Packet Cache Network Function for Peer-to-Peer Traffic Management with Bloom-Filter Based Flow Classification

Kengo Sasaki*†
* Toyota Central R&D Labs., Inc., Aichi, Japan
Aichi, Japan
Email: sasaki-ken@mosk.tytlabs.co.jp

Akihiro Nakao†
† The University of Tokyo
Tokyo, Japan
Email: {sasaki,nakao}@nakao-lab.org

***Abstract*—Following the emergence of peer-to-peer (P2P) applications, millions of computer users have used P2P systems to search for desired content. P2P traffic is known to be highly redundant because of its inherent self-scaling characteristics, which means that file sharing is performed more efficiently when more users exchange the same content. To remove redundant P2P traffic, we have proposed a method to control the P2P traffic through a packet-level data cache that acts as a network function at the edge of the Internet service provider (ISP) networks [1]. However, our previous method involves high levels of memory consumption.**

Software-defined networking (SDN) and network functions virtualization (NFV) are representative trends in network softwarization that may lower the barrier to deployment of network management functions that are considered to be useful but are difficult to actually implement and deploy.

In this paper, we propose a new flow classification for P2P that uses a queue Bloom filter (QBF) to reduce the memory consumption of the P2P cache. The QBF is a time series queue that manages Bloom filters and it can remove inserted Bloom filter elements without generating false positives. If the router can confirm that P2P flows are carrying duplicate contents using QBF, it then begins to cache the duplicate content. Our analysis shows that the proposed method reduces memory consumption to 67% and improves the P2P cache hit ratio by 4% when compared with the previous approach, while its performance in removing redundancy from the P2P traffic is degraded by only 14%. In addition, we discuss the implementation and deployment of the proposed system at the edge routers of ISP networks by applying SDN and NFV.

***Index Terms*—P2P, Overlay Network, Cache System, Bloom filter, SDN, NFV**

I. Introduction

Following the emergence of peer-to-peer (P2P) applications, such as BitTorrent [2] and PPTV [3], millions of computer users have used P2P systems to search for desired content. P2P networks have also demonstrated considerable potential to become a popular network tool for use not only in file sharing but also in video streaming [4] or to act as contents delivery networks (CDNs) [5] on the Internet. While there is a prediction that P2P traffic for file sharing applications will not increase in the future [6], it still represents today a significant fraction of the Internet traffic in Asia at present [7]. In particular, BitTorrent accounts for 48% of fixed access upstream traffic in Asia. However, most P2P-based systems are designed without any awareness of the network topology and generate a large amount of Internet service provider (ISP) traffic that transports the same content repeatedly.

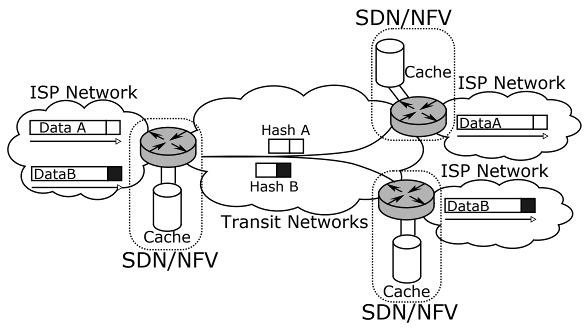


Fig. 1. Overview of packet-level data cache. The proposed system can be implemented and deployed at the edge routers by applying SDN and NFV.

To remove the redundancy from P2P traffic, we have proposed a method to control P2P traffic through a packet-level data cache located on the router [1], [8]. When using the previous method, more than 95% of the packets are compressed to less than 35 % in bytes of their original size. However, the previous method requires high memory consumption. It is preferable to reduce the memory consumption to enable scaling of the system as the traffic volume increases. Therefore, the router needs to detect flows that include redundant content before caching the P2P traffic.

