Scalable Data Center Multicast using Multi-class Bloom Filter

Dan Li*, Henggang Cui*, Yan Hu†, Yong Xia †, Xin Wang‡

*Computer Science Department of Tsinghua University †NEC Research China ‡Stony Brook University
tolidan@tsinghua.edu.cn, cuihenggang@gmail.com, {hu_yan,xia_yong}@nec.cn, xwang@ece.sunysb.edu

***Abstract*—Multicast benefits data center group communications in saving network bandwidth and increasing application throughput. However, it is challenging to scale Multicast to support tens of thousands of concurrent group communications due to limited forwarding table memory space in the switches, particularly the low-end ones commonly used in modern data centers. Bloom Filter is an efficient tool to compress the Multicast forwarding table, but significant traffic leakage may occur when group membership testing is false positive.**

To reduce the Multicast traffic leakage, in this paper we bring forward a novel multi-class Bloom Filter (MBF), which extends the standard Bloom Filter by embracing element uncertainty. Specifically, MBF sets the number of hash functions in a per-element level, based on the probability for each Multicast group to be inserted into the Bloom Filter. We design a simple yet effective algorithm to calculate the number of hash functions for each Multicast group. We have prototyped a software based MBF forwarding engine on the Linux platform. Simulation and prototype evaluation results demonstrate that MBF can significantly reduce Multicast traffic leakage compared to the standard Bloom Filter, while causing little system overhead.

I. Introduction

Data center is the key infrastructure for cloud computing. Servers, with a scale of tens of thousands, or even hundreds of thousands, are interconnected to run distributed computations for cloud services. As the network bandwidth has become a bottleneck for data center distributed computations [1], recently many advanced network topologies are proposed to improve the network capacity, such as DCell [2], BCube [3], Fat-Tree [4], VL2 [5] and FiConn [6]. These new proposals use much more switches to form the network infrastructure than the current practice of tree topology.

Besides increasing the network capacity of data centers, it is equally important to make efficient use of the available network bandwidth. In this paper we focus on group communication, which is commonly needed to support data center computations, such as redirecting search queries to indexing servers [7], replicating file chunks in distributed file systems [8], [9], as well as distributing executable binaries to a set of servers for Map-Reduction like cooperative computations [8], [1], [10]. Network-level Multicast is a natural choice to support group communications for both saving network bandwidth and increasing application throughput. Although Multicast is not widely deployed in the Internet due to management and pricing issues, the controlled environment of data center provides a good opportunity for its resurgence.

The work is supported by the National Basic Research Program of China (973 Program) under Grant 2011CB302900 and 2009CB320501, and the National Natural Science Foundation of China (No.61170291, No.61133006).

The technical trend for modern data center design is to use a large number of low-end switches for server interconnection. The space of the expensive and power-hungry fast memory in this kind of switches, such as SRAM, is usually narrow to control the economical cost, especially considering the link speeds in data centers are increasing rapidly. Hence, a significant challenge in the forwarding plane of data center switches is how to maintain a potentially large number of forwarding entries in the small memory space. Compared with Unicast, it is especially difficult to aggregate Multicast forwarding entries due to two reasons [18]. First, Multicast addresses are logical identifiers without any topological information. Second, each Multicast forwarding entry maintains an outgoing interface set rather than a single outgoing interface, so it is less probable for different forwarding entries holding common forwarding rules. Previous investigation showed that typical access switches can contain only 70∼1500 Multicast group states [11].

Bloom Filter is widely used to compress forwarding table and help realize scalable forwarding in recent years, with the cost of traffic leakage resulting from the false-positive forwarding. For Multicast forwarding, we can let each switch interface maintain a Bloom Filter, which encodes all the Multicast groups on the interface. When the switch receives a Multicast packet, all the Bloom Filters on its interfaces are checked to determine whether to forward or not. In order to design *efficient* Bloom Filters to control the traffic leakage, we usually need to know the exact number of Multicast groups on each switch interface. But this is quite difficult in data center networks. First, Multicast tasks dynamically start and end, and thus the number of Multicast tasks in data centers is not fixed. Second, for a certain Multicast task, the group membership distribution is even not determined, considering the flexibility in virtual machine (VM) placement on physical servers as well as VM migration during computation.

To solve the problem above, one possible choice is to dynamically update the Bloom Filter based Multicast forwarding engine in the switch data plane to accommodate group join and leave. But the overhead is quite high. Frequently updating the forwarding engine at runtime may cause forwarding pauses or even errors. We instead choose to optimize the Bloom Filter setup for some steady-state Multicast communication loads. We then run the forwarding engine for a relatively long period of time, until when the Multicast communication load changes significantly. It is feasible to infer some network-

wide conditions for such steady states, such as the number of Multicast groups and the probability for each group to join a switch interface. Hence, we can set the optimal Bloom Filter parameters based on the probabilisticly expected number of Multicast groups to join the interface. But the standard Bloom Filter cannot well embrace this kind of membership uncertainty in minimizing the traffic leakage.

