Bloom-Quotient Based Name Matching Technique in Content Centric Networks

Mohammad Alhisnawi1*
1 College of Information Technology- University of Babylon- Iraq.

Email: mohammad.alhisnawi@uobabylon.edu.iq

Abstract. Content Centric Networking (CCN) is a novel network paradigm in which the communication process focuses on content rather than the host. In the current network architecture, routers forward the incoming packets based on the routing information that already had. Whereas, forwarding decisions in CCN are taken depending on networks situation. The contents in CCN are retrieved depending on their names which are URL like names that are composing of a number of string parts separated by '/'. These names are stored inside a specific data structure, called Forwarding Information Base (FIB), inside CCN routers and they are used to forward any incoming packet. Therefore, the main two challenges that face the design of FIB table are: search speed and storage utilization. Consequently, in this paper, we propose a new name matching technique (named BF-QF FIB) to design and implement a FIB table in CCN routers in order to decrease the storage utilization and to increase the lookup speed. This technique utilizes two kinds of query data structures: Bloom filter (BF) and Quotient filter (QF) as its main data structures. The utilization of these two data structures will ensure low memory usage and high lookup speed. The results of our evaluation show that BF-QF FIB can guarantee high search rate and offer perfect scalability to large FIB tables.

1. Introduction

In CCN paradigm, content retrieving is accomplished by utilizing content names rather than position-based IP addresses. There exist two kinds of packets in CCN that are used in the operation of retrieving any content: Interest packet and Data packet. Whenever a consumer wants to request any content, he will generate an Interest packet that holding the CCN name of the requested content. As a response to this Interest packet, a Data packet will be transmitted back to the consumer. This Data packet carries the requested content. In contrast to the current network in which the requested content must be retrieved from the original producer, in CCN, any router which is in the path between the consumer and the producer can send back the requested content if it is already cached. The final feature will minimize noticeably the time taken to retrieve the requested content also it will has an important role in minimizing the overhead that will occur on the original producer [1].

Every CCN router holds three data structures: Content Store (CS), Pending Interest Table (PIT), and Forwarding Information Base (FIB). FIB table has an important role comparable to the
routing table in the current network architecture that it holds CCN names and its corresponding output ports and it is utilized to forward the incoming Interest packets. PIT holds the names of the requested but not satisfied yet contents and it is used to forward back the incoming Data packet through the ports that already requested it. CS is used to cache, temporarily, the retrieved content that is passed through the CCN router [2]. CS has limited capacity and, hence, it will be used to cache some of the retrieved contents and special algorithms are used to decide which of the retrieved contents should be cached. Also, there will be some caching strategies to decide how long each content should be remain within the cache [3].

Many techniques have been suggested previously to increase the performance of FIB table for CCN routers. Name Component Encoding (NCE) technique has been suggested in [4] in which the CCN names are encoded first before doing insertion/search operations. In this technique, a character trie is utilized to hold the outcome of the transformation process of CCN names into codes. The time taken by the encoding process will affect on the performance of this technique. In [5], another technique, called NameFilter, that utilizes two phases Bloom filters has been suggested. This technique depends completely on Bloom filters without utilizing any other data structure to store the CCN names and this will affect the accuracy of the technique. In [6], three main data structures have been employed: name prefix trie (NPT), BF, and a hash table. Bloom filter is utilized to store the CCN names that are initially stored in NPT. In the search operation, there will be no need to access NPT, rather, BF will be accessed first and the hash table will be accessed next. This technique consumes extra storage and cannot accommodate with increasing number of CCN names. In [7], an encoding technique, named RaCE, that uses radix trie structure has been suggested. Radix trie has been used to eliminate the need to store redundant portions of CCN names. The authors of [8] suggested to employ a patricia trie, which is an ordered data structure, to design and implement FIB table. Bit strings are used to store the portions of CCN names in an ordered way.