Network softwarization is an overall transformation trend for the design, implementation, deployment, management and maintenance of network equipment and network components through software programming, using software characteristics such as flexibility and rapidity of design, development and deployment throughout the life cycles of the network equipment and components [9]. Software-defined networking (SDN) [10] and network functions virtualization (NFV) [11] are representative trends in network softwarization that may lower the barriers to deployment of network management functions that are considered useful but are difficult to actually implement

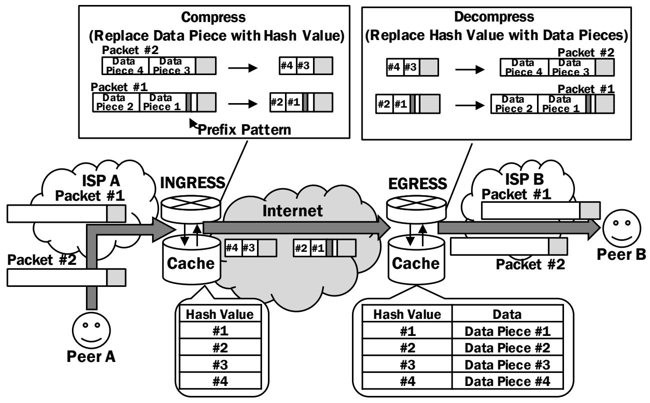


Fig. 2. Detailed view of packet-level data cache.

and deploy. One example of such a network management function is packet cache, often observed in P2P applications.

In this paper, we propose a new flow classification for P2P using the queue Bloom filter (QBF) to reduce the P2P cache memory consumption. The QBF is a time series queue that manages Bloom filters [12] and can remove elements of inserted Bloom filters without generating false positives. When the router receives content, the router memories the contents on QBF. By referring to the QBF, if the router can confirm that the P2P flows are transporting duplicate content, it then begins to cache the content of these flows. In addition, we discuss whether the proposed system can be implemented and deployed at the edge routers of ISP networks by applying SDN for flow classification and NFV for the packet cache function, as shown in Fig. 1.

Based on an analysis using a P2P traffic trace, we show that our proposed method reduces memory consumption to 67% of that for the previous method, and improves the P2P cache hit ratio by 4% compared with the previous approach, while the redundancy removal performance in the P2P traffic is degraded by only 14%.

This paper is organized as follows: Section II describes the background for the work in this paper. Section III proposes a flow classification using the QBF. Section IV evaluates and compares the performance of a simple P2P caching method and the proposed method using a BitTorrent traffic log. In Section V, we discuss the implementation and deployment using SDN and NFV. Section VI describes the related work in this field. Section VII provides brief conclusions.

II. Background

In this section, we explain both our previous cache method and the Bloom filter.

A. Previous work

Our previous method [1], [8] involves a network layer caching architecture that improves packet delivery for overlay P2P applications. Fig. 2 shows the packet-level data cache in detail. These cache systems are deployed at the edge routers, which are referred to here as INGRESS and EGRESS, between the ISPs and the Internet. When the edge router receives a P2P packet, it divides that packet into data pieces based on prefix patterns. If a data piece is transported first in EGRESS, EGRESS caches a mapping between a hash value and the data piece on the memory and then transports that hash value to INGRESS. When INGRESS receives the hash value, INGRESS caches the value. In turn, when INGRESS receives a P2P packet and constructs data pieces, INGRESS checks the cache. If the appropriate hash value exists in the cache of INGRESS, the data piece is replaced with that hash value. This replacement process is referred to as “compression”. By transporting these “compressed” packets between the ISPs, the P2P traffic is reduced. If the compressed packets are transported in EGRESS, then the hash values in the packet are replaced with the data pieces in the map of the router. This replacement process is referred to as “decompression”.

We built a prototype of the compression/decompression method and demonstrated that more than 95% of the packets are suppressed to less than 35 % of their size in bytes when compared with the original size, which required high levels of memory usage [1].

