COMPRESSION OF PENDING INTEREST TABLE WITH ADAPTIVE PREFIX BLOOM FILTER IN NAMED DATA NETWORKING

1Anurudh Kumar, 2Dr. K. Suresh Joseph
1Research Scholar, Department of computer science, Pondicherry University, Puducherry, India.
2Assistant Professor, Department of Computer Science, Pondicherry University, Puducherry, India.
anurudh23@gmail.com, ksjoseph.csc@gmail.com

Abstract: In the recent years, Named Data Networking (NDN) become more popular which is the complementary design of the traditional IP architecture. NDN gives more importance to content rather than the data hosts. It enables the end user to give a data content request with no knowledge about the hosting entity. It manages user mobility, highly secure, flexible and scalable than the conventional Internet. Though NDN has several advantages, it has some issues in the hardware aspects like Pending Interest Table (PIT). PIT is a similar to a cache table which holds the interest, i.e. content and originating face. The scalability of PIT is an important challenge in the high-speed forwarding of NDN. This paper proposes an Adaptive Prefix Bloom filter with Name Lookup engine (NLAPB) to efficiently manage the size of PIT. The proposed work divides every NDN name (prefix) to B-prefix succeeded by T-suffix. The performance evaluation shows that it minimizes the memory requirement by 48% with an error probability of 0.076%.

Keywords: Bloom filter, Pending Interest Table (PIT), Named Data Networking, Name Lookup

1.Introduction

The Named Data Networking (NDN) is also termed as Content Centric Network (CCN) which is developed to modify the conventional communication design to cope up with the enormous applications of content distribution on Internet [1]. The communication is usually carried out by the receiver and there are two kinds of packets namely interest and content. The user requests are considered as interests and the response to the requests are called as content objects. The interests are routed using routers in Forwarding Information Base (FIB). In addition, the interests are sent back to the same path which generates the respective interests in reverse direction. This method is assured by a data structure called Pending Interest Table (PIT), which holds the list of interests. Every item of PIT contains a name (key) and a collection of faces (value). Since the entry counts are very large in many routers, it is a big challenge for routers to finish the fast forwarding when the PIT is archives in larger as well as slower storage spaces. Based on [2], the interests are arriving at a rate of 125million/sec and every packet requires 80ms as average for its round-trip time. For the process of fast forwarding, it is highly recommended to compress PIT.

The forwarding model [3] of a NDN node is given in Fig. 1. When a face receives an interest and the node fails to fulfill the interest, it adds the CCN name into the interest and its succeeding face in the PIT. Next, the interest is transmitted to the next hop in the direction of the source of content. The direction is decided based on the data available at FIB. When some faces to the content are available, routers will choose the correct one by executing an algorithm which elects the best next hop. For the purpose of compressing PIT, there are two characteristics to be noted and they are pre-packet write a key with variable length.

- Per-packet write: When a router received a packet every time (interest or data), PIT will be modified with adding, deleting or changing an entry. So, it becomes useless to partition the PIT to two versions: compressed version archived in compact but faster storage space and entire version archived in larger but slower storage spaces. At the end, this way will suffer from the non-availability of whole information.
Key with variable length: In reality, archiving a large amount of entries with the full name and variable length is a wastage of storage space. The name (key of the table) set of the entries in the PIT are less in the namespace defined by the occupied storage space. It shows that compression can be applied on names.

The above situations motivated us to propose an Adaptive Prefix Bloom filter with Name Lookup engine (NLAPB) to efficiently manage the size of PIT. The proposed work divides every NDN name (prefix) to B-prefix succeeded by T-suffix. The B-prefix is coordinated by Bloom filters (BF) and T-suffix is progressed by small-scale trie. The length of B-prefixes (and T-suffix) is strangled using the nature of popularity to enhance the lookup process. The performance evaluation shows that it minimizes the memory requirement by 48% with an error probability of 0.076%.

The succeeding part of the study is structured as follows: Section 2 provides an overview of existing works on PIT. The background information of BF is explained in Section 3. The proposed work is discussed in Section 4. The performance analysis are explained in Section 5 and the paper is ended in Section 6.

2. Related work

In this section, the existing BF to compress the PIT are discussed. For the sake of simplicity, the compared BF with the proposed method such as Counting Bloom Filter (CBF, [4]) and United Bloom Filter (UBF) [5] are explained here.