We propose a novel multi-class Bloom Filter, or MBF, to solve the element uncertainty problem. MBF extends the standard Bloom Filter by choosing the number of hash functions at a per-element level, based on the probability of each element to be inserted into the Bloom Filter. Then we apply MBF into the Bloom Filter forwarding engine on data center switches to control the Multicast traffic leakage. Since in practice the number of Multicast groups in the data center can be very large, we develop a simple yet effective algorithm to calculate the number of per-group hash functions. We divide the Multicast groups into multiple slots and use a low-complexity *sorted enumeration* method to get the optimal number of hash functions for each slot.

Simulations in typical data center networks show that MBF based Multicast forwarding can significantly reduce the traffic leakage ratio compared with the standard Bloom Filter, especially when the group sizes are various, the Bloom Filter size is narrow, or the network size is large. Even if we cannot accurately estimate the number of Multicast groups in the steady state, MBF still outperforms the stand Bloom Filter when the estimation falls into a reasonable range.

We have prototyped a software based MBF forwarding engine for data center Multicast on the Linux platform. Experiments running on our testbed demonstrate that the MBF forwarding engine brings neglectable CPU overhead even when it forwards packets at full network speed. The packet loss ratio is also comparable with that in traditional routing entry based Multicast forwarding.

The rest of this paper is organized as follows. Section II discusses the background and design challenges. Section III presents the design of MBF and its application in scalable data center Multicast. Section IV and Section V evaluate the performance of the MBF based Multicast forwarding by simulations and experiments, respectively. Section VI introduces the related work. Section VII concludes the paper.

II. Background and Design Challenges

In this section, we discuss the background of Bloom Filter based Multicast forwarding as well as the design challenges.

A. Bloom Filter based Multicast Forwarding

Due to the growing forwarding entries on routers/switches and the high cost of fast memory, recently Bloom Filter has been adopted to achieve scalable Multicast forwarding, in both the Internet [12], [13] and the data center networks [14], [15].

A standard Bloom Filter works as follows. Suppose we have a Bloom Filter with the size of m bits and there are n elements to insert. Initially, all bits in the Bloom Filter are set to 0. k hash functions are used to hash each element to k bits in the Bloom Filter, and the hashed bits are set as 1. After inserting all the n elements, the probability that one bit in the Bloom Filter is still 0 is $(1-\frac{1}{m})^{nk}$. The probability that one bit has been set to 1 is hence $1-(1-\frac{1}{m})^{nk}$. When checking whether an arriving element is in the Bloom Filter or not, there can be false positive since all the hashed bits can be set as 1 by other elements. But there will be no false negative. The false positive probability, fp, for an incorrectly matched element is $fp = [1-(1-\frac{1}{m})^{nk}]^k$. To minimize the false positive probability, we need to set $k = \ln 2 * \frac{m}{n}$, and the minimum value of the false positive probability is $(\frac{1}{2})^k$.

There are two types of Bloom Filter based Multicast forwarding schemes, namely, in-switch Bloom Filter and in-packet Bloom Filter. For in-switch Bloom Filter based Multicast forwarding, each interface in the switch uses a Bloom Filter to maintain the Multicast groups it joins. The traffic overhead comes from the false positive forwarding in some interfaces. Each interface can independently set the parameters of its Bloom Filter to minimize the false positive probability and thus control the traffic leakage.

In contrast, in-packet Bloom Filter encodes the Multicast tree information into a Bloom Filter field carried in the packet header, and eliminates the necessity of Multicast forwarding entries in switches. However, the traffic overhead of in-packet Bloom Filter based Multicast forwarding not only comes from the false positive forwarding by switches, but also includes the Bloom Filter field in the packet. Previous studies showed that given the group size, there is an optimal Bloom Filter length to minimize the bandwidth overhead [15]. For a certain group, the Bloom Filter parameters for it should be set consistently on the switches for forwarding correctness.

B. Design Challenges

Modern data center networks tend to use abundant low-end switches to interconnect a massive number of servers for a lower cost and higher network capacity. Since the space of fast memory to maintain the forwarding entries in such kind of low-end switches is quite limited, it is important to design scalable Multicast forwarding scheme to embrace a potentially large number of Multicast groups. We prefer in-switch Bloom Filter rather than in-packet Bloom Filter for data center Multicast forwarding for two reasons. First, in-switch Bloom Filter requires updating the switches only, while in-packet Bloom Filter needs to simultaneously update both the switches and servers. The deployment is simpler if we make fewer changes on the existing systems. Second, the bandwidth overhead of in-switch Bloom Filter only comes from the false positive forwarding, while that of in-packet Bloom Filter is comprised of both the false positive forwarding and the in-packet Bloom Filter field. Hence, we have more room to optimize the traffic overhead using in-switch Bloom Filter.