The remainder of this paper is structured as follows. Section 2 depicts the process of packet forwarding in CCN. Section 3 gives a brief description for Bloom filter. Section 4 describes main structure of Quotient filter. Section 5 describes the suggested technique. Section 6 describes our evaluation to the proposed technique. In Section 7, we conclude our work.

2. Packet Forwarding in CCN

To request a content, the consumer trigger an Interest packet that holds the CCN name of the requested content. When this Interest packet arrives to any router in the way between the requester and the producer, the router will examine its CS first. If the requested data is found, then the router will encapsulate it within a Data packet, and it will transmit back this Data packet to the consumer [9]. Generally, the Data packet traverse exactly the invert direction of the Interest packet. Every router can cache the data that exist within the Data packet that is passes through it. If the content store does not have the content, the router will search for the CCN name inside PIT table. If it is existing, then only the input port will be recorded without further forwarding the Interest packet. If not, this CCN name along with its incoming port will be recorded inside PIT and FIB table will be examined to forward the Interest packet over the appropriate output port [10].

Figure 1 depicts the forwarding operation of the CCN packets. Whenever a Data packet reaches to any CCN router, the latter will store, optionally, a copy of the retrieved data and it will check its PT. If there is a match, then the Data packet will be forwarded back through all the ports that registered in PT. Otherwise, the Data packet will be ignored. Notice that the examination process for the CCN names within the router will be done depending on the longest prefix match strategy [11,12].
Figure 1. Packet forwarding in CCN [11].

3. Bloom Filter

Bloom Filter (BF) is one of the most familiar approximate membership query data structures that is used to represent a set of items with a minimum memory requirement. It is simply a vector that stores $m$ positions each of them can store solely one bit. It performs insertion/query operations by employing $k$ hash functions. This memory efficiency of Bloom filter is come with a little ratio of false positive in which the Bloom filter response with positive query while the item is not already inserted. Remember that all kinds of such data structures have no false negative [13].

Initially, all the positions of BF contain zeros. To insert an item, the item will be forwarded to the various $k$ hashes and the corresponding $k$ locations inside BF will be set to ones. To check for an item, again the item will be forwarded to these $k$ hashes and the corresponding $k$ locations in BF will be examined. If any position of them holds zero that means the item absence otherwise the item is already inserted [14, 15].

Figure 2 presents an instance for item insertion/query operations. In this example, three hash functions ($k=3$) have been used: $h_1$, $h_2$, and $h_3$ and the size of Bloom filter is 10. The left side describes the mapping process of two elements: $x$ and $y$. Whereas, the right side describes the check process for two elements: $z$ and $r$ and here $z$ is an instance of false positive and $r$ is an instance of an absent element.
4. Quotient Filter

Quotient filter (QF) [17] is a compact query data structure that comprises m positions each one of them stores the fingerprint of an item. It utilizes one hash function and it depends in its work on a technique named “quotienting” in which the retrieved hash value of an item is split into two portions: quotient (q) and remainder (r). The remainder will be stored in a bucket that has index (q). In QF, canonical slot is used to store (r) in the absent of collision. Run preserves the hole slots with the identical (q). Whereas a cluster represents a larger series of taken slots in which solely the first item is saved in the canonical slot. Generally, a cluster preserves at least one run. Every position (i.e., slot) in QF has three attached bits: is_shifted: describes, when it is zero, the beginning of the cluster. is_continuation: describes, when it is zero, the beginning of the run. is_occupied: describes, when it is one, that the slot is canonical for any item. These three bits are used along with the first item in a cluster to restore the (r) value, i.e., fingerprint, of each saved item inside the cluster [18,19].