CBF is the enhanced version of BF to carry out he compression process as same as B-AAFFALO [6]. In addition, it is ineffective in real world scenarios. CBF has the demerit of false positives. As the information in compressed PIT is inaccurate, the false positive results in wrong deletion. Consider a router having 3 interfaces and the interests are entering from first face and transmitting to third face. When the transmitted data returns from the content provider and comes to the third face, the router questions the name in CBFs related to the first and second face. Though the first CBF leads to true positive answer, the second CBF might results in false positive answer. On the other hand, the router is unknown of it and it is a miscalculation due to the loss of original information. So, CBF deletes the name. Then, the counters in CBF reduce wrongly. When any of the counters reduces to zero, false negative will take place when succeeding queries uses the counter. This is called as counter leak.

False negative [7] also results in redundant traffic and packet loss. And, the number of counters leaks will increase when the time increases and the size of the value become very large. It leads to the degraded performance of CBF. To eliminate this issue, a timeout mechanism is included. The time is separated to various epochs and integrated a timeout bit for each counter in CBF. Using this process, the counters leaks can be rectified. To eliminate counter leaks, the value of m should be increased to decrease the probability of false positive. Based on bloom filter theory, m is defined as

\[ m = \frac{3r k T}{\ln 2} \quad (1) \]

Where \( r \) is the average round-trip time, \( m \) is the total number of bits in BF and \( k \) is the number of hash functions.

To overcome the drawbacks of CBF, UBF is proposed to reduce the requirement of deletion to stay away from unexpected anomalies. The names in PIT will present for a shorter duration and UBF utilizes this characteristic. It involves two BF. It partitions the time to various epochs.
In the beginning of every epoch, the non-current BF is set to 0. In this epoch, the addition is done on both BF and the query takes place on the present BF. In the end of the epoch, the BF will swap over the identities. All names in the BF will be erased at the end of the next epoch. Timeout behaves like an implicit deletion in UBF. The benefit of UBF is that the density can be managed as there is no incorrect deletion. But, it also has a drawback. The interests with similar name cannot be two or many times in one epoch couple. This issue is eliminated by the NDN cache. When a router caches all Data received in an epoch couple, the PIT manages the Interests with a specific name only once in this epoch couple. The parameters of UBF are given as

\[ m = \frac{2kT}{\ln 2} \]  

(2)

where \( k = \log_2 \left( \frac{u}{p} \right) \). To overcome the issues of BF in compressing PIT, the proposed method is introduced.

### 3. Bloom filter

A BF [8] is a bit-vector which represents the membership information of a group of elements. It indicates a set \( S \{x_1, x_2, ..., x_n\} \) of \( n \) elements is explained by an array of \( m \) bits and is assumed to be zero. This filter involves two functions namely programming and querying. In the first function, for an element \( x_i \) (for \( i = 1, ..., n \)) in the set \( S \), \( k \) diverse hash indices are calculated, where \( 0 \leq h_j(x_i) < m \) for \( j = 1, ..., k \). Every bit locations match to the \( k \) hash indices are fixed as 1 in Bloom filter. In other ways, every bit of a Bloom filter represents to a hash index, and as a replacement for of archiving the data itself in the hashing index, the equivalent bit of a Bloom filter is assumed to one and it indicates the presence of data.

Querying is done to verify that an input \( y \) is a member of the set. For any input \( y \), \( k \) hash indices are created based on the similar hash functions employed to design a filter. The bit-locations in the Bloom filter denoting the indices are tested. When a location is zero, then \( y \) is surely not a member set \( S \), and it is called negative. When all the location of the hash indices are found to be 1, the input might be an element of the set, and it is called positive. Though, it is feasible that the locations are not fixed by remaining elements in the set. This kind of positive results area called as false positive. For an element \( y \), which is not present in \( S \) (\( y = 2S \)) under querying, the false positive probability \( f \) can be calculated as [9],

\[ f = (1 - p)^k = \left( 1 - \left( \frac{1}{m} \right)^m \right)^k \]

For a given ratio of \( m = n \), it is clear that the false positive probability is reduced as the number of hash functions \( k \) involves the subsequent relation [10]:

\[ k = \frac{m}{2^{\left\lfloor \log_2 n \right\rfloor}} \ln 2 \]

As a total, a BF might give false positives but not false negatives. The false positive rate of a Bloom filter is managed easily by improving the BF size, but cannot be totally eliminated.

