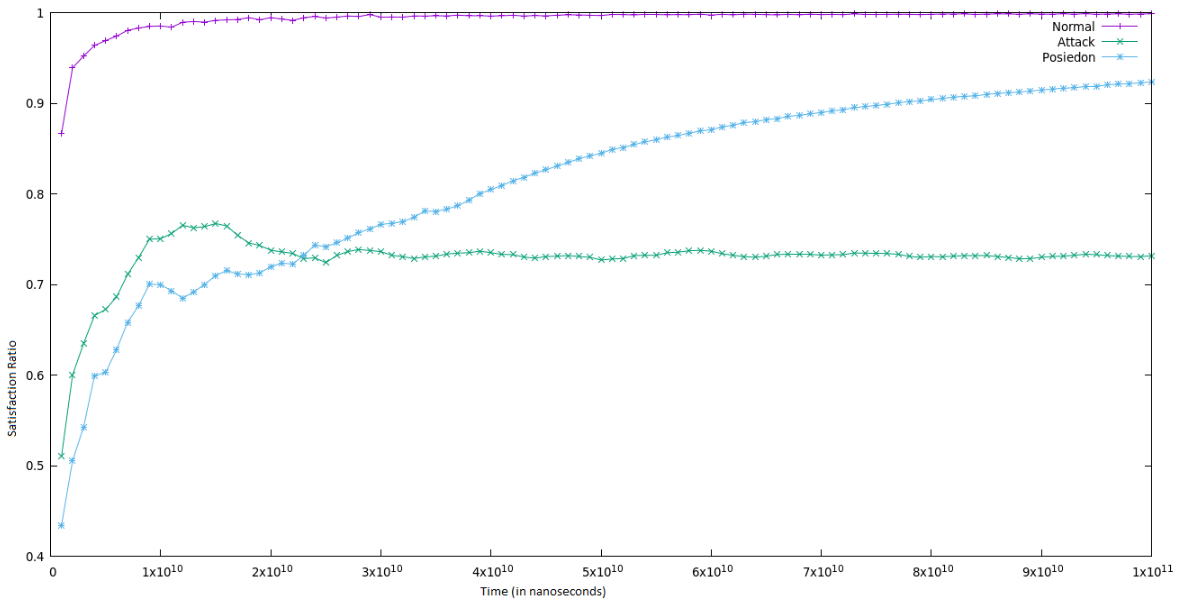


Figure 14: IFA and Poseidon in Linear Topology

The Second Scenario is the **Xie Complex** topology (XC topology). This is a network with several consumers and just one attacker. In this scenario, we have taken the sole attacker's frequency as 20 times of the average consumer frequency. This has been done to demonstrate the maximum impact a single attacker can have on a network.