Figure 3 shows an instance of QF. It holds items A to H. Remainder and quotient for each item of them are presented in the left. The top of figure 3 is a hash table storing items A to H and the bottom part describes the way of saving the remainders in the filter. It can be concluded that items C, D, and E have identical quotient and, hence, they constitute a run. Item C is saved in its canonical slot and, hence, it represents the beginning of a cluster.
5. Proposed Technique
The proposed technique starts initially by storing all the existing CCN names into both Bloom and Quotient filters. All the portions of each CCN name will be stored without repetition. Bloom filter utilizes $k$ hash functions, and, in our technique, we will suppose that $k=2$ (i.e., hash1 and hash2 will be employed). Quotient filter utilizes one hash function and to minimize the time of insertion/query process we will make use of hash1 for both Bloom and Quotient filters. Here, we will make a modification to the standard Quotient filter by adding a new field beside the remainder field such that the new QF will store both the remainder value and the value of the output port in every position.

As an example, suppose that we have the following CCN name: com/fruit/apple that must be forwarded by port number 4, then the following steps will take place:
1. Com will be stored in both BF and QF with the NULL value as its output port.
2. Com/fruit will be stored also in both BF and QF with the NULL value as its output port.
3. Com/fruit/apple will be stored also in both BF and QF with the (4) value as its output port.

Now, suppose that we have the following CCN name: com/fruit/banana that must be forwarded by port number 6, then the following steps will take place:
1. Com will not be stored because its already have been stored in both BF and QF.
2. Com/fruit will not be stored because its already have been stored in both BF and QF.
3. Com/fruit/banana will be stored in both BF and QF with the (6) value as its output port.

Algorithm 1 depicts the insertion process of the CCN names inside Bloom and Quotient filters. This algorithm takes two inputs: content name and its output port. Then, in step 2, it extracts the number of portions inside this content name. In step 3, a loop will start that will work on all the portions of the content name from portion 1 to the final portion. In the first step in this loop, namely step 4, the algorithm extracts the present portion, then, it calculates the values of hash function1 and hash function2. Then, it examines if this present portion is already inserted in BF and QF or not. If it is already inserted, then the algorithm will not insert it and it will continue with the next portion. If not, the algorithm will insert it in BF and before inserting it in QF the algorithm will examine if the present portion is the final portion or not. If it is the final portion, then p value will be inserted with this present portion in QF. Otherwise, NULL value will be inserted with this portion in QF.
**Algorithm 1: insertion process**

1: Input: content name (n), output port (p)
2: portions= extracts number of portions (n)
3: for i = 1 to portions do
   4: presentpart= extract parts 1 to i from n
   5: calculate hash1 and hash1 for (presentpart)
   6: // examine if the presentpart is already inserted into BF and QF or not
   7: if ((examine(presentpart), BF, QF, hash1, hash2) = false) then
      8: // return true, if it is already inserted, and false if not
      9: insert (BF, presentpart, hash1, hash2)
   10: if (final portion of n has been reached) then
      11: insert (QF, presentpart, hash1, p)
   12: else
      13: insert (QF, presentpart, hash1, NULL)
   14: end if
   15: end if
4: end for

**Algorithm 2: examination process**

1: Input: content name (n)
2: outputport= defaultoutputport
3: portions= extracts number of portions (n)
4: for i = 1 to portions do
   5: presentpart= extract parts 1 to i from n
   6: calculate hash1 and hash1 for (presentpart)
   7: if (examine(presentpart), BF, hash1, hash2) then
      8: if (examine(presentpart), QF, hash1) then
         9: if (port field in QF <> NULL) then
            10: outputport= value of port field in QF
         11: end if
      12: else
         13: break
      14: end if
   15: else
      16: break
   17: end if
4: end for

Algorithm 2 depicts the examination process that shows how the CCN router deals with an incoming content name. The value of the default output port will be placed first in outputport variable. Then, in step 3, the algorithm extracts the number of portions inside this content name. In step 4, a loop will start that will work on all the portions of the content name from portion 1 to the final portion. In the first step in this loop, namely step 5, the algorithm extracts the present portion, then, it calculates the values of hash function1 and hash function2. Then, it examines if this present portion is already inserted in BF or not. If not, the algorithm will break the examination process immediately because this content name is not existing. Remember that the previous algorithm stores all the portions of the content name and, hence, if any portion of the content name is not existing in the BF that means the later portions will, absolutely, not be exist. If the content name is in the BF then the algorithm will examine QF (remember that BF has little ratio of false positive) and if the name is not found in QF then the algorithm will break the examination process immediately. Otherwise, the algorithm will examine the port value inside QF to replace the value of outputport with it if it is not NULL.
6. Evaluation

