Scalable Pending Interest Table Design: From Principles to Practice

Haowei Yuan and Patrick Crowley
Computer Science and Engineering
Washington University
St. Louis, Missouri 63130
{hyuan, pcrowley}@wustl.edu

Abstract—**A Pending Interest Table (PIT) is a core component in Named Data Networking. Scalable PIT design is challenging because it requires per-packet updates, and the names stored in the PIT are long, requiring more memory. As the line speed keeps increasing, e.g., 100 Gbps, traditional hash-table based methods cannot meet these requirements. In this paper, we propose a novel Pending Interest Table design that guarantees packet delivery with a compact and approximate storage representation. To achieve this, the PIT stores fixed-length fingerprints instead of name strings. To overcome the classical fingerprint collision problem, the Interest aggregation feature in the core routers is relaxed. The memory requirement and network traffic overhead are analyzed, and the performance of a software implementation of the proposed design is measured. Our results show that 37 MiB to 245 MiB are required at 100 Gbps, so that the PIT can fit into SRAM or RLDRAM chips.**

I. Introduction

Named Data Networking (NDN) [1] is a recently proposed network architecture that follows the content-centric approach [2], which focuses on the content itself rather than where the content is from. There are two types of packets in NDN, Interest packets and Data packets. As in other content-centric proposals, NDN packets have globally unique names, so that Data packets can be fetched, cached, and redistributed in the network. In NDN, packets are forwarded based on name lookup results. In the forwarding process, the Pending Interest Table (PIT) is one of the core components.

The Pending Interest Table keeps track of the currently unsatisfied Interest packets. Arriving Interest packets are forwarded to the next hop based on a Forwarding Information Base (FIB) lookup only if the PIT finds no pending Interest packet with the same name. The PIT also stores the destination information for Data packets. For each Data packet, the PIT is queried to find the incoming face(s) that requested the content, and then the Data packet will be delivered and its content name is deleted from the PIT. Hence, the PIT requires per-packet updates, including memory writes. At 100 Gbps, a 100-byte packet could have an arrival rate of eight nanoseconds. In the meantime, the PIT memory size becomes larger as the link speed increases, which makes space-limited high speed memory devices infeasible to use. In each PIT entry, the content name needs to be stored. The names are similar to URLs, and today's URLs typically require tens of bytes of storage. For example, the URLs for the pictures and videos on popular social networking websites, which include long hash numbers, are more than 80 bytes long. Moreover, there are websites that include article names in the URLs, making the URLs longer. As a result, designing a fast and scalable Pending Interest Table is challenging.

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

Our approach is to reduce the PIT memory size for core routers, so that fast memory chips, such as SRAM or RL-DRAM, can be employed to support per-packet updates. Uniquely, our design guarantees packet delivery with a compact and approximate storage representation. We propose a network-wide solution to the scalable Pending Interest Table problem. In our design, we classify network routers as either *core* routers or *edge* routers, just as in today's Internet. The edge routers function as usual, but the core routers store a fingerprint instead of the full name string for each Interest packet. As a result, the core router PIT memory requirement is greatly reduced. However, fingerprint collision, a classical problem, arises. To guarantee packet delivery, we relax the Interest aggregation requirement in core routers, i.e., every Interest packet received by the core router PITs will be forwarded. To provide enough time for potential multiple Data packets to arrive, a colliding fingerprint entry will not be deleted until it has expired. Another problem is that the proposed system cannot differentiate fingerprint collisions from duplicate Interest requests. Thus, for duplicate fingerprints, we leverage the idea that most Interest aggregation happens at the edge routers, and also use the Content Store in the core routers to help prevent receiving duplicate Interest requests. The proposed design introduces additional network traffic, but it will be dropped by the edge routers. Hence, the entire packet processing procedure is transparent to the users and content providers. In this paper, we make the following contributions:

1) We propose a PIT design that takes advantage of a compact storage representation and the edge router filtering effect, so that the memory requirement of the Pending Interest Table in core routers is reduced significantly.
2) We use analytical modeling to analyze the memory requirement and demonstrate the network traffic overhead is acceptable. Our results show that 36.77 MiB to 244.44 MiB are required at 100 Gbps.
3) We have implemented the PIT design in software, and

measured the fingerprint collision rate and update latency. At 1 Gbps, the measured latency is 1.2μs.

II. Background

In this section, we review the functionality and design principles of the Pending Interest Table, and then consider general hash-based techniques.

A. The Pending Interest Table and Its Design Principles

The Pending Interest Table provides two major functions in the NDN architecture, namely Interest packet aggregation and Data packet multicast. Each incoming Interest name is looked up in the PIT, and duplicate Interest requests are aggregated, i.e., the Interest packet will be forwarded only if its name is not found in the PIT. The PIT keeps track of which face has requested what content, and each PIT entry stores a list of its Interest incoming faces. When a Data packet arrives, the PIT is queried to fetch all the outgoing faces, and then the Data packet is delivered. The PIT design has been recognized as a flow table management problem [3]. In IP networks, generally a five-tuple rule is used to define a flow. Likewise, we could define each NDN flow using a content name. Each flow has its expiration time and a list of incoming faces. NDN packets have hierarchically structured names, while in this paper we use exact string matching for PIT name lookup. While our approach provides the essential functionality, it is worth noting that the PIT lookup in the NDN reference implementation has additional features [4]. It is also suggested in [3] that the core and edge routers should have different features. Indeed, our proposed design leverages the differences between these two types of routers. The detailed differences between the edge and core routers are presented in Section III.

B. Hash-Based Techniques

Hash tables have been studied extensively in the past decade for high-speed packet processing [5]. Hash table designs generally aim at a constant number of off-chip DRAM references. Previous studies [6][7][8] show that storing filters on a small SRAM chip greatly reduces the number of DRAM accesses. The filters stored in SRAM are typically Bloom filters, or counting Bloom filters. It has also been proven that storing fingerprints in a hash table provides the same filtering function [9]. However, applying these designs directly for PIT still requires at least one DRAM access to write the content name and other information for each insertion. Since a long content name exceeds the DRAM bus capacity, each PIT entry access requires multiple cycles, or can be optimized as one or two burst accesses. Our approach seeks to eliminate the DRAM accesses. Hash tables are preferred to Bloom filters since the expiration time and the face list can be easily stored.

