Interest Cash: An Application-based Countermeasure Against Interest Flooding for Dynamic Content in Named Data Networking

Zhaogeng Li
li-zg07@mails.tsinghua.edu.cn

Jun Bi
junbi@tsinghua.edu.cn

Institute for Network Sciences and Cyberspace, Tsinghua University
Department of Computer Science, Tsinghua University
Tsinghua National Laboratory for Information Science and Technology (TNList)

ABSTRACT

As a design of information-centric network architecture, Named Data Networking (NDN) provides content-based security. The signature binding the name with the content is the key point of content-based security in NDN. However, signing a content will introduce a significant computation overhead, especially for dynamically generated content. Adversaries can take advantages of such computation overhead to deplete the resources of the content provider. In this paper, we propose Interest Cash, an application-based countermeasure against Interest Flooding for dynamic content. Interest Cash requires a content consumer to solve a puzzle before it sends an Interest. The content consumer should provide a solution to this puzzle as cash to get the signing service from the content provider. The experiment shows that an adversary has to use more than 300 times computation resources of the content provider to commit a successful attack when Interest Cash is used.

Keywords

NDN, Interest Flooding, Interest Cash.

1. INTRODUCTION

As content distribution becomes the primary application of the Internet, a lot of Information Centric Network architectures are proposed. Named Data Networking (NDN [1], also called Content Centric Network, CCN [2]) is one of these ICN designs. In NDN, every piece of data is identified by a URL-like name. To get the data, the content consumer needs to send an Interest packet with the correct name. Any node which keeps the copy of the data can return the content when it receives the Interest.

A Data packet is signed by the content provider, which enables the content consumers to verify the provenance of the content (with the help of NDN trust management system [3]). However, the signing operation is computation-intensive. For dynamic generated content, signing is a burden for the content provider because every received Interest will trigger a signing operation. To test the efficiency of NDN signing, we implement a very simple NDN application based on NDNx (a prototype implementation of NDN). In this application, the content consumer sends an Interest with a name carrying a random string, and the content provider will echo the random string as content. We run this application on a host with Xeon X5650 CPU (with frequency of 2.66GHz). We find the content provider is able to give response to about 3000 Interests at most (with a single core) in a second. These 3000 Interests only consumes 1.53MBps bandwidth because a Data packet size is 463 Bytes and an Interest packet size is 46 Bytes in this experiment. Even if the Data packet size is 1500 Bytes ([4]), the bandwidth consumption is only 4.6MBps. The bottleneck is signing. In fact, we test the efficiency of RSA signing function provided by openssl, which is also used in the current version of NDNx. We find it takes 0.298ms for this CPU to complete the RSA signing. DSA algorithm is better but still takes 0.180ms.

An adversary can take advantages of this feature to deplete the computation resources of content providers. It will be Denial-of-Service attack since the content provider can't provide content service normally under this attack. For convenience, we call this kind of attack *Signing-DoS attack*. One countermeasure against *Signing-DoS attack* is to disable signing for dynamic content, which is obviously not a good idea because the content can be spoofed easily.

In this paper, we propose Interest Cash, an application-based countermeasure to mitigate *Signing-DoS attack*. To the best of our knowledge, this is the first work on how to defend signing-DoS attack in NDN. The key idea is that the content consumer has to do extra computation before the content provider serves it. This is also called proof-of-work. The extra work is to solve a puzzle. The content consumer has to append the solution to this puzzle as cash to the content name in Interest. The content provider only signs the content for the content consumer who offers a valid cash. Since we increase the cost to send a valid Interest, we can prevent potential adversaries from sending too many Interests for dynamic content. At the same time, normal users are hardly affected because the extra computation is not a problem if they do not send Interests too fast. The key challenge of Interest Cash is to generate puzzle for each Interest

CFI'14, June 18 - 20 2014, Tokyo, Japan

http://dx.doi.org/10.1145/2619287.2619298.

with low overhead. The experiment shows that an adversary has to use more than 300 times computation resources of the content provider to commit a successful attack when Interest Cash is used.