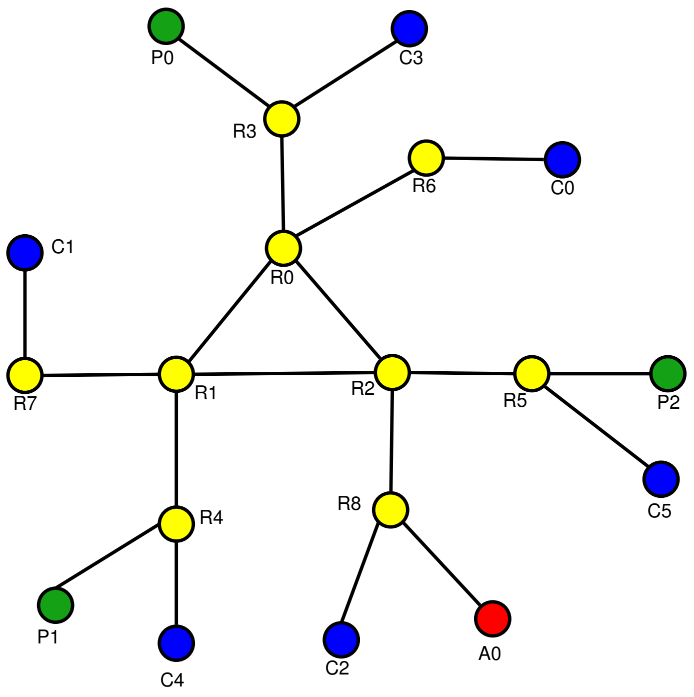
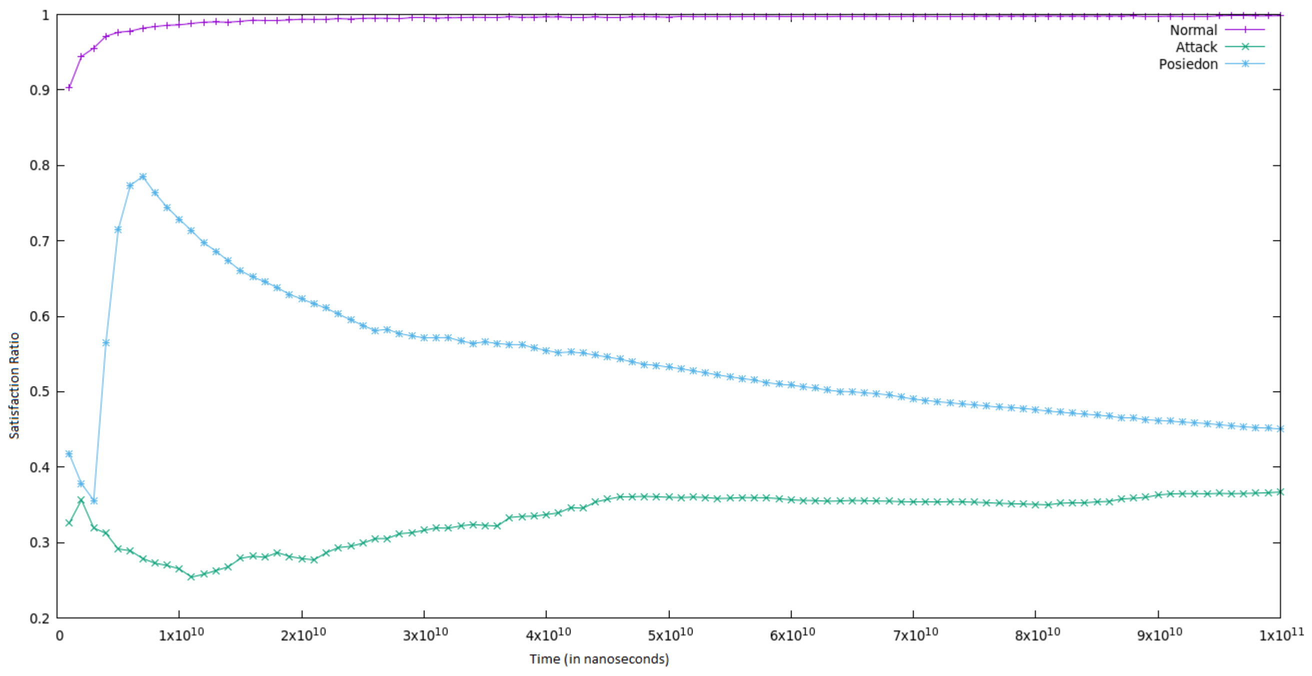
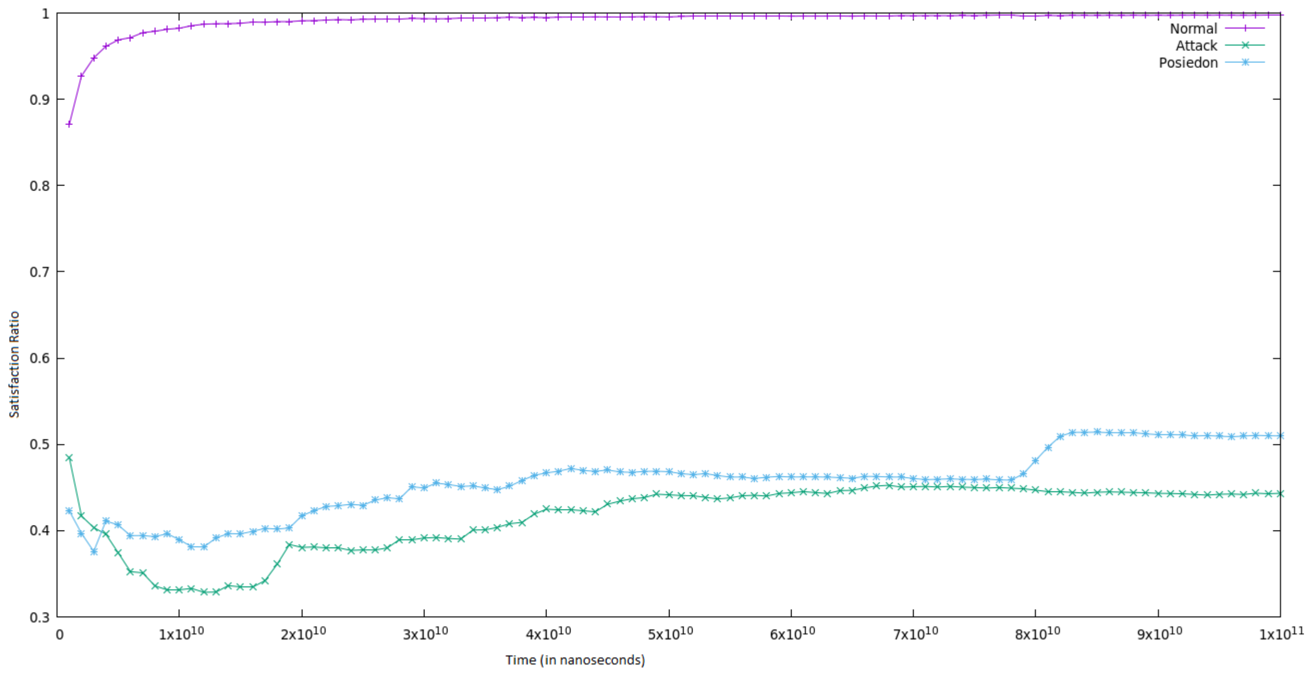