III. Pending Interest Table Requirements

In this section, we discuss the differences between edge and core routers, and then highlight their requirements.

Edge routers connect consumers to ISP networks, and therefore the numbers of interfaces on edge routers are typically large, reaching 64 thousand [10]. The throughput is not high for edge routers, so in this paper, we select 1 Gbps as the link rate. Network traffic aggregates at edge routers and then enters the backbone networks. Core routers are deployed in backbone networks, where the throughput rather than the number of interfaces is the primary concern. A high-end router may contain multiple line cards, and therefore its bandwidth can reach 10 Gbps, 100 Gbps, or more. In this case, the number of entries in the PIT is large, and the PIT needs to be updated efficiently. Generally, there are not as many features on core routers as on edge routers, and the number of faces is small, ranging from a few interfaces to tens of interfaces [11].

TABLE I
Pending Interest Table Requirements

Line Rates	Edge(1 G)	Core(10 G)	Core(100 G)
Interfaces	thousands	hundreds	tens
Best Case	0.156 Mpps	1.563 Mpps	15.625 Mpps
Worst Case	1.25 Mpps	12.5 Mpps	125 Mpps
Best Mem	0.625 MiB	6.25 MiB	62.5 MiB
Worst Mem	5 MiB	50 MiB	500 MiB

The number of packets arriving at each face is also affected by the packet sizes. Since NDN can run on top of Ethernet directly, the packet size could be as small as 64 bytes, or as large as 1500 bytes. Generally, Interest packets are relatively small, and Data packets are large. In a flow balance mode, one Data packet is transferred for one Interest packet. The best case can be configured as 100-byte Interest packets and 1500-byte Data packets. In the worst case, the Data packets can be as small as the Interest packets; therefore, we set 100 bytes for both of them. Table I lists both the operation frequency and memory requirements, assuming the round trip time T_{rtt} = 80 ms [10], S_I = 100 Bytes, the best case S_D = 1500 Bytes, and the worst case S_D = 100 Bytes. We are not considering the case where most of the Interest packets cannot be satisfied and have to wait for expiration, because this behavior could happen only when the PIT is under a flooding attack. PIT security issues are discussed in Section VIII. From Table I, for low-end routers, the memory size in the worst case can be easily fit into SRAM. Even for the 10 Gbps case, it is possible to fit them into RLDRAM. However, for the 100 Gbps case, its memory size cannot be fit into RLDRAM, thus DRAM has to be used. It is worth noting that we are designing the PIT for the worst case (i.e., $S_D = 100$ bytes), since we believe the PIT should be capable of handling the worst case traffic. Moreover, even under other operation modes, our design could still save considerable memory space.

IV. Fingerprint-only Pending Interest Table

In this section, we present the proposed fingerprint-only Pending Interest Table design in detail.

A. Design Overview

Our design is based on the ideas that storing fingerprints saves memory space, and that edge routers can aggregate

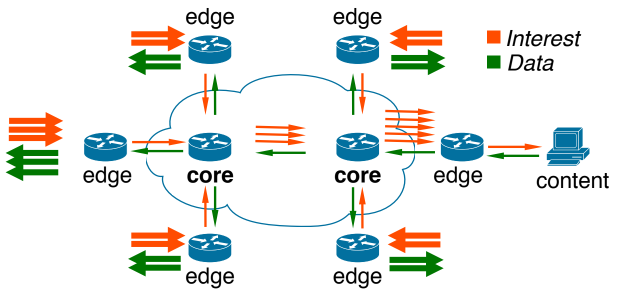


Fig. 1. System Design

most of the duplicate Interest packets. Thus, our system-wide solution to scalable PIT can relax the Interest aggregation requirement for the core routers. Figure 1 shows the system design. A wider arrow denotes a larger number of packets. In the figure, Interest packets are aggregated at the edge routers and then enter the core network. The core routers simply forward all the received Interest packets. Eventually, duplicate core Interest packets are aggregated at the edge router before reaching the content provider. The content provider receives only one Interest request. Then one Data packet is replied and distributed to the users. The entire packet processing procedure is transparent to the users and content providers.

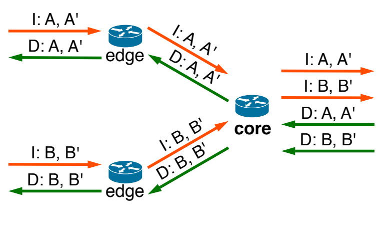
The PITs in edge routers operate as described in the NDN design [2], where name strings are stored, and Interest aggregation is supported. The PITs in the core routers store fixed-length fingerprints instead of name strings. Two challenges arise with this approach: *fingerprint collisions* and *duplicate Interest requests*. To guarantee packet delivery, Interest aggregation is not supported in the core routers when fingerprint collisions occur. Colliding fingerprints are not deleted from the PIT until they reach the expiration time T_{exp}, giving enough time to wait for potential multiple Data packets. To achieve this, each PIT entry in the core routers records if duplicate fingerprints have been received. The PIT entry expiration time and face list information are managed in the same fashion in edge and core routers. The second challenge is duplicate Interests. Although a duplicate Interest packet from a different face would make its PIT entry stay longer, we leverage the idea that most Interest packets are aggregated in the edge routers, and we will analyze the effect of duplicate Interest requests in Section VI.