The rest of this paper is organized as follows: In section 2, we introduce some background of NDN and proof-of-work. We describe Interest Cash in detail in section 3, including the basic idea and some advanced design. Evaluation of Interest Cash is in section 4. We make some discussion about why Interest Cash is useful in section 5. We conclude this paper in section 6.

2. BACKGROUND

2.1 Named Data Networking

There are two types of packets in NDN: Interest and Data. To get the desired content, one needs to send an Interest with the content name to the NDN network. The NDN network will forward the Interest to the content provider according to the content name, instead of the destination address. When the content provider receives the Interest, it can return the corresponding Data packet which contains the desired content. The Data can be static or dynamic generated. The Data is forwarded back to the content consumer along the reverse path of the Interest, with the help of Pending Interest Table (PIT) in the intermediate nodes. PIT can be considered as a soft state table which stores information about every passing Interest and its incoming face. The Data can be cached in any intermediate node. The cached copy of Data in any node can be returned directly if the node receives another Interest with the same name. Content-based security is one of several advantages of NDN. The Data (contains the content name and the content itself) can be signed by the content provider. The content consumer can verify the signature using the public key provided in NDN trust management system[3].

Denial of Service (DoS) attack is a noteworthy problem of NDN. NDN can resist some current common types of DoS attack easily because an adversary can hardly send too much meaningless traffic to the victim (the data delivery is driven by content consumers). However, there are some new forms of DoS attack in NDN. Interest Flooding is one of the new forms which deserves more attention[5]. An adversary can send a huge number of Interests. If these Interests ask for nonexistent content, the PIT of NDN router may be filled with entries which will not be deleted normally (have to wait for timeout). Some countermeasures have been proposed to cope with this problem ([6], [7], [8], [9] [10]). If these Interests ask for dynamic generated content, the content provider has to spend considerable time on serving these Interests, especially signing the Data. One countermeasure against this attack is to verify the source of every Interest, and filter some Interests from malicious source, but the consumer verification is not available anytime. In fact, Interest Flooding for dynamic generated content hasn't been well studied. In this paper, we propose a countermeasure against this kind of attack.

2.2 Proof-of-work

Proof-of-work is widely used for defending DoS Attack ([11]). Proof-of-work requires a client to do extra computation to prove it is a normal user. To get a service from a server, a client has to solve a puzzle first. The client can get the service from the server only when the client provides the correct solution. All the puzzle patterns share a feature that it is difficult to find the correct solution, but it is very easy to verify whether the solution is correct or not. Reverse hash (to find a string whose hash value meets some specific requirements) is the most common puzzle pattern.

The philosophy of proof-of-work is to increase the cost to get a service. A normal user can hardly be aware of this protection mechanism, but the heavy burden to solve the puzzles is unacceptable for an attacker. It is almost impossible for one host to generate too many valid requests. Attackers can still commit a distributed attack if they have control of a large number of hosts. However, as the total number of requests is slashed, the severity of distributed attack will be also mitigated greatly.

3. INTEREST CASH

3.1 Protocol Overview

Interest Cash employs the concept of proof-of-work. Before getting the signing Data, the content consumer needs to solve a puzzle. The puzzle pattern is reverse hash similar to client puzzle. However, if the puzzle is generated by the content provider for each Interest like client puzzle, the dynamic puzzle itself will become the target of attack. Based on this consideration, the puzzle in Interest Cash should be different from client puzzle. In Interest Cash, the content provider will prepare a static meta-puzzle. Every content consumer will get this meta-puzzle to generate the final puzzle for the Interest they will send. Inspired by Hashcash [12], we call the solution of the final puzzle Interest cash. In the rest of the paper, we use the word cash for abbreviation. The literal meaning of cash is that the content consumer has to buy signing service from the content provider with the solution.