Figure 15: XC Topology [2]

In the simulation, the average satisfaction ratio is 1.00 which drops to 0.37 and 0.44 under attacks by cIFA and bIFA respectively. For Poseidon, we have got best results for parameters $satisfactionRateThreshold = 10.0$ and $pitSpaceUsedThreshold = 0.15$. As is evident from the graphs in Fig 16, Poseidon is able to reduce the impact of the impact of the attack to some extent in cIFA, but fails in the case of bIFA as we claimed.



(a) cIFA and Poseidon



(b) bIFA and Poseidon

Figure 16: Satisfaction Ratios in XC Topology

The Third scenario is **Deutsches Forschungsnetz (DFN)** topology. This topology is noted for its complexity which is close to a realistic scenario. This has numerous consumers with as many as 4 attackers. The attackers are also spread strategically to have maximum impact on the network. In our experiment, we have set the attacker frequency to $5 times$ that of the average consumer frequency.

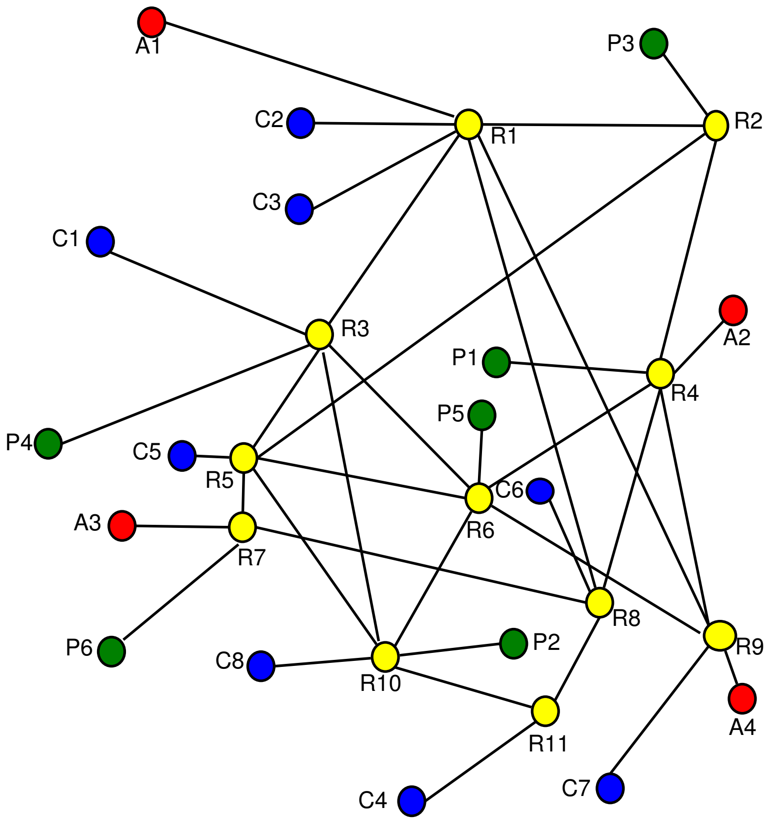
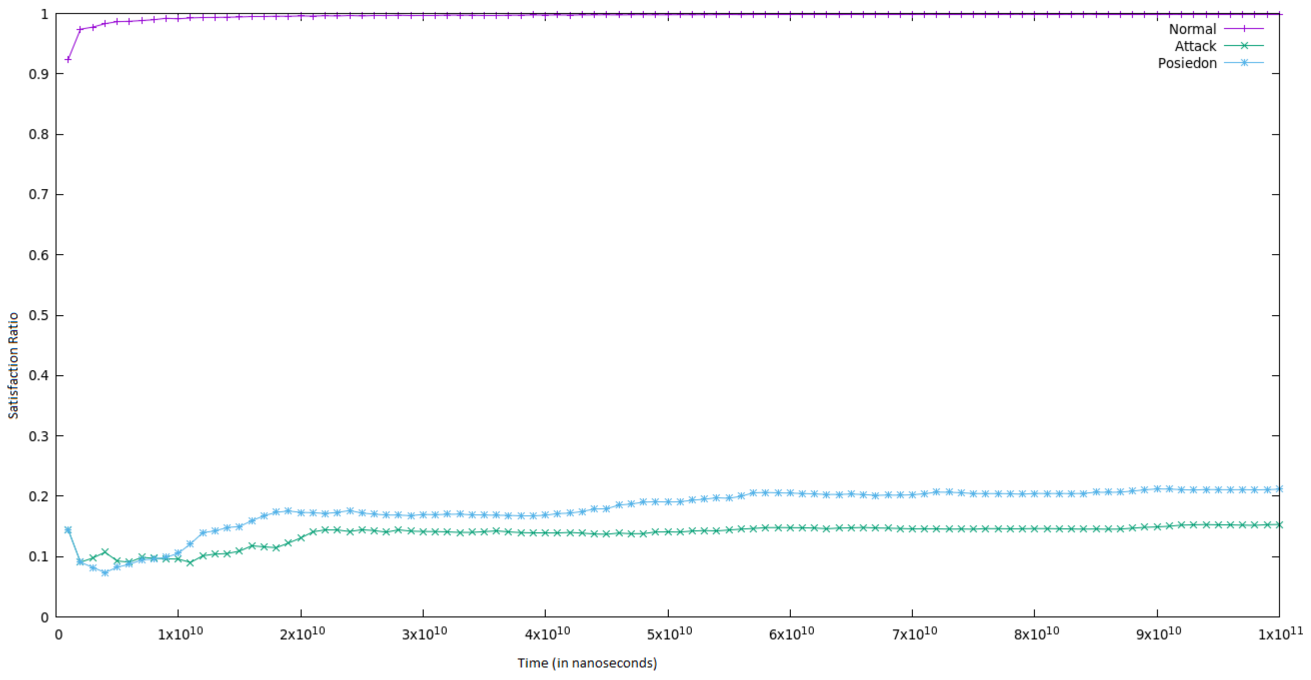
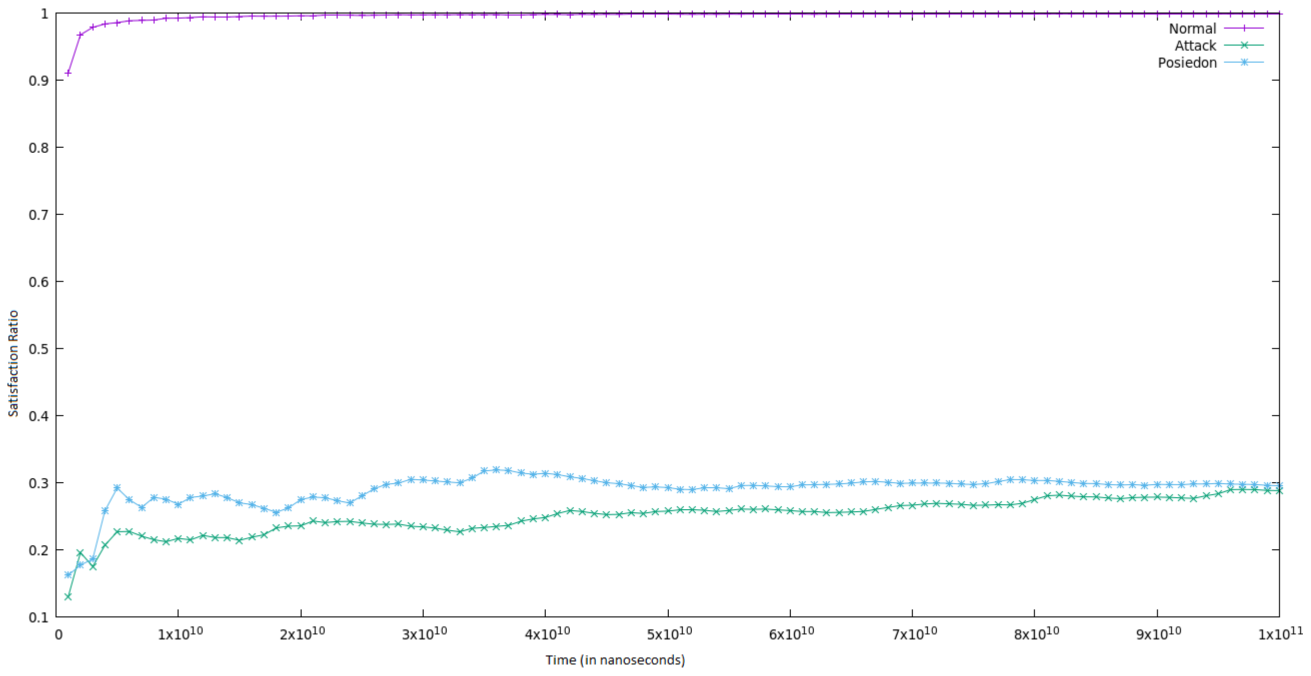


Figure 17: DFN Topology [2]

Here, the average satisfaction ratio under normal scenario is 1.00 which reduces to as low as 0.15 and 0.28 when under attack (see Fig 18) by cIFA and bIFA respectively, due to the larger number of attackers. In this scenario Poseidon fails to provide any significant advantage in both cases. While, a minor improvement is seen in case of cIFA, the average satisfaction ratios do not improve in the case of bIFA.