While the edge routers follow the original operational flows as illustrated in [3], the operational flows of the core routers are modified. When an Interest packet arrives at the core router, the Content Store is queried to see if the content is cached. If the requested content is not cached, the PIT is consulted to see if it has already been requested. The lookup key is the fingerprint of the content name. If there is no match in the PIT, then this fingerprint is inserted, its expiration time is set, and its incoming face is recorded. If there is a match for this fingerprint in the PIT, then additional information about the collision is updated, the expiration time is refreshed, and the incoming face is added. In the end, the Interest packet is forwarded to the appropriate outgoing face by performing a FIB lookup, regardless of whether there is a PIT match or not. On the arrival of a Data packet, the packet will be selectively cached based on the Content Store caching policy. Then, the packet is looked up in the PIT, with the content name fingerprint as the key. If the fingerprint is found, then the Data packet is delivered to the face(s) stored in the face list. If this fingerprint has been received only once, it is removed from the PIT immediately; otherwise, it stays until the expiration time is reached. There are three operation situations, as shown in Figure 2, that could occur in this design.

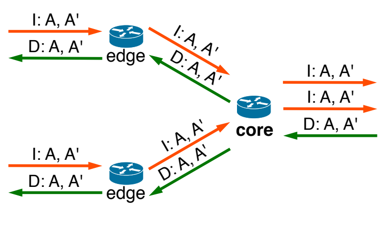
1) Normal Operation: In the normal case, different Interest names map to different fingerprints. For example, name A maps to fingerprint A', and name B maps to B'. As a result, the operation is the same as the case where name strings are stored. No traffic overhead is introduced.

2) Duplicate Interest Requests: When duplicate Interest requests arrive at the core router from different faces, the fingerprint stays in the PIT longer. Hence, there is memory overhead compared with the normal operation, and there is Interest traffic overhead since duplicate requests are not aggregated.

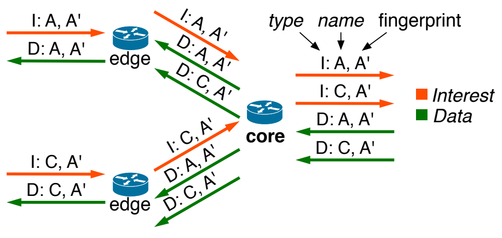
3) Fingerprint Collisions: If two content names share the same fingerprint, then these two Interest packets take one PIT entry, rather than two, reducing the number of stored PIT entries. Still, the colliding PIT entry stays longer in the PIT, which introduces memory overhead compared with the normal case. Since the two Data packets are delivered to both faces, Data traffic overhead is also introduced. In a special case,



(a) Normal Operation



(b) Duplicate Interest Requests



(c) Fingerprint Collisions

Fig. 2. System Operations

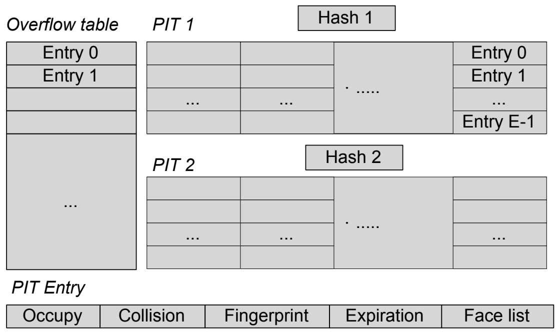


Fig. 3. PIT Data Structures

if one Interest has been requested by many faces, then the absolute value of the additional traffic is much larger.

B. Data Structures

Our proposed PIT design is based on a d-left hash table [5], where the hash buckets are grouped to d subtables. To insert an item, all of the d subtables are visited and the item is inserted to the least loaded subtable. Figure 3 shows the PIT architecture with $d = 2$ and also a PIT entry example. Each hash table has a capacity of N/d entries, and it has B buckets, where each bucket holds up to E entries. The architecture also has an overflow table to store the items that cannot fit into the hash tables. The hash function determines the bucket location of a name string and also its fingerprint. As described in [9], the d hash functions in the PIT have to use permutation to generate the bucket indexes and fingerprints.

Each PIT entry consists of five parts: the Occupy bit, the Collision bit, the fingerprint, the expiration time, and the face list. The Occupy bit indicates whether this PIT entry space is occupied or not. It also enables lazy deletion, since deleting an entry requires only setting this bit to 0. The Collision bit shows whether more than one Interest requests have been received for this fingerprint. The fingerprints have fixed lengths, w-bit long. The expiration times also have a fixed length of t bits. The value of t is determined by the configured expiration time support and its granularity. In our design, $t = 10$ and its unit is 16 ms, thus the timer counts up to 16 seconds. The face list stores the incoming faces. In our design, it is a bit-vector of length n, where n is the number of faces. The bit vector is initialized to 0s, and the i^{th} bit is set to 1 when an inserted Interest comes from face i. In our analysis, we assume a core router with 16 ports. When the number of ports is larger, bit vector may not be the best face list representation, thus memory-efficient alternatives need to be explored, but it is beyond the scope of this paper.

Algorithm 1 shows the steps for inserting an item into the PIT, while the method that handles timer rollover is not included here. We use a lazy expiration time check scheme, so the expiration time will not be examined unless it is visited during some operation. The deletion algorithm, which employs the lazy deletion method, is not shown here due to space limits.

Algorithm 1: INSERT AN ENTRY INTO THE PIT

```
Input: PIT $T$, Name $name$, Incoming face $f_i$
$loc \leftarrow H(name)\ mod\ B$
$fp \leftarrow \text{SHIFTRIGHT}(H(name), log(B))\ mod\ 2^w$
foreach $entry$ in $T(loc)$ do
    if $entry.occupy = 1$ then
        if $entry.expiration < current$ then
            RESET($entry$)
        else
            if $fp = entry.fp$ then  $fp$ is found
if $fp$ is found at $entry$ in $T(loc)$ then
    $entry.collision \leftarrow 1$
else
    $entry \leftarrow empty\ entry\ in\ T[loc], or\ overflow\ table$
    $entry.occupy \leftarrow 1$
    $entry.fp \leftarrow fp$
$entry.facelist[f_i] \leftarrow 1$
$entry.expiration \leftarrow current + T_{exp}$
```

C. Segregated Pending Interest Table

Although our fingerprint-only PIT significantly reduces the memory requirement, it may still exceed the size of a single memory chip when the link rates are high. We propose a segregated Pending Interest Table to address this problem. In this design, the single PIT is divided into s segregated PITs in the router. Each segregated PIT keeps track of the unsatisfied Interest packets from n/s faces. As a result, the effective bandwidth of each segregated PIT is $1/s$ of the full link rate, and thus the required memory is much smaller. The Interest packets are sent to the appropriate segregated PIT based on their incoming faces, and the rest of the packet processing stays the same. In contrast, on the arrival of Data packets, all of the segregated PITs are queried to find the incoming face(s) of the corresponding Interest packets.