In this section, we will explain our evaluation for the suggested BF-QF FIB technique. In this evaluation, we employed an open-source package, namely, ndnSIM [20], which accomplishes CCN protocol stack for NS-3 simulator. The utilized CCN names have been downloaded from open-source, standard datasets [21]. These CCN names have various formats and lengths and have been collected from various URLs. Our suggested technique has been compared in the evaluation with three previously suggested techniques: patricia trie based name forwarding (N-FIB) [8], NPT-BF [6], and RaCE [7].

6.1. Storage Utilization

The storage utilization for all techniques on a range from 2 million to 20 million CCN names has been presented in figure 4. It can be concluded that the storage utilization for all techniques is growth proportionally with the growing number of CCN names. BF-QF FIB technique requires the minimum storage among all the other techniques because it depends on utilizing both BF and QF and both of these data structures require less storage. The greatest storage utilization is needed by NPT-BF because the need to use three data structures: name prefix trie, counting BF and finally a hash table.

![Figure 4. Storage utilization for N-FIB, NPT-BF, RaCE, and BF-QF FIB.](image)

6.2. Insertion Rate

In this experiment, we inserted a range from 2 million to 10 million CCN names into all techniques and registered the time of insertion. Our suggested technique has the lowest insertion time compared to the other three techniques. The insertion process in BF-QF FIB is performed quickly because our utilization of only two hash functions. The values of these two hash functions have been utilized in the insertion inside Bloom filter whereas in Quotient filter we utilized the same first hash function that has been utilized in Bloom filter. RaCE has extremely complicated structure which make the insertion operation consumes the highest time among all the other techniques. The result of this experiment has been showed in figure 5.
6.3. Search Rate
To measure the search rate, we made an experiment by performing a search process on a range from 2 million to 10 million CCN names and register the time taken by all the techniques. Figure 6 presents the result of this experiment. It can be concluded that all techniques have better search rate than the insertion rate and from all of them BF-QF FIB technique has the better search rate. RaCE has the worst search rate and, again, because of its complicated structure. NPT-BF technique show better search performance than RaCE because it depends on querying Bloom filter and a hash table and do not need to search the name prefix trie.

Figure 6. Search rate for N-FIB, NPT-BF, RaCE, and BF-QF FIB.
7. Conclusion
In this paper, we suggested a novel CCN name matching technique named BF-QF FIB to minimize the insertion/search time and to decrease the storage overhead in designing forwarding information base (FIB) table in content centric networks. The suggested technique is based on employing two main data structures: Bloom and Quotient filters. Our choice for these two kinds of data structures is based on their minimum storage utilization and their ability to accomplish storing/search operations softly and quickly. Moreover, both of them have simple structure that do not need a specific hardware to be implemented. Also, to speedup entire process, we utilized one of the two hash functions of Bloom filter to access the Quotient filter in order to decrease the time taken in inserting/searching CCN names in QF. Our evaluation demonstrated that our suggested technique has high performance and it can be utilized efficiently in designing and implementing a FIB table in CCN routers.