Interest Cash is application-based. Fig 1 depicts the interaction between the content consumer and the content provider when the application enables Interest Cash. The meta-puzzle is composed of a timestamp t, a random string x, a solution length l (bits), a puzzle difficulty factor k ($l > k$) and an expiration time T. After obtaining the meta-puzzle ($/prefix/puzzle$), the content consumer can generate the final puzzle for the Interest with the original name n ($/prefix/content\_1/$). The final puzzle for the content consumer is to find the *valid cash* according to the following definition (Here, + denotes concatenation of string, and $hash$ is a one-way hash function, such as MD4, MD5 or SHA-1.).

DEFINITION 1 (VALID CASH). *For a meta-puzzle (t, x, l, k, T), the valid cash for the content with name n is an l-bit string s, which makes the first k bits of string $hash(n + t + x + s)$ all 0.*

After calculating the valid cash s, the content consumer should append s to the name n and send the Interest with reconstructed name to the content provider. The reconstructed Interest is required to carry the timestamp t as well as s ($/prefix/content\_1/t/cash\_1$). Since n, t, x, s and k are all known by the content provider, the correctness of s can be easily verified. If t in the name is different from the timestamp in current meta-puzzle kept in the content provider, the cash will be considered stale or fake directly without verification. The content consumer can save this meta-puzzle.

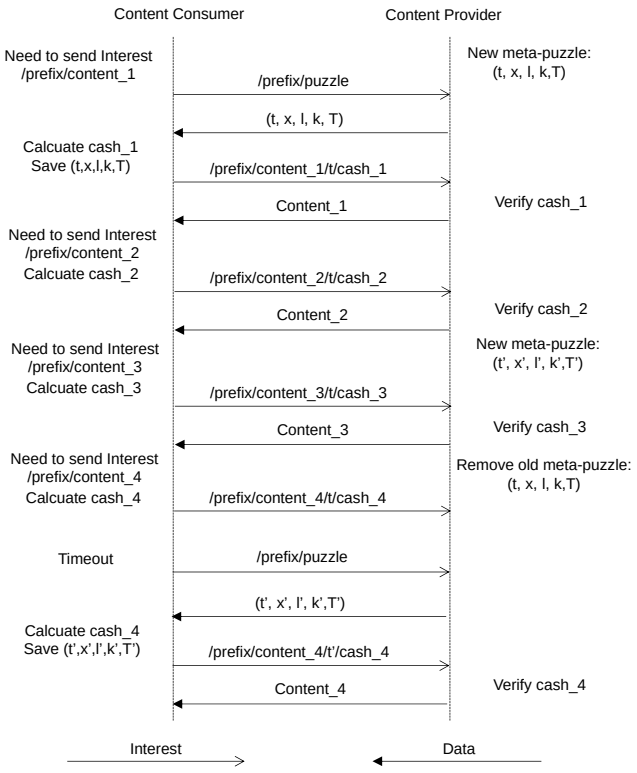

Figure 1: Interest Cash Protocol Overview.

When the content consumer needs to send another Interest, it can use this saved meta-puzzle to calculate cash directly ($/prefix/content\_2$).

The meta-puzzle should be regenerated after a period of time, to avoid potential attacks. When a new meta-puzzle is generated, old meta-puzzle should be kept for a while (for example, 10s) as some Interest using old meta-puzzle may reach during this time. Before the old meta-puzzle is removed, t matching the timestamp in either the old one or the new one is considered valid ($/prefix/content\_3/t/cash\_3$). If the old meta-puzzle has been removed, t matching the timestamp in the old meta-puzzle will be considered invalid ($/prefix/content\_4/t/cash\_4$). The content provider will not answer the invalid Interest. After the content consumer realizes the Interest has timed out, it will redirect an Interest for meta-puzzle again. A new cash will be calculated when the content consumer obtained the new meta-puzzle.