To reduce the memory consumption of the P2P cache, we introduce a new flow classification method based of use of the QBF on EGRESS. By selecting the flows that forward duplicate data pieces, EGRESS can cache only the required data pieces and reduce memory consumption.

B. Bloom Filter

The Bloom filter [12] is a bit array that is used to test whether or not an element is a member of a set. The key idea behind the Bloom filter is the use of hash functions. These hash functions map the elements to uniform random numbers within the size limit of the bit array.

For example, we consider a Bloom filter that is used to test whether or not an element y is a member of a set $S = \{x_1, x_2, \ldots, x_n\}$. The Bloom filter size is m bits, the number of hash functions is k and the hash values of an element $x_i \in S$ are $h_1(x_i), \ldots, h_k(x_i)$, which are in the range $1, \ldots, m$. The initial state of the Bloom filter is set to 0. First, the Bloom filter is inserted S. In the case of insertion of an element $x_i \in S$, the bits $h_j(x_i)$ are set to 1 for $1 \leqq j \leqq k$. A location value can be set to 1 multiple times, but only that first change has an effect. The check is then performed to determine whether y is a member of S or not. If all $h_i(y)$ are set to 1, then y is judged to be an element in S and we call a “ Hit”. Otherwise, y is clearly an element that is not part of S.

The Bloom filter can be used to represent sets with low memory consumption. However, the Bloom filter has two intrinsic problems. The first is false positive, i.e., the Bloom filter may judge an element y to be in S, even though it is not. Because these elements are inserted into the Bloom filter, it then has a high false positive ratio. The other problem is updating. The Bloom filter cannot remove elements without creating false negatives. To overcome these problems, we propose the QBF, which can remove the old elements without

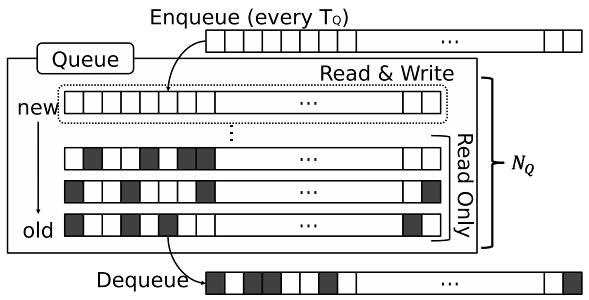


Fig. 3. Data structure of queue Bloom filter.

generation of false negatives, and use the QBF for flow classification.

III. Proposed Method

In this section, we propose the QBF and the flow classification process when using it.

A. Queue Bloom filter

The QBF is a time series queue that manages Bloom filters. Fig. 3 shows the QBF data structure. A Bloom filter is enqueued in the QBF at every T_Q. When the number of Bloom filters in the QBF is more than N_Q, the oldest Bloom filter in the QBF is then dequeued from the QBF. In the case of insertion of new elements into the QBF, the elements are inserted into the newest Bloom filter in QBF. In the case of checking of the new elements in the QBF, all Bloom filters in the QBF are checked.

A simple Bloom filter cannot remove any elements without generating false negatives. In contrast, QBF can remove old elements using a small step (T_Q) without generating false negatives. The update structures of the Bloom filter were proposed in [13]–[15] and we explain these structures in section VI.

B. P2P Flow Classification using the QBF

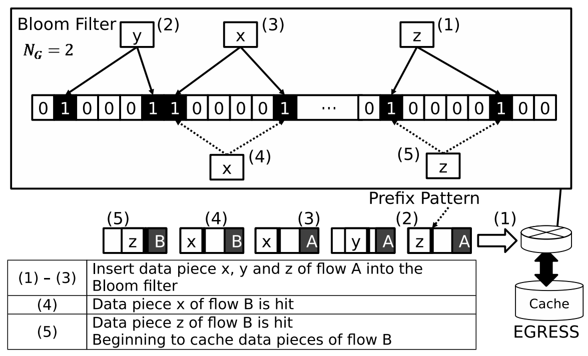
Our flow classification process selects flows that are transporting duplicate content. In this paper, the flow is composed of a source IP address/port and a destination IP address/port. Our flow classification is run on EGRESS, as shown in Fig. 2. EGRESS monitors each flow using a QBF.