V. POPULAR CONTENT OPTIMIZATION

In this section, we first discuss the challenges of popular content, and then present optimization methods that address these problems.

A. Popular Content

Our fingerprint-only Pending Interest Table is based on the idea that most Interest packet aggregation happens at the edge routers, and therefore the core routers are more likely to receive unique Interest requests. We have two observations that further support this assumption. The first observation is that the effective Interest waiting time is short, generally equivalent to a packet round trip time T_{rtt}. In today's Internet, T_{rtt} is around 80 ms on average [10], thus the chance of receiving a duplicate Interest during such short period is low for general content. The second observation is that highly accessed pieces of content, as time goes on, are gradually cached in edge networks, so subsequent Interest requests can be satisfied locally and will not enter the core networks. Thus, the chance of receiving duplicate Interest requests during T_{rtt} is further reduced. However, certain types of content could still be requested multiple times during T_{rtt}. For instance, live sports games and concerts could possibly be watched by millions of people from different edge networks at the same time. These Interest requests are more likely to be synced and arrive within T_{rtt}. In this case, the PIT entry stays at least T_{exp} long since a colliding entry cannot be removed until it has expired. In theory, the worst case could be that another Interest packet arrives right before the current entry expires, causing this entry to stay for at least another T_{exp} amount of time. Thus the entry could stay for approximately $T_{exp} \times (n-1)$ long in the worst case scenario. Interest requests like this increase the PIT memory requirement and therefore need to be mitigated.

B. Optimization

We present three orthogonal methods to mitigate the popular content problem.

1) Content Store: The worst case presented earlier assumes the requested Interest packets cannot be satisfied by the Content Store (CS), which caches Data packets in an NDN router. Interest packets do not query the PIT if they are satisfied by the Content Store. Storing popular content in the Content Store not only prevents colliding PIT entries being refreshed once the Data packets are cached, but also reduces the content delivery time for the users. Hence, a dynamic selective Content Store that gets invoked when a PIT entry becomes hot can be employed to mitigate the popular content problem. For instance, the Content Store can be configured to cache Data packets that have been requested more than once. Under this policy, each PIT entry stays at most $T_{rtt} + T_{exp}$ long. The size of the Content Store is determined by the number of duplicate Interest requests received.

2) Adaptive PIT: The proposed PIT is designed for the worst case, where minimal-size Data packets are used. In practice, network traffic does not stay at peak all the time. As a result, we propose an adaptive PIT design, where fingerprints are stored under heavy traffic load, and full name strings are stored under light traffic load. Storing the names supports Interest aggregation, therefore the popular content problem would not occur. This idea is conceptually similar to the dynamic bit assignment method for fingerprint-based hash tables [12], which adjusts the fingerprint length based on the number of occupied entries in a hash bucket.

3) Segregated PIT: The segregated PIT design can also be used in this context. Since the incoming traffic is divided into multiple sub-streams, the chance of getting duplicate Interest requests is reduced. In a special case, a smaller PIT can be deployed for each face. There are no duplicate Interest requests at each face due to Interest aggregation at the edge routers. Approaches that deploy a Bloom filter for each face [13][14][15] have been studied.

In addition, application-level optimizations, such as designated broadcast names, can be explored to help mitigate the popular content problem.

VI. Analysis

Memory size and *network traffic overhead* are the two major metrics for evaluating the proposed Pending Interest Table design, and both of them are affected by the fingerprint distribution in the PIT. In this section, we first analyze the fingerprint-based hash table and derive an upper bound of the number of names that have duplicate requests, and then we present a detailed analysis of the memory size and network traffic overhead. Since the actual characteristics of the traffic in the NDN core networks is not known, we use analytical modeling and Zipf-like fingerprint distributions instead. It is worth noting that relaxing Interest aggregation changes the dynamics in the core networks, and there are more Interest packets when duplicate Interest requests occur. For instance, when every content is requested twice, there are two Interest packets for every Data packet, so that 66.7% of the network packets are Interest packets.

A. Fingerprint-Based Hash Tables

Our analysis differs from previous studies: we are more interested in the fingerprint distribution than in the hash table false positive rates, which are typically considered since fingerprint-based hash tables are used as a membership query tool [9]. In these studies, duplicate items have no effect on how long the corresponding fingerprint is kept in the table. In contrast, our design uses hash tables to manage a stateful flow table. Duplicate Interest requests will cause the PIT entry to be recognized as collided and to be kept longer in the table.

The fingerprint distribution problem can be formulated as follows: *Given an Interest name distribution function F for a hash table with capacity N, and w-bit long fingerprints, what is the corresponding fingerprint distribution function f?* The value of F_i in function F is defined as the number of names that have exactly i duplicate copies (including itself) during T_{rtt}. For instance, there are F_1 number of Interest names being requested only once. Hence, the total number of unique Interest names is $N_{total} = \sum_{i=1}^{n} F_i$, where n is the number of faces corresponding to this PIT. The total number of Interest packets is $P_{total} = \sum_{i=1}^{n} F_i \times i$.