The expiration time T in the meta-puzzle helps the content consumer to decide whether a meta-puzzle it got before is expired or not. If T is a past time, the content consumer should try to find the new meta-puzzle. However, if T is a future time, the meta-puzzle kept in the content consumer may still be out of date. The reason is that the content provider may update the meta-puzzle before T has come if it receives too many valid Interests. Since T is only an auxiliary value, the time synchronization between the content consumer and the content provider is not important.

3.2 Security Analysis

To find correct s, the content consumer has to try many different strings with brute-force. One needs to execute the hash function for average 2^{k-1} times to find the solution. It is k which decides the difficulty to solve the puzzle. However, the content provider needs to execute the hash function only once to verify whether the solution is right or not. If it is a right solution, the provider will return the signed content; Otherwise the provider will not answer the Interest. Note that there may be more than one solutions for one puzzle. The average number of solutions is 2^{l-k}. The valid cash could be any of these solutions.

For each invalid Interest, the content provider will answer nothing. This reaction will turn the dynamic content into nonexistent content. The number of PIT entry in the NDN routers will be increased since the entries for invalid Interests have to wait for timeout. Although answering nothing for invalid Interest has some disadvantages, we think it is the best option, because: 1) The content provider cannot return signed warning, because the warning is a dynamic content which will become the target of attack. 2) Unsigned warning may be spoofed so that the content consumer cannot ensure whether the received warning is generated by the content provider or not. 3)There already have been some methods to cope with the Interest Flooding for nonexistent content just as mentioned in section 2.1.

Now we analyze what an adversary can do if the application enables Interest Cash.

1. As the meta-puzzle is a static content so that the puzzle content itself will not suffer *Signing-DoS attack*.

2. Repeatedly sending Interest with the same name (carrying the same cash) will not affect the content provider, because repeated Interests will be answered by cache in NDN.

3. An adversary can send randomly generated cash to force the provider completing the verification. Since the verification consumes very few resources, this attack can hardly affect the content provider. What's more, when the network enabled the mechanism to limit Interest Flooding for nonexistent content, one can never send such a huge number of Interests in a short time. On the other hand, the probability to generate valid cash in this way is only $2^{l-k}/2^l = 2^{-k}$, which can be ignored if k is large enough. For example, when $k = 20$, the probability to generate valid cash in this way is only about 10^{-6}. In other words, one needs to send about 10^6 Interests to get a valid Interest.

4. To commit a successful attack, an adversary has to use hundreds of times (distributed) resources to calculate valid cash. As a result, Interest Cash can increase the cost for an adversary to commit an attack. To make it clear, we give the following quantitative analysis. Let t_k denote the mean value of calculation time for a content consumer to generate a valid cash when the difficulty is k. As we know $t_k = 2 \times t_{k-1}$, we have

$$t_k = 2^{k-i} \times t_i \quad (i = 1, 2, 3, ...). \tag{1}$$

Let t_v denote the verification time for the content provider to verify the cash and t_s denote the signing time for the content provider to sign a content. To deplete the computation resources of the content provider with m hosts (CPU cores), the adversary should use M hosts (CPU cores). Then we have

$$\frac{M}{t_{k_u}} \geq \frac{m}{t_v + t_s}. \tag{2}$$

Here k_u denote the upper limit of k. We will discuss this later. According to (1) and (2), we have

$$M \geq \frac{2^{k_u - i} \times t_i m}{t_v + t_s} = Cm \times 2^{k_u - i}. \quad (3)$$

Here C can be determined by experiment in reality (consider the computation resources of the adversary and the content provider).

3.3 Meta-puzzle Update

As mentioned above, the meta-puzzle should be regenerated after a period of time. The content provider should know when to update and how to update. If anything is going well, the meta-puzzle will be regenerated when expiration time T comes. However, if the content provider receives too many or too few valid Interest, it needs to regenerated the meta-puzzle in advance to change the puzzle difficulty k and the random string x.

Algorithm 1 k update when a new time slot comes.