Fig. 4 shows the P2P flow classification algorithm. When EGRESS receives P2P packets that have the required prefix patterns, EGRESS constructs data pieces and checks for data pieces that follow the prefix patterns. If a data piece is not a hit, EGRESS inserts that data piece into the QBF. Otherwise, EGRESS counts the number of hits in the flow. If the number of hits in the flow is more than N_{bf}, EGRESS assumes that the flow is transporting duplicate content and begins to cache the data pieces of the P2P flow.

Fig. 5 shows an example of the flow classification process. In this example, the data pieces are inserted into a simple Bloom filter rather than the QBF and N_{bf} is 2. EGRESS receives the packets in numerical order from (1) to (5). When EGRESS receives packets (1), (2) and (3), EGRESS checks the data pieces in the Bloom filter, which is in flow A, and the data pieces are not hit. Therefore, EGRESS inserts these data pieces into the Bloom filter. When EGRESS receives packet (4), EGRESS checks the data pieces in the Bloom filter, which is in flow B, and the data piece is a hit. Therefore, the number of hits of flow B is 1. When EGRESS receives packet (5), it checks the data pieces in the Bloom filter, which is in flow B, and the data piece is again a hit. Therefore, the number of hits in flow B is 2 ($= N_{bf}$) and EGRESS begins to cache the data pieces in flow B.

```
$C[*]$ := A hit counter for each flow
$N_{bf}$ := A threshold value of hits for beginning to cache
**loop**
  A P2P packet received
  $fl$ := A flow of the P2P packet
  **if** The P2P packet includes the prefix pattern **then**
    Construct data pieces
    $dp_{fx}$ := A data piece following the prefix pattern
    **if** EGRESS does NOT know flow $fl$ **then**
      $C[fl] := 0$
    **end if**
    **if** $dp_{fx}$ is hit **then**
      $C[fl]++$
      **if** $N_{bf} <= C[fl]$ **then**
        Beginning to cache
      **end if**
    **else**
      Insert $dp_{fx}$ into QBF
    **end if**
  **end if**
**end loop**
```

Fig. 4. P2P flow classification algorithm.



(1) – (3)	Insert data piece x, y and z of flow A into the Bloom filter
(4)	Data piece x of flow B is hit
(5)	Data piece z of flow B is hit Beginning to cache data pieces of flow B

Fig. 5. P2P flow classification using Bloom filter.

IV. Evaluation

To evaluate the proposed method, we simulate EGRESS with and without the flow classification. We analyze anonymized packet log data (100GB) from BitTorrent [2], which was recorded over 8 h by an ISP backbone router. We evaluate the data in terms of three metrics, which are *(i) memory consumption, (ii) cache hit count, and (iii) cache hit* *ratio*, and use three evaluation methods, which are *(i) Pool, (ii) QBF, and (iii) Pool + QBF*.

A. Metrics

"Memory consumption" is defined here as the "size of all cache entries". "Cache hit count" is defined here as the "the number of times that the router refers cache entries per unit time". "Cache hit ratio" is defined here as "the number of times that the router refers cache entries per unit time / the number of data pieces that EGRESS receives per unit time". The size of a cache entry is given by a map composed of a hash value and a data piece. The sizes of a data piece and the hash value are 1024 bytes and 16 bytes, respectively. Therefore, the size of a cache entry is 1040 bytes. We calculate the three metrics above once a minute using the log data. We set the unit time to be 3000 s, which is equal to both T_Q and T_{cache}, as described in IV-C.