Collisions affect the fingerprint distribution. A fingerprint collision occurs if and only if both the hash bucket locations and the fingerprints are identical. The average number of keys stored in each bucket is determined by the hash table load factor. In our design, each bucket has eight entries, and the load factor is 75% as this configuration provides good performance [9]. As analyzed in [9], the fingerprint collision rates are bounded by $E \times d \times ld \times (1/2^w)$, where E is the number of entries per bucket, d is the number of subtables, and ld is the load factor of the hash tables. In our case, when $d = 4$, $E \times d \times ld = 24$, thus the average fingerprint collision rates are bounded by $24/2^w$. As a result, the ratio of the names that involve fingerprint collisions is $24/2^{w-1}$. In addition, we derive that the probability of having i names collide is bounded by $\binom{24}{i-1}/2^{w\times(i-1)}$.

Deriving the entire fingerprint distribution function f seems possible, but requires complex mathematical work. Instead, we derive an upper bound on the number of duplicate fingerprints rather than seeking an accurate distribution function. The upper bound is sufficient for performing the worst case analysis. The number of fingerprints that appear exactly once is

$$f_1 \geq F_1 \times (1 - 24/2^{w-1}). \tag{1}$$

In the worst case, each duplicate Interest name is requested exactly twice. Thus the number of duplicate fingerprints is

$$\sum_{i=2}^{\infty} f_i \leq (P_{total} - f_1)/2. \tag{2}$$

B. Memory Size

The PIT memory size is determined by the number of entries and the lifetime of each entry. Given a fingerprint distribution f, the total number of Interest packets is $P_{total} = \sum_{i=1}^{\infty} f_i \times i$. The PIT memory size is

$$M_{total} = \left(f_1 + \sum_{i=2}^{\infty} (f_i \times \frac{T_i}{T_{rtt}}) \right) \times \frac{M_{bucket}}{ld}, \tag{3}$$

where ld is the load factor of the hash table, and T_i is the average Interest lifetime of the names that have exactly i duplicate Interest requests. In our design, 10-bit expiration timestamps, 16-bit fingerprints, and 16-bit face list bit vectors are used, therefore $M_{bucket} = 2 + 10 + 16 + 16 = 44$ bits.

1) Worst Case Analysis: When there is no Content Store, the worst case lifetime of an entry can be as long as $T_{exp} \times (n-1)$ in theory, where n is the number of interfaces in the router. The traffic pattern that causes this worst case behavior should rarely happen for general content. In the case where they might occur, such as a live sports broadcasting, the Content Store should be deployed as we have discussed in Section V. To make the worst case manageable, we assume there is a Content Store. With a proper CS caching policy, Data packets corresponding to duplicate fingerprint entries are cached. Thus the duplicate Interest requests that arrive during T_{rtt} will be inserted into the PIT and then forwarded, while the ones arriving after T_{rtt} will be satisfied by the CS. The content needs to be cached for at least T_{exp} long, so that the PIT entry can expire and get deleted. In this case, the Content Store size is $r \times T_{exp}/T_{rtt}$, where r is the number of names that have duplicate requests during T_{rtt}. In this scenario, each duplicate fingerprint entry stays for at most $T_{rtt} + T_{exp}$ long in the PIT. The worst case is that every content is requested by exactly two faces. Thus we have the memory size as

$$M_{total} = \left(f_1 + \frac{P_{total} - f_1}{2} \times \frac{T_{rtt} + T_{exp}}{T_{rtt}} \right) \times \frac{M_{bucket}}{ld}. \tag{4}$$

We define $T_{life} = T_{rtt} + T_{exp}$ as the Interest lifetime for the collided fingerprint. In addition, we let $T_{life} = k \times T_{rtt}$ and call k the lifetime factor. Figure 4 presents the memory requirement with different numbers of subtables, lifetime factors, and duplicate request traffic percentages. In Figure 4, we use a 100 Gbps link, assuming the Interest and Data packet sizes are $S_I = S_D = 100$ bytes, $T_{rtt} = 80$ ms, load factor $ld = 75\%$, the lifetime factor k is increased from 6 to 10, and the duplicate request percentage is increased from 0% to 100%. The fingerprint length is set to 16 bits, and the the number of subtables d is set to 2 and 4. In both cases, the memory size increases almost linearly as the percentage of duplicate traffic increases. The memory requirement of 4-left hash tables is slightly higher than the one of 2-left hash tables due to a higher fingerprint collision rate. The lifetime factor k increases the memory requirement linearly at each duplicate traffic percentage. With $k = 10$, in the ideal case when there is no duplicate Interest traffic, the memory requirement is 5.34% of the original hash table that stores name strings with the same 75% load factor; in the worst case that all traffic is duplicate, the memory requirement could be as high as 35.51% of the original. At 100 Gbps, the ideal case requires 36.77 MiB, which can be stored in SRAM; and the worst case needs 244.44 MiB, which can be implemented using RLDRAM.

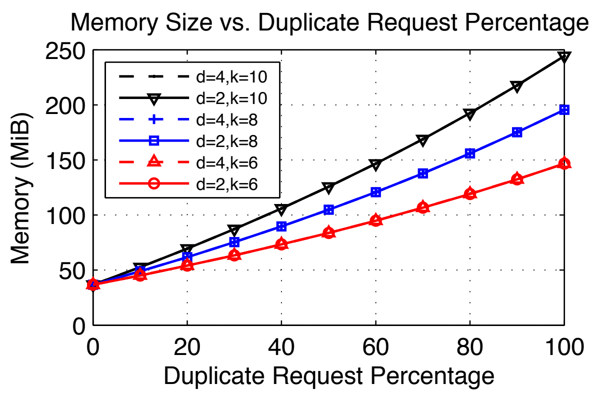


Fig. 4. Memory Requirement

2) Zipf-like Fingerprint Distributions: Zipf-like distributions have been observed in many network activities, such as the content request patterns to web caching proxies [16]. We use Zipf-like distributions to simulate the Interest fingerprint distribution. Zipf-like distributions are configured with an exponent characterizing parameter, S. And we have $f_i \times i = P_{total}/(H \times i^S)$, where $H = \sum_{i=1}^{16} 1/i^S$. Four Zipf distributions are used so that we could see the trend of the memory requirement, and the real-world cases are more likely to be covered. Table II lists the percentage of unique fingerprints and the memory requirement with $d = 4$ and

TABLE II
MEMORY REQUIREMENT OF ZIPF-LIKE FINGERPRINT DISTRIBUTIONS

S	**1**	**2**	**3**	**4**
Unique (%)	29.6	63.1	83.3	92.4
Memory (MiB)	101.1	79.1	58.6	47.4
vs. Original (%)	14.7	11.5	8.5	6.9
vs. Ideal	2.7x	2.2x	1.6x	1.3x

$k = 10$ at 100 Gbps. The memory sizes are also compared with the original hash table and the ideal case. From Table II, when $S = 1$, which yields the largest memory requirement, the memory size is 14.7% of the original hash table. Thus in practice, the PIT is lightly loaded when the worst case memory size, 35.51% of the original hash table, is configured.