Require: k, n in last time slot, a counter i, and N, k_l, k_u.
Ensure: k' in current time slot.

```
if n ≥ N then
    k' ⇐ k_u, i ⇐ 0
else if 0.8N ≤ n < N then
    i ⇐ 0
    if k = 0 then
        k' ⇐ k_l
    else if k < k_u then
        k' ⇐ k + 1
    else
        k' ⇐ k
    end if
else if n < 0.4N then
    if i < 3 then
        i ⇐ i + 1
    else if k = k_l then
        k' ⇐ 0, i ⇐ 0
    else if k > 0 then
        k' ⇐ k − 1, i ⇐ 0
    else
        k' ⇐ k
    end if
else
    k' ⇐ k, i ⇐ 0
end if
```

We divide time into different time slots first. The content provider will count the number of valid Interests it receives in each time slot (for example, $30s$), which we use n to denote. The meta-puzzle will be updated when one time slot ends if the content provider thinks there is a necessity. To define necessity, the provider needs to know how many valid Interests it can serve in one time slot at most. We use N to denote the maximum number.

Algorithm 1 depicts the update procedure in pseudo code (when the expiration time of meta-puzzle has not come). In order to avoid making the cash calculation takes too long, we should set an upper limit for difficulty (k_u). When $n > N$, meaning there are too many valid Interests, the puzzle difficulty k' should be increased to k_u. When $0.8N < n < N$, meaning there will be too many valid Interests, the puzzle difficulty k' should be increased by 1. When $n < 0.4N$ in three consecutive time slots, meaning there are too few valid Interests, the puzzle difficulty k' should be decreased by 1. The reason why we do not decrease k when $n < 0.4N$ in one time slot immediately is that we should avoid oscillation of difficulty. When k is too small, the cash calculation makes no sense. We need to set a lower limit for the difficulty (k_l), too. If there are still too few valid Interests when $k = k_l$, we set $k' = 0$ directly. If there will be too many Interests when $k = 0$, we set $k' = k_l$. If $k' = k$, the meta-puzzle will not be regenerated.

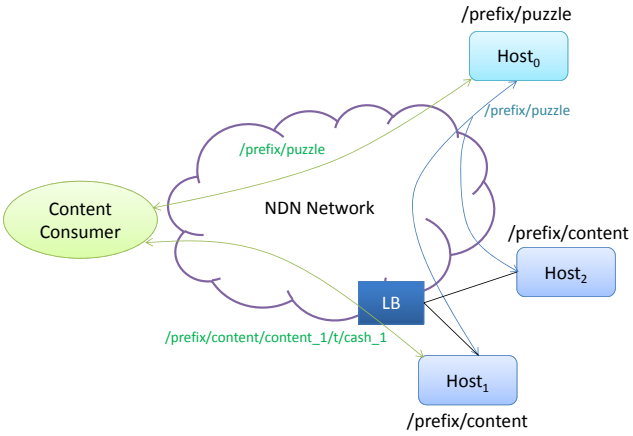

Figure 2: Puzzle Separation. The puzzle module is separated to an independent host $Host_0$. $Host_1$ and $Host_2$ both get the puzzle from $Host_0$ as well as content consumers. LB is a load balancer which can assign user Interests evenly between $Host_1$ and $Host_2$.

3.4 Puzzle Separation

As Data signing consumes many computational resources, a single host can only serve a relatively small number of Interests for dynamic content during a fixed period of time. If we want to serve further more Interests, we have to prepare more hosts as the content provider. These hosts work independently. Any of them receives the Interest and replies with a Data. All the hosts can share one puzzle module. In other words, the puzzle module can be extracted into a separate application.