B. Evaluation Methods

(1) The Pool method [1] caches all data pieces without flow classification. However, EGRESS cannot continuously cache all data pieces because it has finite memory resources. Therefore, EGRESS removes unused data pieces for T_{cache} s.

The Pool method can cache data pieces rapidly but consumes huge amounts of memory space. This means that the Pool method is suitable for small-sized flows.

(2) The QBF method caches the data pieces of the flows that are detected using the QBF. If the number of hits of the flow is greater than N_{bf}, EGRESS begins to cache the data pieces of the P2P flow. As per the Pool method, EGRESS removes the unused data pieces for T_{cache} s.

The QBF method can reduce memory consumption by selective caching. However, it takes time to select the flows and may possibly overlook data pieces that should be hits. This means that the QBF method is suitable for large-sized flows.

(3) The Pool + QBF method uses both the "Pool" and "QBF" methods. The Pool + QBF method prepares two memory regions, called PoolMemory and QBFMemory. When EGRESS receives P2P packets, it caches the data pieces in the PoolMemory (Pool method). Simultaneously, EGRESS executes P2P flow classification. If EGRESS detects a flow, EGRESS then begins to cache the data pieces of the flow in the QBFMemory (QBF Method). EGRESS removes the unused data pieces every T_{tmp} s from the PoolMemory and every T_{cache} s from the QBFMemory. T_{tmp} is a shorter time than T_{cache}

The "Pool + QBF" method thus uses the "Pool" method for small-sized flows and the "QBF" method for large-size flows.

C. Parameters

Table I shows the parameters used for the simulation. N_{bf} is a threshold at which addition of a cache entry of a flow begins. B_{bf} is the size of the Bloom filter and N_Q is the size of the Queue. Therefore, the QBF uses 1 GB of memory for flow classification. T_Q is the time interval required to enqueue and dequeue the QBF. T_{tmp} and T_{cache} are the lifetimes of the cache entries, i.e., EGRESS removes unused data pieces every T_{tmp} or T_{cache} s.

TABLE I
PARAMETERS FOR SIMULATION

Parameter	Value
N_{bf}	30
B_{bf}	2^{27}[byte]
N_Q	8
T_Q	3000 [s]
T_{tmp}	1000 [s]
T_{cache}	3000 [s]

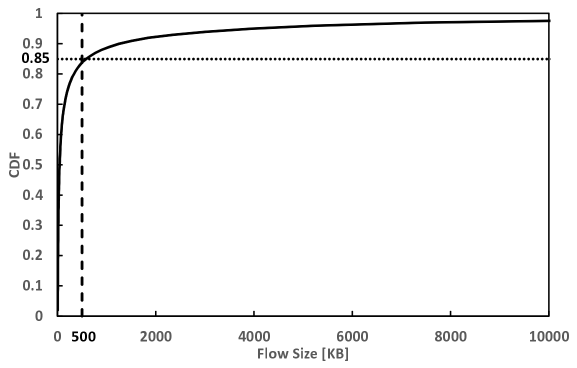
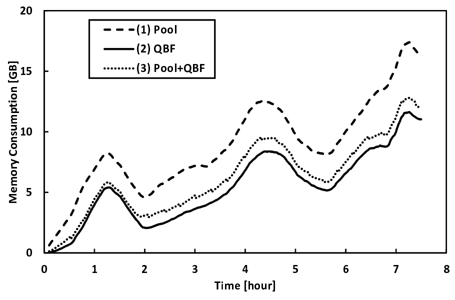


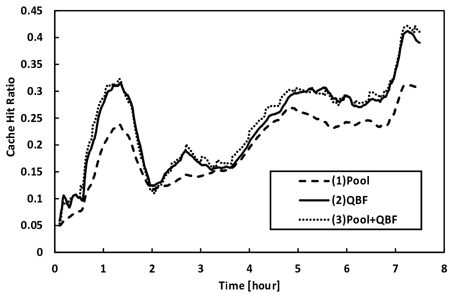
Fig. 6. Measured CDF as a function of flow size.