3) Dynamic Expiration Time Management: Configuring an appropriate expiration time T_{exp} is important in practice since it has a linear effect on the memory requirement. A longer T_{exp} increases the memory requirement, while it provides a stronger guarantee to receive potential multiple Data packets. The lifetime factor k is set to 10 in Figure 4 and Table II because we believe $T_{exp} = (10 - 1) \times T_{rtt}$ is long enough for the Data packets to arrive. In practice, the values of T_{rtt} vary for packets that going to different websites. Popular websites usually support reliable short response times, and $T_{exp} = 9 \times T_{rtt}$ could be overkill. To optimize the PIT memory requirement, T_{exp} can be dynamically configured for each PIT entry. The strategy layer introduced in NDN [2] could provide T_{rtt} for each name prefix, thus each name prefix could maintain a recommended T_{exp}. For instance, T_{exp} can be reduced to 2 or 3 times of T_{rtt} for popular websites. In the case where the content cannot be fetched quickly on the server, the pending Interest would expire, but the application will resend an Interest request. And this time, the object could be retrieved quickly by the server since it has just been requested.

C. Network Traffic Overhead

Network traffic overhead is introduced due to the relaxation of Interest aggregation and fingerprint collisions. Since the Interest packets are always forwarded, the Interest traffic overhead is $T_I = P_{total} - N_{total}$. It should be noted that the Interest traffic never exceeds the link capacity, since the link rates are configured to support the case where every Interest is unique. The Data traffic could exceed the link capacity, but as we will show, the overhead is very small. The Data traffic overhead is caused by fingerprint collisions. Assuming each Interest name is requested by p faces, and that q packets collide, then the total traffic overhead for these q packets is bounded by $p \times (q-1) \times q$. To provide an estimation, we use the average number of duplicate Interest requests, P_{total}/N_{total}, for each name. As described earlier, the probability of having i colliding fingerprints is $\binom{24}{i-1}/2^{w \times (i-1)}$, thus we have

$$T_D \approx P_{total} \times \sum_{i=2}^{\infty} \frac{\binom{24}{i-1} \times (i-1) \times i}{2^{w \times (i-1)}}. \qquad (5)$$

When 16-bit long fingerprints are used, the equation is reduced to $T_D = 7.3281 \times 10^{-4} \times P_{total}$. The total amount of additional traffic in the network compared with the ideal design is $T_{total} = T_I \times S_I + T_D \times S_D$, where S_I is the Interest packet size and S_D is the Data packet size.

Figure 5 shows the normalized network traffic of the proposed design and the ideal case, where Interest aggregation and Data multicast are supported by storing name strings. The traffic is normalized to the Internet traffic for delivering the same amount of content, and neither Interest aggregation nor Data multicast is supported in the Internet case. The difference between the traffic of the proposed design and the ideal case shows the overhead. From Figure 5, the traffic overhead increases as α increases, mostly due to the Interest traffic overhead. In the worst case, when $\alpha = 16$ and $S_D = 100$ bytes, the normalized traffic of the proposed design (53.16%) is 8.51 times of the ideal case (6.25%). The traffic overhead decreases as S_D increases. When $\alpha = 16$ and $S_D = 1500$, the traffic is 1.95 times of the ideal. The Data traffic overhead increases as α or S_D increases, and the peak Data traffic overhead is 1.10%, where $\alpha = 16$ and $S_D = 1500$. When $\alpha = 1$ and $S_D = 1500$, the normalized traffic of our design is 100.069%, which exceeds the link capacity by 0.069% due to Data traffic overhead. It is worth nothing that the average number of requests α generally is very low, thus the traffic overhead would be small. Moreover, the proposed design overloads the link capacity by at most 0.069%, because the ideal design also needs to support the case where every Interest is unique. When the Interest traffic overhead is a concern, optimization methods, such as storing names for popular requests, can be applied.

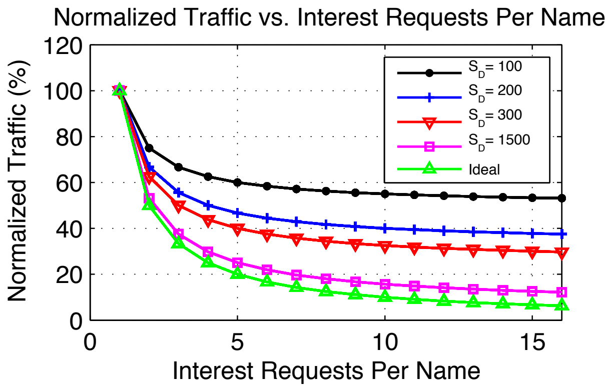


Fig. 5. Network Traffic Overhead

TABLE III
TRAFFIC OVERHEAD OF ZIPF-LIKE FINGERPRINT DISTRIBUTIONS

S	1	2	3	4	Data
Unique (%)	29.6	63.1	83.3	92.4	NA
100 B (%)	56.8	16.1	5.5	2.2	0.078
200 B (%)	37.9	10.7	3.7	1.5	0.104
300 B (%)	28.5	8.1	2.8	1.2	0.117
1500 B (%)	7.2	2.1	0.8	0.3	0.147

We again use Zipf-like distributions to study the traffic overhead in general networks. Table III lists the percentage of the traffic overhead compared with the ideal case using multiple Zipf distributions and Data packet sizes. The maximum Data traffic overhead is also listed in Table III. When the exponent characterizing factors S is greater than 2, the traffic overhead is always less than 16.1%. The overhead is always small with large Data packets. In addition, the peak Data overhead is 0.147%, which is negligible in most network environments.