(a) cIFA and Poseidon



(b) bIFA and Poseidon

Figure 18: Satisfaction Ratios in DFN Topology

Hence, the relative comparison in the graphs (Fig 16 and Fig 18) shows that while Poseidon and other similar currently used countermeasures do well against traditional IFA, but fail to improve performance significantly in case of bIFA and cIFA.

Chapter 5

Comparison with Other Cache Protection Methods

We have compared our work with two other cache protection methods.

- Cache Protection Method Based on Prefix Hierarchy by Kamimoto et al. [5].
- Enhancing Cache Robustness: CacheShield by Xie et al. [9].

5.1 Cache Protection Method Based on Prefix Hierarchy

This method detects malicious user requests by observing the variation in request rate of the prefix being requested. Request rate p(i) of content i is calculated as:

$$p(i) = \frac{n_r(i)}{\sum_{j \in S n_r(j)}} \tag{9}$$

It then blacklists the prefixes requested by attackers if the variation in request rate goes any beyond the Blacklist threshold(τ). This value may have to be adjusted according to the topology of the network. After experimenting with various values, we chose the $\tau = 0.1$, as it gave the best results for the topology we have used.

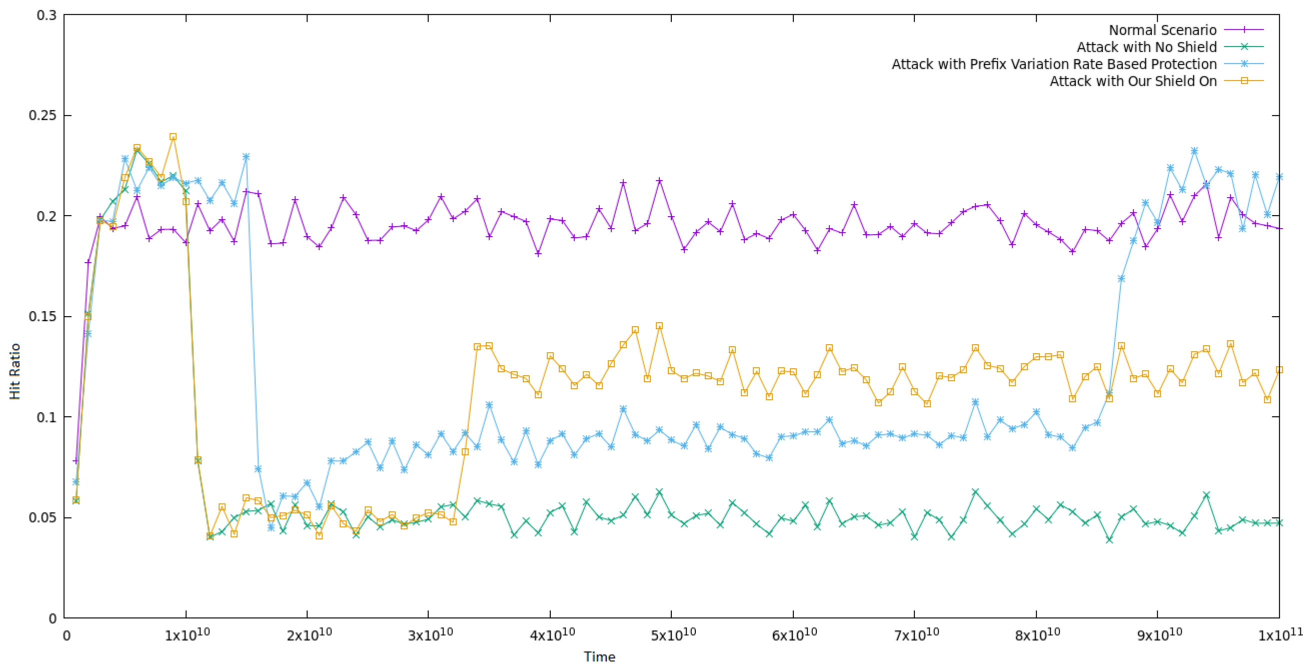


Figure 19: Comparison of Hit Ratios with Prefix Variation Rate Based Protection Method in DFN Topology

In Fig 19 , various scenarios on **DFN Topology** are depicted. It could easily be observed in the figure that the Hit ratio which is at the lower level of 0.08 with **Prefix variation based Approach** has been raised to the level of 0.15 with our Shield. Our method takes a little more time compared to the prefix variation based method, to train before actually being able to improve the hit ratios. However, it can improve the hit ratios by a larger factor later on.