We determine these parameters using traffic statistics. Fig. 6 shows a cumulative distribution function (CDF) of the "flow size", which is defined as the byte count that a flow forwards, in the log data. The X-axis is the flow size (KB), and the Y-axis is the CDF. Fig. 6 shows that 85% of the flow is less than a flow size of 500 KB. In section IV-B, we stated that the Pool method and the QBF method are suitable for small-sized flows and large-sized flows, respectively. Therefore, we set the parameters such that the "Pool + QBF" method can cache flows of less than 500 KB with the PoolMemory and flows of more than 500 KB with the QBFMemory. In our log data, the median of the number of data pieces followed by a single prefix pattern is approximately 18. If N_{bf} is 30, the QBF method covers flows with sizes of more than $30 * 18 * 1024 = 540 \approx 500$KB. Therefore, we set N_{bf} to be 30.

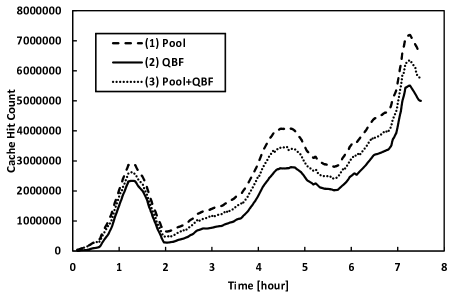
Fig. 7 shows the relationship between the flow size and the "flow life time", which is defined as the time required to receive all packets from a flow. The X-axis is the flow size, and the Y-axis plots the median flow life-time value for each flow size. We then decide T_{tmp}, T_Q and T_{cache} based on the flow life time. In Fig. 7, if the flow size is less than 500 KB, the flow life time is less than 1000 s. Thus, we set $T_{tmp} = 1000$s. However, when the flow size is large, it is difficult to determine an accurate median flow life-time value. Therefore, we set $T_Q = T_{cache} = 3000$s, because 95% of all flows have lifetimes of less than 3000 s.



(a) Memory consumption vs. time.



(b) Hit ratio vs. time.



(c) Hit count vs. time.

Fig. 8. Measured memory consumption, cache hit count and cache hit ratio as functions of time.

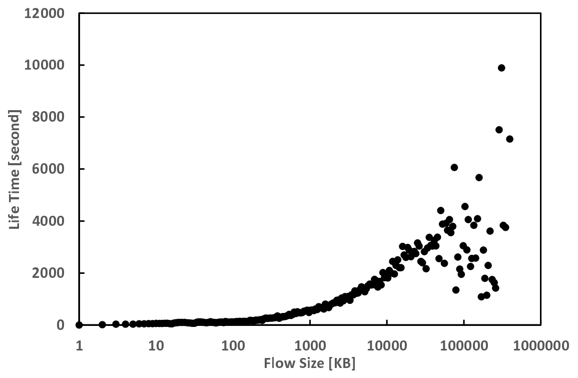


Fig. 7. Relationship between flow size and flow life time

D. *Results*

Fig. 8a shows the change in the memory consumption. The X-axis is time, and the Y-axis is memory consumption. The QBF and QBF + Pool methods cut the average memory consumption to 56% and 67% when compared with the Pool method.

Fig. 8b shows the change in the cache hit ratio. The X-axis is time, and the Y-axis is the cache hit ratio. The cache hit ratios of the QBF and QBF + Pool methods show a maximum improvement of 10 % and an average improvement of 4% when compared with the Pool method.

Fig. 8c shows the change in the cache hit count. The X-axis is time, and the Y-axis is the cache hit count. While both the QBF and QBF + Pool methods can reduce memory consumption when compared with the Pool method, they also reduce the cache hit count because of their low memory consumption. The Pool method contains all cache entries at a certain point, and the gap from the Pool method to the QBF or QBF + Pool methods is equal to the missed cache hit count. Therefore, the QBF method overlooks 29% of the cache hit count when compared with the Pool method. However, by using the PoolMemory, the QBF + Pool method can suppress the missing of cache hits by 14%.