### 4. Proposed method

In this paper, a compression of PIT table is achieved by employing Adaptive Prefix Bloom filter with Name Lookup engine (NLAPB). It is based on hybrid lookup engine which combines counting bloom filter (CBF) and trie. It is encouraged due to the listed practical difficulties:

- NDN names contains a sequence of delimited components, which makes it hard to find the total number of filters are required in standard CBF-based solutions are used.
- Long NDN names are divided to a pair of short segments: B-prefix with a fixed length and T-suffix with a variable length. They can be individually treated by CBF and trie-based methods.
- The hierarchical NDN names supports aggregation, which indicates the number of diverse B-prefix, will be limited in a smaller when a particular length is employed. In this situation, it became easier to adopt CBF with a small \( fpr \).
- The trie data structure is highly flexible and scalable by dynamic memory allocation, which allows processing of the T-suffixes with variable lengths effectively.
The proposed work partitioned the name into B-prefix and T-suffix. The structure is represented in Fig. 2 illustrates the limited length B-prefixes, \( B1, B2, \ldots \), are effectively archived and processed by CBFs; T-suffixes, \( T1, T2, \ldots \), as an alternative, with variable lengths which are shorter than the real names are managed easily by trie, thus building a group of small-scale T-suffix trees which are combined to the B-prefixes with a hash table. In detail, a filter is assigned for every B-prefixes of individual length, and the filter size is measured its share of the total number of B-prefixes with various lengths. This relationship eliminates the asymmetric behaviors among various CBFs. When a prefix has to add or delete, B-prefix is initially added to or deleted from the filter and the counters representing the hash values are increased or decreased. Next, the hash table connecting B-prefixes with T-suffixes trees are updated. In the T-suffixes tree, every encoding component represents an element of trie. The initial element (root) of a trie is bound to a B-prefix with the hash table. From Fig. 2, it is shown that the parents of root elements are labeled as “-1”. Using the knowledge of the root, the subsequent components can be added or deleted with the trie by dynamic memory allocation and recycling. The proposed lookup procedure involves two stages using the separation of B-prefix and T-suffix, which completely varies from the Name Filter in [7].

The lookup framework is illustrated in Fig. 3. In the initial stage, the B-prefix of a name is initially processed by CBFs, and a hash table helps to determine the location of respective T-suffix tree. And, when the name is shorter, then the outgoing faces will be returned openly. In the next stage, using the knowledge of the root information (e.g. P1), the prefix matching persists on the basis of trie structure till a longest matching prefix is searched. Next, the respective forwarding face(s) will be returned. Likewise, the updates also involve the same two stage process. It is noted that there is a big probability that numerous diverse content names/prefixes distribute the similar B-prefix, thus, the number of dissimilar B-prefixes will be smaller, and the possibility of CBF updating is minimized.

An important component is still not available for the lookup process. With a particular value \( t \), each time the router looks for famous prefixes, the matching process involves more repetitive memory accesses for T-suffixes in the second stage. To decrease the unwanted repeating operations, B-prefix length can be varied with popularity of a prefix: the B-prefix will be longer when the prefixes are more famous. Finally, adaptive mechanism is presented to control B-prefix length in a particular range adaptively. Using this method, it is better to allocate a relative longer B-prefix and respective shorter T-suffix for famous prefixes. It can efficiently avoid many repetitions especially for famous prefixes, therefore, improving the processing of the entire system.
5. Performance Evaluation

A simple simulation experiment is carried out to determine the proposed PIT compression methodology. A CCN router with 4 hosts is connected to it. Every link holds the similar properties in terms of delay and throughput. Every host transmits Interests with (with an average length of 50 bytes) periodically to eliminate the effect of cache. The content of these names is consistently distributed in any of the other three hosts. The Interest generation rate achieves Poisson distribution. To realize the real-time environments, the hosts will wait for a random time before they return Data to simulate the varying RTT. Based on [119], the RTT to various hosts over the world is uniformly distributed to a particular range. Considering the content request meets Zipf-like distribution [12]. And more popular content is likely has the lower RTT because of cache or CDN. When the average RTT is 80ms, one possible range of RTT is [10ms, 250ms]. In experiments, the expected error probability of proposed work is lesser than 0.076%. For CBF, the memory is minimized about 10% with 0.1% error probability. In the proposed work, the storage reduction can reach 48% with only 0.076% error probability. This proves that in this scenario, the proposed work is a better choice than CBF and UBF.

6. Conclusion

For the process of fast forwarding, it is highly recommended to compress PIT. This paper proposes an Adaptive Prefix Bloom filter with Name Lookup engine (NLAPB) to efficiently manage the size of PIT. The proposed work divides every NDN name (prefix) to B-prefix succeeded by T-suffix. A simple simulation experiment is carried out to determine the proposed PIT compression methodology. A CCN router with 4 hosts is connected to it. The performance evaluation shows that it minimizes the memory requirement by 48% with an error probability of 0.076%.

References