Fig 2 depicts the separation of puzzle module. There are three hosts belonging to the content provider. $Host_0$ acts as the puzzle module, which announce the name prefix: $/prefix/puzzle/$. $Host_1$ and $Host_2$ both act as the main application module, which announce the name prefix: $/prefix/content$ through load balancer LB. $Host_0$ generates the meta-puzzle. $Host_1$ and $Host_2$ get the newest meta-puzzle from $Host_0$ every a few seconds. All the content consumers get the meta-puzzle from $Host_0$ before they want to send an Interest to $Host_1$ or $Host_2$ (LB decides which host will serve for the Interest). In this way, either $Host_1$ and $Host_2$ can verify the validity of cash provided by the content consumers.

If the content provider employs puzzle separation, the meta-puzzle update will be more complex. That means the puzzle module should know n and N of any host. For example, in Fig 2, $Host_0$ should ask for the number of valid

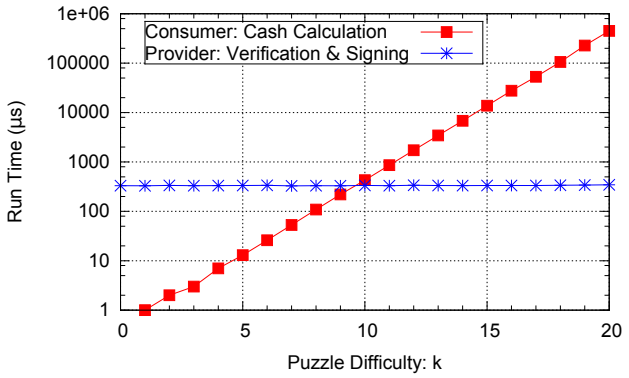

Figure 3: Average run time of calculating cash for the content consumer and verification and signing for the content provider.

Interests received by $Host_1$ and $Host_2$ (n_1 and n_2) periodically. The maximum number of valid Interests can be served by $Host_1$ and $Host_2$ (N_1 and N_2) is configured in advance. $Host_0$ should decide the new difficulty k according to $(n_1 + n_2)/(N_1 + N_2)$. This mechanism works well when the LB can assign Interests evenly between $Host_1$ and $Host_2$.

4. EVALUATION

In this section, we show our experiment results to prove that Interest Cash can mitigate *Signing-DoS Attack* considerably at a low cost.

4.1 Experiment Setup

We consider a very simple NDN application which provides dynamic content mentioned in section 1. In this application, one content consumer sends an Interest with a name carrying a random string, and the content provider will echo the random string as content to the user. The application is built over NDNx. We implement two instances of this application: without Interest Cash and Interest Cash enabled. For Interest Cash, we set $l = 128$, and we use MD5 as the hash function. We use two hosts (both have two Xeon X5650 CPUs[1]) to act as the content provider (Host P) and content consumers (Host C). The two hosts are both connected to the same switch with a 1Gbps link. At NDN layer, Host C is connected to Host P with UDP protocol directly.

4.2 Effectiveness of Interest Cash

Fig 3 shows that the average run time of calculating cash for the content consumer and verification and signing operation for the content provider. We get the data by calculating the arithmetic mean value of 10000 Interests. Note that the run time when $k = 0$ indicates the situation when Interest Cash is not enabled (there is neither cash calculation nor cash verification). The run time for content consumer increases exponentially with the puzzle difficulty, which proves the theoretic deduction that the average run time of calculating cash is 2^{k-1}. The run time for content provider is composed of the run time for cash verification and Data signing.

Interest Cash can limit the number of valid Interests sent by a content consumer. In the experiment, the content consumer needs to take about $100ms$ to calculate a valid cash when $k = 18$. That means the content consumer can only generate 10 valid Interests at most per second. However, if Interest Cash is not enabled, the content consumer (host C) can send millions of valid Interests per second. Interest Cash can reduce the number the valid Interests generated by one potential adversary to 10 while increase only $100ms$ extra delay for normal user when $k = 18$. As Host P can sign about 3000 Data packet per second, there should be about 300 hosts like Host C to overwhelm Host P when $k = 18$. Fig 3 also shows that the run time for the content provider remains unchanged relatively because the run time for signing is static (about $330\mu s$), and the run time for cash verification is too short no matter what k is (all shorter than $1\mu s$). That means the overhead of cash verification is very low.