From the above analysis, we have shown that the QBF method adds cache entries selectively and the QBF + Pool method effectively covers the cache hit count using PoolMemory and we thus consider the QBF and QBF + Pool methods to be better than the Pool method. Comparison of the QBF method with the QBF + Pool method, shows that the QBF method has a lower cache hit count, cache hit ratio and memory consumption. This means that the QBF + Pool method covers small flows and the QBF method does not. Therefore, if the users need to cover small flows, they should use the QBF + Pool method. Alternatively, if the users only need to cover large flows, they should use the QBF method.

V. Implementation and Deployment Consideration

In this section, we will briefly discuss the implementation and deployment of the proposed method as a feasibility study.

In this paper, we have proposed a new packet classification method using Bloom filters to selectively and efficiently perform packet caching to eliminate redundant traffic, particularly that observed in P2P traffic. Both the classification and packet caching functions have been considered to be difficult to implement and deploy, because the existing switches and routers, and even the emerging SDN OpenFlow switches are not equipped with flexibly programmable data plane elements.

However, the emerging concept of network softwarization enables flexibility and rapidity in the design, development and deployment of network equipment and components throughout the life cycle [9], and include the emerging concept of the *software-defined data plane* [16] to enhance SDN data plane functionality by software programming; this, combined with the advent of NFV to enable various network functions, should lower the barriers to the implementation and deployment of the proposed network management method.

In our approach, we start from the application view point, i.e., we exercise *application-driven thinking*. Therefore, to enable redundancy elimination in network management for P2P traffic, we define a new Bloom-filter based flow classification along with a new packet cache network function. Because of the basis of this approach, we have not yet fully investigated how to map this proposal into the current harmonization architectures of SDN and NFV [16] However, it is clear to us that *to enable a new network management for redundancy elimination such as that proposed in this paper, we must at least enhance the data plane elements of the current SDN* *architecture and embed the packet cache network functions as part of the SDN data plane or in the NFV network functions.*

One part of our immediate future work is the design of an overall network management strategy, including the implementation of our proposed traffic classification and packet cache in the software-defined data plane and its southbound interface, using deeply programmable network nodes such as FLARE [16]. We also plan to design a logically centralized controller over multiple network nodes that is equipped with these data plane enhanced SDN components. In the required design, harmonization of SDN and NFV, or even further unification of the SDN and NFV concepts may be necessary for softwarized network management.

VI. Related Work

The Bloom filter [12] has already had various applications in SDN [17], content-centric networks [18], P2P networks [19] and various network applications [20]. However, these usages of Bloom filters in specific applications are different to our proposed method.

An updated Bloom filter structure has been proposed to detect data duplication in data stream with low false positive ratios [13]–[15]. The stable Bloom filter (SBF) [13] is based on the counting Bloom filter (CBF) and consists of a counter array, rather than a bit array. When an element is inserted into the SBF, the k selected counters of the element are set to the maximum counter value, and the counter values, which are randomly selected, are decreased by 1 to reduce the false positive ratio. Although, this leads not only to false positives but also to false negatives, it stabilizes the false positive ratio and the false negative ratio [13]. A decaying Bloom filter (DBF) [14] is another Bloom filter that is based on a CBF. The SBF reduces randomly selected counter values in the case of insertion of elements. In contrast, when an element is inserted into the CBF, all counter values are decreased by 1. As a result, the DBF can avoid false positives. The time decaying Bloom filter (TDBF) [15] is similar to the DBF. All counters are decreased by 1 at constant time intervals, and not before addition of elements. These update structures can check the Bloom filter with lighter loads than the QBF. However, the QBF has a lower false positive ratio than that in related works.