D. Segregated PIT Analysis

The segregated PIT design reduces the memory requirement for each smaller PIT since the number of entries is only $1/s$ of the single PIT. Moreover, the entry size of the segregated PITs is also smaller because the face list length is reduced to $\lceil n/s \rceil$. However, the fingerprint length needs to be increased slightly. Assuming the fingerprint collision rate of a single PIT is f_p, then the fingerprint collision rate of the segregated PITs is $1-(1-f_p)^s \approx s \times f_p$. To maintain the same overall fingerprint collision rate, the length of the fingerprints stored in the smaller PITs needs to be $\lceil log(s) \rceil$ bits longer. Thus, each PIT entry size is reduced by $n - (\lceil log(s) \rceil + \lceil n/s \rceil)$ bits. For instance, when a 16-face router is divided into two smaller PITs, the PIT entry size is reduced by seven bits. On the other hand, the memory bandwidth of the segregated PITs needs to be increased by s times, because all s segregated PITs are queried when Data packets arrive.

VII. Performance

In this section, we evaluate the performance of the proposed Pending Interest Table design.

A. Experimental Setup

We implemented the fingerprint-only Pending Interest Table in C++. The hardware platform used was a machine equipped with 8 Intel Xeon E5540 cores, 8 MiB of L3 cache, and 12 GiB of DDR3 memory. Since real-world NDN core router traces were not available, a synthetic Interest trace was generated by appending a number ranging from 0 to 999 to each domain name in the Alexa top 1 million websites [17].

B. Simulation

We measured the fingerprint collision rates and the number of overflowed names with different PIT size N, and different numbers of subtables d. Each bucket has $E = 8$ entries, the PIT load factor ld was set to 75%, 16-bit fingerprints were used, and the lifetime factor k was set to 10. In the experiment, m names were inserted in the first phase. A certain percentage, denoted as d_p, of the names are not unique (requested twice), and d_p was set to 0%, 20%, 40%, 60%, 80%, and 100%. The value of m was adjusted accordingly for each d_p so that the targeted number of occupied entries in the PIT was always close to $ld \times N$. In the second phase, names were updated; and the i^{th} name was deleted and then the $(i+m)^{th}$ name was inserted, where $i \in \{0...25 \times m\}$. In the simulation, we use the Interest name index, i, as a timing tool. The T_{life} is configured to be the number of names, m_{10}, inserted during $10 \times T_{rtt}$. The time at any point was set to the index of the latest inserted name. Thus, an Interest with index i would be considered as expired once the $(i+m_{10})^{th}$ name was inserted. The largest measured fingerprint collision rates, denoted as f'_p, and the number of the overflowed names, denoted as M, are listed in Table IV. When the PIT has $d = 4$ subtables, it never overflowed, thus its overflow size is not listed. In the table, the PIT with size $1,048,576$ is denoted as $1M$.

TABLE IV
FINGERPRINT COLLISION RATES AND OVERFLOW SIZES

Type	d	1M	2M	4M	8M	16M
$f'_p(10^{-4})$	2	1.79	1.78	1.77	1.77	1.75
	4	3.55	3.53	3.56	3.57	3.53
M(10^5)	2	0.08	0.17	0.34	0.67	1.32

From Table IV, the fingerprint collision rates in both $d = 2$ and $d = 4$ cases are close to the theoretic values, which is expected. When there are $d = 4$ subtables, overflow never occurs as each inserting name has 32 choices. But when $d = 2$, each inserting name has only 16 choices, overflow occurs as names being updated. The overflow size increases linearly as the PIT table size increases. When the PIT size $N = 16$ million, 12.53 million names are stored, and the overflow size is 132.23K. Even in this case, the overflow table memory requirement is less than 0.73 MiB, which can be stored on SRAM or TCAM.

C. Latency Measurement

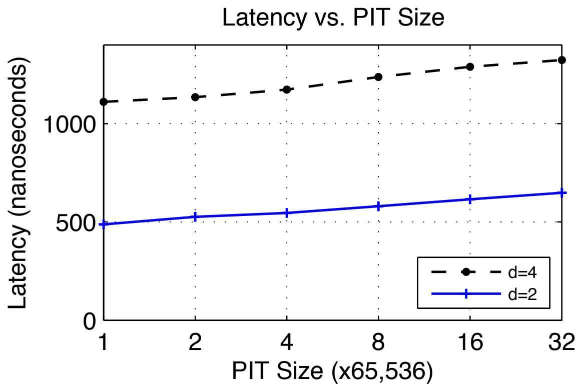


Fig. 6. PIT Operation Latency

We measured the PIT operation latency of the software implementation of the proposed PIT. The experimental setup was the same except that we fixed d_p at 0% since this case required the highest update frequency. Hash values were computed using the 64-bit CityHash function [18]. Figure 6 shows that the average latency of each operation increases as the PIT size increases. The average latency with 4 subtables is approximately twice of the case with 2 subtables, since the number of memory accesses is doubled. In hardware designs, multiple words can be read from SRAM chips in parallel, which reduces the latency. At 1 Gbps, the PIT size is close to $N = 65,536$, and the measured latency is about 1.2μs.

VIII. Discussion

In this section, we discuss the modification of the proposed Pending Interest Table when false positives are allowed, and also consider the PIT security issues.