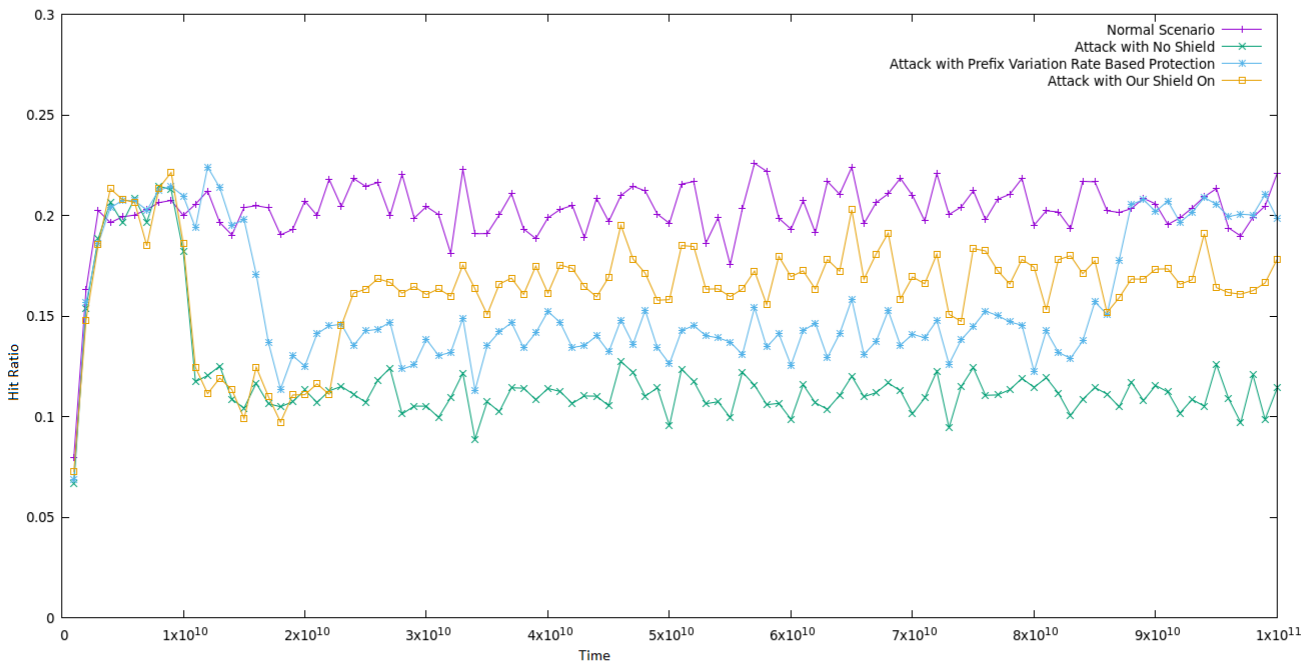


Figure 20: Comparison of Hit Ratios with Prefix Variation Rate Based Protection Method in XC Topology

In Fig 20, various scenarios on **XC Topology** are depicted. It could easily be observed in the figure that the Hit ratio which is at the lower level of 0.15 with **Prefix variation based Approach** has been raised to the level of 0.20 with our Shield.

5.2 Enhancing Cache Robustness: CacheShield

This method works best against locality disruption attacks. Upon content hit, the requested item is sent back immediately. However, when a miss occurs, the decision to cache the data is made by using a shielding function.

$$\psi(t) = \frac{1}{1 + e^{(p-t)/q}}, t = 1, 2, .. \tag{10}$$

Here, t denotes the t^{th} continuous miss for the specific content, whereas p and q are constants to be chosen according to the network. Upon experimentation we found the values $p = q = 50$ give the best results for our chosen topologies.