VII. Conclusion

In this paper, we have proposed a flow classification method using a QBF for P2P caching as the target of the network management functions. The QBF is a Bloom filter equipped with an updated structure using queues and it can remove older elements without generating false positives. Our proposed system can reduce the memory consumption of the P2P cache and achieves high cache hit ratios by detecting P2P flows that are transporting duplicate content. Based on our analysis of the P2P traffic trace at an ISP, we have shown that the QBF + Pool method reduces memory consumption to 67% and improves the P2P cache hit ratio by 4% when compared with the previous approach, while the redundancy removal performance in P2P traffic is degraded by only 14%.

Our proposed future work is as follows. We intend to implement an actual system using SDN/NFV for a feasibility study. First, we will design the overall network management including implementation of our proposed traffic classification and packet caching method in software-defined data plane and its southbound interface, using deeply programmable network nodes such as FLARE. Second, we plan to design a logically centralized controller over multiple network nodes that are equipped with these data plane-enhanced SDN components. Finally, we plan to deploy the proposed method on a virtual network and evaluate its performance.

References

[1] A. Nakao, K. Sasaki, and S. Yamamoto, "A remedy for network operators against increasing P2P traffic: Enabling packet cache for P2P applications," *IEICE transactions on communications*, vol. 91, no. 12, pp. 3810–3820, 2008.

[2] BitTorrent. [Online]. Available: http://www.bittorrent.com/

[3] PPTV. [Online]. Available: http://www.pptv.com/

[4] N. Magharei, R. Rejaie, I. Rimac, V. Hilt, and M. Hofmann, "ISP-friendly live P2P streaming," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 1, pp. 244–256, 2014.

[5] B. Maggs, "A first look at a commercial hybrid content delivery system," in *Keynote presentation at 15th IEEE Global Internet Symposium*, 2012.

[6] Cisco, "Cisco Visual Networking Index: Forecast and methodology, 2013–2018," *CISCO White paper*, 2014.

[7] Sandvine. (2015) Global Internet Phenomena Asia-Pacific & Europe. [Online]. Available: http://www.sandvine.com/trends/global-internet-phenomena/

[8] S. Yamamoto and A. Nakao, "P2P packet cache router for network-wide traffic redundancy elimination," in *Computing, Networking and Communications (ICNC), 2012 International Conference on*. IEEE, 2012, pp. 830–834.

[9] (2015) FG IMT-2020: Report on Gap Analysis (presented at SG13 meeting). ITU-T.

[10] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, 2009.

[11] (2012) Network functions virtualization Introductory White Paper. ETSI.

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[13] F. Deng and D. Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, 2006, pp. 25–36.

[14] H. Shen and Y. Zhang, "Improved approximate detection of duplicates for data streams over sliding windows," *Journal of Computer Science and Technology*, vol. 23, no. 6, pp. 973–987, 2008.

[15] K. Cheng, L. Xiang, and M. Iwaihara, "Time-decaying bloom filters for data streams with skewed distributions," in *Research Issues in Data Engineering: Stream Data Mining and Applications, 2005. RIDE-SDMA 2005. 15th International Workshop on*. IEEE, 2005, pp. 63–69.

[16] A. Nakao, "Software-Defined Data Plane Enhancing SDN and NFV," *IEICE Transactions on Communications*, vol. E98-B, no. 1, pp. 12–19, 2015.

[17] C. A. Macapuna, C. E. Rothenberg, and M. F. Magalhaes, "In-packet bloom filter based data center networking with distributed openflow controllers," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 584–588.

[18] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "Dipit: A distributed bloom-filter based pit table for ccn nodes," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. IEEE, 2012, pp. 1–7.

[19] J. Risson and T. Moors, "Survey of research towards robust peer-to-peer networks: search methods," *Computer networks*, vol. 50, no. 17, pp. 3485–3521, 2006.

[20] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 1, pp. 131–155, 2012.