8. References

[1] J. H. Mun and H. Lim, “Cache sharing using bloom filters in named data networking,” Journal of Network and Computer Applications, vol. 90, pp. 74–82, Jul. 2017.
[2] M. Alhisnawi and M. Ahmadi, “Detecting and Mitigating DDoS Attack in Named Data Networking,” Journal of Network and Systems Management, May 2020.
[3] R. Shubbar and M. Ahmadi, “Efficient name matching based on a fast two-dimensional filter in named data networking,” International Journal of Parallel, Emergent and Distributed Systems, vol. 34, no. 2, pp. 203–221, Aug. 2017.
[4] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen, “Scalable name lookup in NDN using effective name component encoding,” In Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on, pp. 688-697. IEEE, 2012.
[5] Y. Wang, T. Pan, Z. Mi, H. Dai, X. Guo, T. Zhang, B. Liu, and Q. Dong, “NameFilter: Achieving fast name lookup with low memory cost via applying two-stage Bloom filters,” In INFOCOM, 2013 Proceedings IEEE, pp. 95-99. IEEE, 2013.
[6] J. Lee, M. Shim, and H. Lim, “Name prefix matching using Bloom filter pre-searching for content centric network,” Journal of Network and Computer Applications 65, 2016: 36-47.
[7] D. Saxena, V. Raychoudhury, C. Becker, and N. Suri, “Reliable Memory Efficient Name Forwarding in Named Data Networking,” EUC, 2016.
[8] D. Saxena and V. Raychoudhury, “N-FIB: Scalable, memory efficient name-based forwarding,” Journal of Network and Computer Applications, vol. 76, pp. 101-109, Dec. 2016.
[9] S. Kumar, R. Tiwari, M. S. Obaidat, N. Kumar, & K.F. Hsiao, “CPNDD: Content Placement Approach in Content Centric Networking,” In ICC 2020-2020 IEEE International Conference on Communications (ICC) (pp. 1-6).
[10] T. Shigeyasu and A. Sonoda, “Detection and mitigation of collusive interest flooding attack on content centric networking,” International Journal of Grid and Utility Computing, vol. 11, no. 1, p. 21, 2020.
[11] R. Hou, L. Zhang, T. Wu, T. Mao, and J. Luo, “Bloom-filter-based request node collaboration caching for named data networking,” Cluster Computing, vol. 22, no. S3, pp. 6681–6692, Mar. 2018.
[12] R. Shubbar and M. Ahmadi, “A Filter-Based Design of Pending Interest Table in Named Data Networking,” Journal of Network and Systems Management, vol. 27, no. 4, pp. 998–1019, Mar. 2019.
[13] M. Alhisnawi, and M. Ahmadi, “QCF for deep packet inspection,” IET Networks, vol. 7, no. 5, pp. 346–352, Sep. 2018.
[14] R. Shubbar and M. Ahmadi, “Fast 2D filter with low false positive for network packet inspection,” IET Networks, vol. 6, no. 6, pp. 224–231, Nov. 2017.
[15] M. Al-hisnawi, and M. Ahmadi, “Deep packet inspection using Cuckoo filter,” 2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT), Mar. 2017.

[16] M. Tortelli, G. Piro, L. A. Grieco, & G. Boggia, “On simulating Bloom filters in the ndnSIM open source simulator,” Simulation Modelling Practice and Theory, 52, 149-163, 2015.

[17] M. A. Bender et al., “Don’t thrash: How to cache your hash on flash,” Proc. VLDB Endowment, vol. 5, no. 11, pp. 1627–1637, 2012.

[18] M. Alhisnawi and M. Ahmadi, “Deep Packet Inspection Using Quotient Filter,” IEEE Communications Letters, vol. 20, no. 11, pp. 2217–2220, Nov. 2016.

[19] S. Garg, A. Singh, K. Kaur, S. Batra, N. Kumar, and M. S. Obaidat, “Edge-Based Content Delivery for Providing QoE in Wireless Networks Using Quotient Filter,” 2018 IEEE International Conference on Communications (ICC), May 2018.

[20] A. Afanasyev, I. Moiseenko, and L. Zhang, “ndnSIM: NDN simulator for NS-3,” University of California, Los Angeles, Technical Report 4, 2012.

[21] Schnurrenberger, Urs, “The Content Name Collection,” CNC. Accessed June 26, 2019, http://www.icn-names.net/.