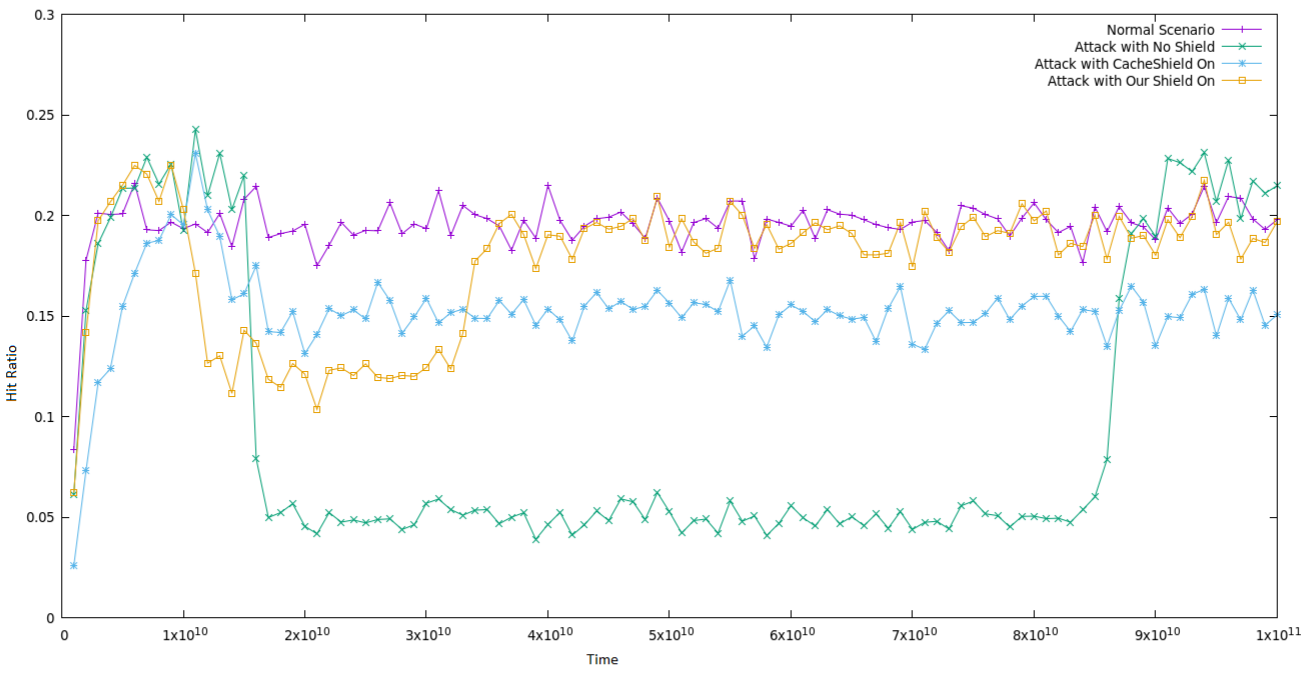


Figure 21: Comparison of Hit Ratios with CacheShield in DFN Topology

In Fig 21, various scenarios on **DFN Topology** are depicted. It could easily be observed in the figure that the Hit ratio which is at the lower level of 0.15 with **Enhancing Cache Robustness Method** has been raised to the level of 0.20 with our Shield.