A. Allowing False Positives

In networks where a small number of false positives are allowed, the proposed fingerprint-only Pending Interest Table can be modified to support Interest aggregation, just like name strings are being stored. This way, only one of the colliding Interest names is forwarded and the rest are dropped. The false positive rates of the fingerprint-only PIT in this case are the same as the previous analysis on the case with no duplicate Interest requests. At 100 Gbps, with the same configuration, the memory requirement of the modified PIT is 36.65 MiB, slightly less than 36.77 MiB, because collided fingerprints do not stay longer in the table. Bloom filters have also been considered to implement PIT when false positives are allowed [13][14]. In these designs, a Bloom filter is deployed for each face, therefore the Interest aggregation is not supported. Overall, when false positives are allowed, the fingerprint-only PIT is preferred because Interest aggregation can be supported, and the expiration time and face list can be easily stored.

B. Pending Interest Table Security

The proposed Pending Interest Table is designed for the worst case flow balance mode, but it will not be able to handle an Interest flooding attack, where every Interest packet stays for T_{exp}. In this case, statistics collected on the routers should be able to detect the attacks and then apply countermeasures. In addition, our proposed architecture could potentially address the Interest flooding problem by designing a fingerprint-only PIT that is large enough to hold all the flooded packets. The memory requirement may still be acceptable since only a fixed-length fingerprint is stored for each Interest.

IX. Related Work

Prior hash-table based works on scalable PIT designs all support Interest aggregation in both the edge and core networks, while our solution is the only one that relaxes the Interest aggregation feature. The principles of scalable PIT designs are highlighted in [3]. The hash-based methods in [19] store both fingerprints and name strings in the PIT. Although string comparison can generally be avoided by checking their fingerprints, at least one DRAM write operation is required to store the content name. Encoding methods [20] have been proposed to reduce the PIT memory size, while additional lookups that encode the content name are required before querying the PIT. Moreover, the transition arrays need to be dynamically updated, increasing the complexity of the system. The performance of the encoding methods and hash-based methods are compared in [21]. When false positives are allowed, our design can be modified and supports Interest aggregation as described in Section VIII; Distributed Bloom filters [13] [14] have also been used as PITs in this context.

X. Conclusion

Fast and scalable Pending Interest Table design is a challenge in Named Data Networking. In this paper, we propose a Pending Interest Table design that guarantees packet delivery and significantly reduces the memory requirement by storing fingerprints rather than name strings. The Interest aggregation feature in the core routers is relaxed so that packets are guaranteed to be delivered even when fingerprint collisions occur. We have studied the memory requirement and network traffic overhead analytically, and demonstrated that the additional network traffic is acceptable. We have also measured the performance of a software implementation of the proposed design. Our results show that 37 MiB to 245 MiB of memory are required at 100 Gbps, so the PIT can be implemented with SRAM or RLDRAM chips.

Acknowledgment

This work has been supported by NSF Grant CNS-1040643.

References

[1] Lixia Zhang et al. Named Data Networking (NDN) Project. Technical Report NDN-0001, NDN, 2010.

[2] Van Jacobson et al. Networking Named Content. In *CoNEXT '09*, pages 1–12, New York, NY, USA, 2009.

[3] Haowei Yuan, Tian Song, and Patrick Crowley. Scalable NDN Forwarding: Concepts, Issues and Principles. In *ICCCN '12*, pages 1–9, 2012.

[4] CCNx. http://www.ccnx.org/. Accessed: 07-22-2013.

[5] Adam Kirsch, Michael Mitzenmacher, and George Varghese. Hash-Based Techniques for High-Speed Packet Processing. In *Algorithms for Next Generation Networks*, pages 181–218. 2010.

[6] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John Lockwood. Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing. In *SIGCOMM '05*, New York, NY, USA.

[7] Sailesh Kumar and Patrick Crowley. Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems. In *ANCS '05*, pages 91–103, New York, NY, USA, 2005.

[8] Sailesh Kumar, Jon Turner, and Patrick Crowley. Peacock Hashing: Deterministic and Updatable Hashing for High Performance Networking. In *INFOCOM '08*, pages 101–105, 2008.

[9] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. An Improved Construction for Counting Bloom Filters. In *ESA'06*, London, UK, 2006.

[10] Diego Perino and Matteo Varvello. A Reality Check for Content Centric Networking. In *ICN '11*, pages 44–49, New York, NY, USA, 2011.

[11] Cisco Carrier Routing System. http://www.cisco.com/en/US/products/ps5763/index.html. Accessed: 07-22-2013.

[12] Flavio Bonomi, Michael Mitzenmacher, Rina Panigrahy, Sushil Singh, and George Varghese. Bloom Filters via d-left Hashing and Dynamic Bit Reassignment. In *Allerton Conference '06*, 2006.

[13] Wei You, B. Mathieu, P. Truong, J. Peltier, and G. Simon. DiPIT: A Distributed Bloom-Filter Based PIT Table for CCN Nodes. In *ICCCN '12*, 2012.

[14] Wei You, B. Mathieu, P. Truong, J. Peltier, and G. Simon. Realistic Storage of Pending Requests in Content-Centric Network Routers. In *ICCC '12*, pages 120–125, 2012.

[15] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations. In *CoNEXT '09*, pages 313–324, New York, NY, USA, 2009.

[16] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *INFOCOM '99*, volume 1, pages 126–134 vol.1, 1999.

[17] Alexa. http://www.alexa.com/topsites/. Accessed: 07-22-2013.

[18] CityHash. https://code.google.com/p/cityhash/. Accessed: 07-22-2013.

[19] Won So, Ashok Narayanan, Dave Oran, and Yaogong Wang. Toward Fast NDN Software Forwarding Lookup Engine Based on Hash Tables. In *ANCS '12*, pages 85–86, New York, NY, USA, 2012.

[20] Huichen Dai, Bin Liu, Yan Chen, and Yi Wang. On Pending Interest Table in Named Data Networking. In *ANCS '12*, pages 211–222, New York, NY, USA, 2012.

[21] Matteo Varvello, Diego Perino, and Leonardo Linguaglossa. On the Design and Implementation of a Wire-Speed Pending Interest Table. In *INFOCOM '13, mini-conference*, 2013.