Generally, imagine that both content provider and the adversary use the same hosts with us. Then with the help of equation (3), when $i = 18$, $t_i \approx 100ms$, $t_v < 1\mu s$ and $t_s \approx 330\mu s$, we know $M \geq 300m \times 2^{k_u - 18}$. If we set $k_u \geq 18$, the adversary needs at least 300 times computation resources of the content provider to commit a successful attack. However, without Interest Cash, the adversary can commit an successful attack just with not more compute resources than the hosts of the content provider. Note that the factor 300 (factor C in equation (3)) here comes from the hosts used in our experiment. Different hosts will lead different factors. No matter what the factor is in reality, we believe it has a same order of magnitude.

4.3 Attack Defending Example

Now we will show the behavior when the content provider is under *Signing-DoS Attack*. We simulate an attack through starting three adversary processes on Host C. As proved above, sending invalid Interests will not affect the content provider. As a result, all these processes flood as many valid Interests as possible to Host P (at most 10000 Interests per second). Note that the processes work without interference because one process will run on one core of the CPU. Therefore, the adversary has three times computation resources of the content provider. In the experiment, the maximum of valid Interests number the content provider can serve is set to 2800 per second. The freshness of the meta-puzzle Data is set to $10s$. The time slot of the content provider to update the meta-puzzle is $30s$ ($N = 2800 \times 30 = 84000$). The old meta-puzzle will be used for another $10s$ after a new meta-puzzle is generated. We set $k_u = 20$ and $k_l = 10$. The adversary tries to get the new meta-puzzle every $15s$ to avoid the content provider changes the meta-puzzle.

Fig 4 shows the change of puzzle difficulty and number of Data signed in a time slot during the attack. The value at time t (multiple of $30s$) indicates the value in the time slot $[t - 30, t]$. At first, the puzzle difficulty is 0. During the first $30s$, the adversary sends 900000 valid Interests to Host P in total (each process sends 10000 Interests per second). Host P signs only about 86000 Data during this time. Note that not all the Interests sent are served because the content provider is overwhelmed. Then Host P generates a new meta-puzzle with puzzle difficulty $k = 20$ because $n > N$. After the new meta-puzzle is used, the adversary can only send about a hundred of valid Interests in one time slot. As received Interests are too few ($n < 0.4N = 33600$) in consecutive three time slot, Host P decrease k by 1 so

[1]This kind of CPU has 6-cores with frequency of 2.66GHz.

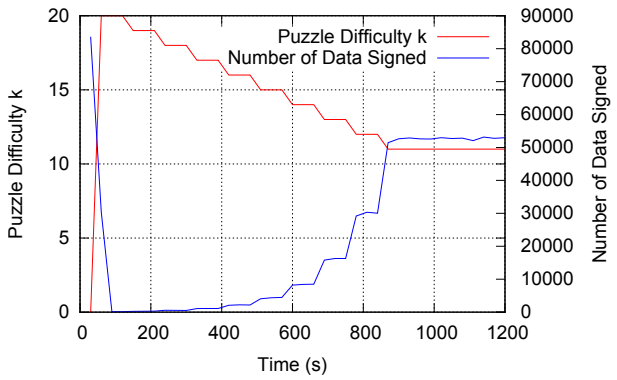

Figure 4: Puzzle difficulty and number of Data signed in each time slot of the content provider during the simulated attack.

that the calculation time can be reduced for normal users. That is why we can see the stepped increase of the number of Data signed and the stepped decrease of k. k will be decreased gradually because the adversary cannot generate enough valid Interests. k is stable when $k = 11$. At this time, the number of received valid Interests is about 52000. The content provider can serve for other content consumers normally (at least serve another 32000 Interests per time slot). In other words, the attack fails. If we do not use Interest Cash, the content provider is overwhelmed by the flooding Interest sent from the adversary.