However, there are also several challenges to design bandwidth-efficient in-switch Bloom Filter for data center Multicast. First, false positive forwarding by Bloom Filter can cause loops. Second, Multicast groups and members dynamically join and leave, but the standard Bloom Filter does not support the deletion of an element. Third, when the number of groups on a switch interface changes, we need to

recalculate the number of hash functions and reset the Bloom Filter bits, so as to minimize the traffic leakage. But it requires extra memory to maintain the exact group information on the interface for rehashing. Besides, the cost of updating the forwarding engine at runtime is very high, which can cause forwarding pauses or even errors.

To address the first challenge, we can leverage the multi-stage graph feature of recently proposed data center networks, using the similar approach as in [15]. In multi-stage graphs, there do not exist two neighboring nodes which have the same distance to a third node. Therefore, we can let a switch always forward packets to its neighboring nodes which are more distant from the source server of the Multicast group. Following this rule, the falsely forwarded packets will be dropped within at most d hops, where d is the diameter of the network. Hence no loop will be formed during the packet forwarding.

For the second challenge, there are two candidate solutions. The first approach is to employ the counting Bloom Filter [23] other than the standard Bloom Filter in the fast memory. The use of the counting Bloom Filter, however, will require a larger memory space. The second approach is to adopt counting Bloom Filter in the slow memory to maintain the Multicast membership, while still using standard Bloom Filter in the fast memory for fast-path forwarding. It follows the idea in BUFFALO [16], which targets for Bloom Filter based Unicast forwarding. Throughout this paper, we still make the discussion based on the standard Bloom Filter, but it can be easily extended to the counting Bloom Filter to support the element deletion.

The primary goal of this paper is to tackle the third challenge, i.e., setting optimal Bloom Filter parameters, in particular the number of hash functions, to minimize the traffic leakage. In the standard Bloom Filter, the optimal parameter setting requires knowing the exact number of Multicast groups on the interface. Since we do not want to frequently rehash Multicast groups at runtime upon group membership change, we choose to optimize the Bloom Filter setup for some steady states. In these steady states, we can estimate the Multicast communication load in the data center, i.e., the number of Multicast groups as well as the group size distribution. We run the forwarding engine designed for a steady state for a relatively long period of time, until when the Multicast communication load changes significantly. But even in the steady states, the exact locations of the Multicast members are not determined. Hence, we still cannot get the deterministic group membership on a certain switch interface. To embrace the membership uncertainty on a switch interface, we propose a novel multi-class Bloom Filter, or MBF, which will be presented in the next section.

III. Design

In this section we design multi-class Bloom Filter to realize scalable data center Multicast.

A. Multi-class Bloom Filter

In the standard Bloom Filter, a set of deterministic elements are inserted into the Bloom Filter, and a consistent number of hash functions are used for mapping elements to the Bloom Filter bits. But we consider a different scenario, i.e., when the elements to join the Bloom Filter cannot be exactly determined before designing the Bloom Filter. Specifically, suppose there are totally N elements in the element space and the Bloom Filter length is m. Each element i has a probability of p_i to be inserted into the Bloom Filter, and we call it the *presence probability* of i. The expected number of elements in the Bloom Filter is thus $\sum_{i=0}^{N-1} p_i$.

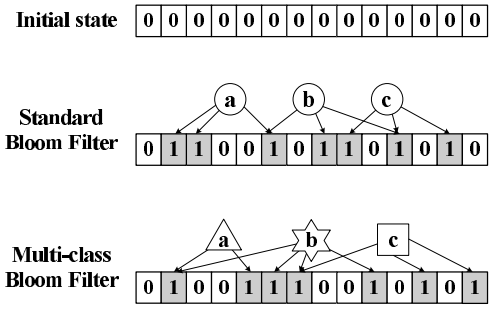

Fig. 1. Illustration of MBF. Different numbers of hash functions are set for different elements.

Each element i has a probability of $1 - p_i$ not to join the Bloom Filter. For the element i which lies beyond the Bloom Filter, it has a *false probability*, f_i, to be false-positively matched because its hashed bits are filled by other elements. The expected number of falsely matched elements, $E(fn)$, should be $E(fn) = \sum_{i=0}^{N-1}(1 - p_i)f_i$. It suggests that when the presence probability of an element is higher, the impact of its false probability on the expected number of falsely matched elements is lower. Intuitively, we can set different number of hash functions for each element to minimize the expected number of falsely matched elements. We call this kind of Bloom Filter *multi-class Bloom Filter (MBF)*.