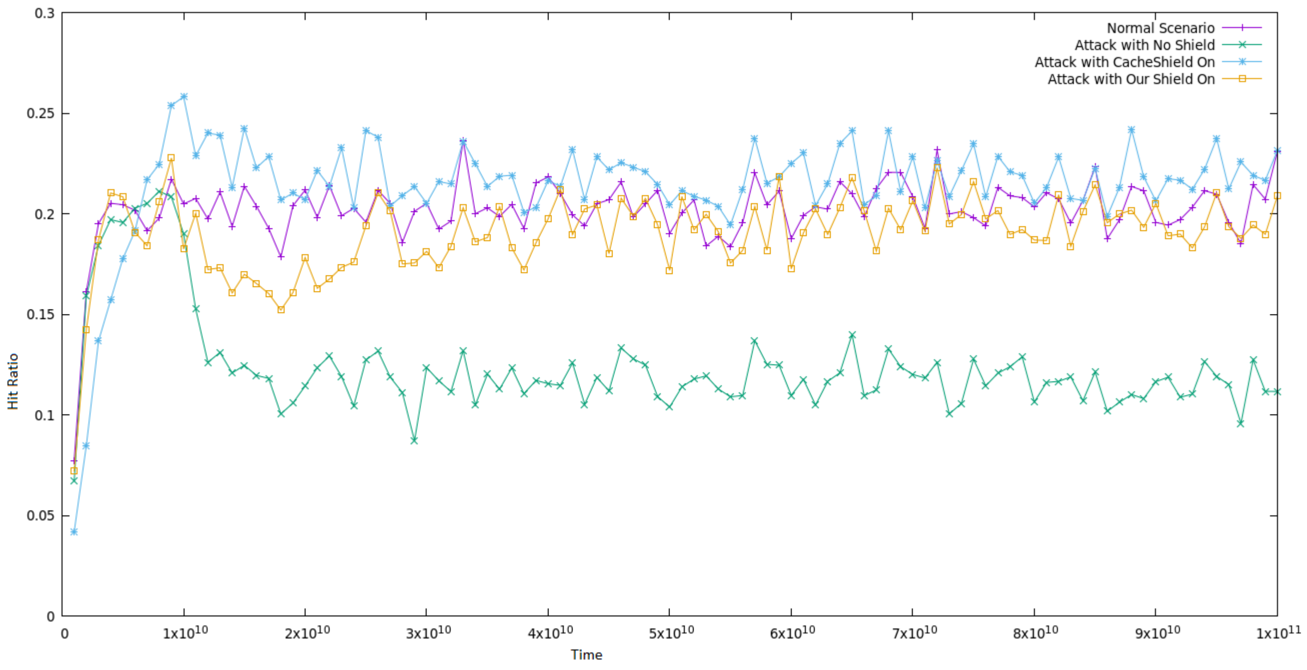


Figure 22: Comparison of Hit Ratios with CacheShield in XC Topology

In Fig 22, various scenarios on **XC Topology** are depicted. Here, we can observe that both approaches give almost similar results. Although CacheShield works slightly better by even improving the hit ratios beyond normal scenario. This is no surprise as CacheShield was developed using the XC topology itself. But, our approach works good nonetheless.

Chapter 6

Conclusion and Future Work

Finally, In this report, we have analysed two serious attacks which pose a threat to NDN.

We have shown that that Cache Pollution Attack is a realistic threat to NDN. The main contribution that we presented is the detection and mitigation of this attack using a new metric, the number of distinct users which helps us in categorizing the data as non-popular or popular, hence, mitigating the attack and we have used machine learning for offline detection of the attack on an interface. The main feature of our work is that we are considering the entire network as a unit rather than working on localized router locations. We have also implemented two already existing cache protection methods and compared the results of our approach with them.

The current design of our detection and mitigation method can be enhanced by integrating the offline detector with our proposed mitigation method. This could be further improved by designing a central controller based system that could help us by coordinating the routers against the attacks and hence mitigate them more efficiently.

We have shown how Interest Flooding Attack can hamper performance in NDN networks. More specifically, we analysed the performance measures in case of two advanced IFA techniques namely, bIFA and cIFA. We also implemented a popular countermeasure to IFA, Poseidon and showed how and why the currently used countermeasures are ineffective against the advanced IFA. Hence, new methods to detect and mitigate IFA are required.

Appendix A

Some Complex Proofs and Simple Results

An exponential moving average (EMA), also known as an exponentially weighted moving average (EWMA) is a type of infinite impulse response filter that applies weighting factors which decrease exponentially. The weighting for each older datum decreases exponentially, never reaching zero. The graph at right shows an example of the weight decrease.

The EMA for a series Y may be calculated recursively:

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha \cdot Y_t + (1 - \alpha) \cdot S_{t-1}, & t > 1 \end{cases} \tag{11}$$

Where:

The coefficient represents the degree of weighting decrease, a constant smoothing factor between 0 and 1. A higher discounts older observations faster. Yt is the value at a time period t. St is the value of the EMA at any time period t.

References

[1] Compagno, A., Conti, M., Gasti, P., and Tsudik, G. Poseidon: Mitigating interest flooding ddos attacks in named data networking. In *38th annual IEEE conference on local computer networks* (2013), IEEE, pp. 630–638.

[2] Conti, M., Gasti, P., and Teoli, M. A lightweight mechanism for detection of cache pollution attacks in named data networking. *Computer Networks 57*, 16 (2013), 3178–3191.

[3] contributors, W. Named data networking — Wikipedia, the free encyclopedia, 2018. [Online; accessed 22-April-2018].

[4] Hunter, J. S. The exponentially weighted moving average. *Journal of quality technology 18*, 4 (1986), 203–210.

[5] Kamimoto, T., Mori, K., Umeda, S., Ohata, Y., and Shigeno, H. Cache protection method based on prefix hierarchy for content-oriented network. In *Consumer Communications & Networking Conference (CCNC), 2016 13th IEEE Annual* (2016), IEEE, pp. 417–422.

[6] Karami, A., and Guerrero-Zapata, M. An anfis-based cache replacement method for mitigating cache pollution attacks in named data networking. *Computer Networks 80* (2015), 51–65.

[7] Salah, H., Alfatafta, M., SayedAhmed, S., and Strufe, T. Comon++: Preventing cache pollution in ndn efficiently and effectively. In *Local Computer Networks (LCN), 2017 IEEE 42nd Conference on* (2017), IEEE, pp. 43–51.

[8] Signorello, S., Marchal, S., François, J., Festor, O., and State, R. Advanced interest flooding attacks in named-data networking. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)* (2017), IEEE, pp. 1–10.

[9] Xie, M., Widjaja, I., and Wang, H. Enhancing cache robustness for content-centric networking. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE, pp. 2426–2434.

[10] Zhang, L., Estrin, D., Burke, J., Jacobson, V., Thornton, J. D., Smetters, D. K., Zhang, B., Tsudik, G., Massey, D., Papadopoulos, C., et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* (2010).