5. DISCUSSION

Interest Cash is just one approach to defend signing-DoS attack. Sometimes, the content provider may choose not to sign the content directly if the content provenance is not important. Interest Cash is designed for those content providers who cares about the provenance of the their content and want to avoid man-in-the-middle attack. Search engine is a good user case of Interest Cash. If the signing is disabled, intermediate entities may generate Data by themselves. Spoofed Content may contains phishing pages, which mislead the users. Therefore, we believe Interest Cash is useful.

As a kind of proof-of-work, Interest Cash only tries to increase the cost to send a valid Interest, so that an adversary needs to use much more resources. As the experiment shows, the adversary needs at least 300 times computation resources of the content provider to commit a successful attack. This result may be controversial because in current Internet, one adversary can easily take control of a botnet which has thousands, even millions of innocent hosts. However, we argue that Interest Cash is still useful. Cloud computing today makes resources easy to extend. For a content provider in cloud, if it is under a DoS attack, it can rent more resources to resist the attack. With the help of Interest Cash, the content provider can rent much less resources than the case without Interest Cash because malicious valid requests are reduced considerably, which will also reduce the operating costs of the content provider.

Previous research ([13]) points out that reverse hash pattern is not effective if the adversary employs GPU to solve puzzles, because of the solution can be solved in parallel. We will study this problem further in our future work.

6. CONCLUSION

In this paper, we propose Interest Cash, an application-based countermeasure against Interest Flooding for dynamic content. Interest Cash requires the content consumer to provide a solution to a puzzle as cash to get the signing service from the content provider. The puzzle is generated with the help of static meta-puzzle provided by the content provider. The difficulty of puzzle can be adaptive. The experiment shows that an adversary has to use more than 300 times computation resources of the content provider to commit a successful attack when Interest Cash is used.

7. ACKNOWLEDGEMENT

This work is supported by the National High-tech R&D Program ("863" Program) of China (No.2013AA010605), and National Science & Technology Pillar Program of China (No.2012BAH01B01). Jun Bi is the corresponding author.

8. REFERENCES

[1] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, D. Smetters, B. Zhang, et al. Named data networking (NDN) project. *NDN Technical Report NDN-0001*, 2010.

[2] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. In *CoNEXT*. ACM, 2009.

[3] C. Bian, Z. Zhu, E. Uzun, and L. Zhang. Deploying key management in ndn testbed. *NDN Technical Report NDN-0009*, 2013.

[4] D. Perino and M. Varvello. A reality check for content centric networking. In *ACM ICN Workshop*, 2011.

[5] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang. DoS & DDoS in named-data networking. In *ICCCN*. IEEE, 2012.

[6] A. Compagno, M. Conti, P. Gasti, and G. Tsudik. Poseidon: Mitigating interest flooding DDoS attacks in named data networking. In *LCN*. IEEE, 2013.

[7] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang. Interest flooding attack and countermeasures in named data networking. In *Networking*. IEEE, 2013.

[8] J. Tang, Z. Zhang, Y. Liu, and H. Zhang. Identifying interest flooding in named data networking. In *GreenCom*. IEEE, 2013.

[9] K. Wang, H. Zhou, H. Luo, J. Guan, Y. Qin, and H. Zhang. Detecting and mitigating interest flooding attacks in content-centric network. *Security and Communication Networks*, 2013.

[10] H. Dai, Y. Wang, J. Fan, and B. Liu. Mitigate DDoS attacks in NDN by interest traceback. In *IEEE NOMEN Workshop*, 2013.

[11] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.

[12] A. Back. Hashcash-a denial of service counter-measure. `http://www.hashcash.org/`, 2002.

[13] J. Green, J. Juen, O. Fatemieh, R. Shankesi, D. Jin, and C. Gunter. Reconstructing hash reversal based proof of work schemes. In *LEET*. USENIX, 2011.