In MBF, we independently choose a set of hash functions, H_i, for element i ($0 \leq i < N$). Assume there is $\|H_i\| = k_i$. If we insert element i into the Bloom Filter, all the hash functions from H_i are used to map to k_i bits in the Bloom Filter. When checking whether an element j is in the Bloom Filter, we also take the hash functions from H_j. Fig. 1 shows an example to differentiate the standard Bloom Filter and MBF. In the standard Bloom Filter, all the three elements, a, b and c, use the same three hash functions to map to the Bloom Filter bits. But in MBF, element a has two hash functions, element b has four hash functions, and element c has three hash functions for mapping.

The false probability for the element i, f_i, can thus be expressed as

$$f_i = [1 - (1 - \frac{1}{m})^{\sum_{j=0}^{N-1} p_j * k_j}]^{k_i}$$

B. Applying MBF into Data Center Multicast

As discussed in Section II, even when we design the optimal Bloom Filter for a switch interface for the steady-state data center Multicast communications, it is still challenging since we have no knowledge of the exact number of Multicast

groups on the interface. But we can apply MBF to embrace the membership uncertainty problem, given we can get the total number of Multicast groups in the data center and their presence probabilities to join the interface.

It is feasible to estimate the number of Multicast groups within the data center in a certain steady state. A data center usually runs typical group applications, such as file chunk replication, Map-Reduce alike distributed computation, as well as OS/application upgrading. The concurrent number of this kind of applications in the steady state should be relatively stable so as to guarantee the task finish time. Besides, we can get the number from historical statistics. We will evaluate the case when the estimation is inaccurate in Section IV.

As for estimating the presence probability of a group on a certain switch interface, we can take both the group size and the data center topology into consideration. For many data center Multicast tasks, the group sizes can be roughly pre-defined, e.g., the number of replicas for file chunk, or the number of mappers and reducers for a Map-Reduce computation tasks. Most data center networks have regular topologies, which further help to estimate the probability of a group joining an interface. We take the Fat-Tree network composed of k-port switches as an example. The total number of servers is $Z = \frac{k^3}{4}$. There is a group with r members. Now we estimate the probability that this group joins a downside interface of an edge-level switch. The presence probability should be $\frac{r}{Z} = \frac{4r}{k^3}$, if we assume the group members are randomly distributed among all servers.

Therefore, we can use MBF for optimizing the Bloom Filter forwarding engine on switch interfaces. Our goal is to limit the traffic leakage. We consider a switch interface with n groups joining it and their traffic volumes are t_0, t_1, ..., t_{n-1}, respectively. However, there are also q groups false-positively matched on this interface by the Bloom Filter, and their traffic volumes are v_0, v_1, ..., v_{q-1}, respectively. Hence, we can define the *traffic leakage ratio* on this interface, lr, as follows.

$$lr = \frac{\sum_{i=0}^{q-1} v_i}{\sum_{i=0}^{n-1} t_i}.$$

In this paper, we set the traffic volumes equal for all the Multicast groups [1]. As a consequence, the traffic leakage ratio can be simplified as

$$lr = \frac{q}{n}.$$

We estimate there are N Multicast groups in the data center network in the steady state, and each group i ($0 \leq i < N$) has a probability of p_i to join this interface. For a group not joining the interface, it also has a probability of f_i to be false-positively matched by the Bloom Filter on the interface. Hence, we can further express the traffic leakage ratio on the interface as

$$lr = \frac{\sum_{i=0}^{N-1}(1-p_i)f_i}{\sum_{i=0}^{N-1} p_i}.$$

Note that here we make an assumption that the packets from all the Multicast groups in the data center have chances to be forwarded to the switch by upstream switches. In other words, the traffic leakage ratio we define is the upper bound for the switch interface. Though there is some gap here, we find that this simple assumption works well in practice (please refer to Section IV). If we use the MBF based Multicast forwarding and set k_i hash functions for the group i, the traffic leakage ratio at this interface can be extended as Eq. 1.

$$lr = \frac{\sum_{i=0}^{N-1}(1-p_i)\{[1-(1-\frac{1}{m})^{\sum_{i=0}^{N-1} p_i * k_i}]^{k_i}\}}{\sum_{i=0}^{N-1} p_i}. \quad (1)$$

Given p_i for each group, we can optimally select k_i for every group, so as to minimize the traffic leakage ratio. It is a non-linear integer programming problem, and we will present our approximate algorithm to solve it in the next subsection.

Here we give an example to demonstrate the advantage of MBF over the standard Bloom Filter in the Multicast forwarding problem. Suppose that a switch interface has a Bloom Filter with the size of 50 bits. 10 groups have a presence probability of 0.2 on the interface, and 10 other groups have a presence probability of 0.9. Hence, the expected number of groups on the interface is $E(n) = 10 * 0.2 + 10 * 0.9 = 11$. If the standard Bloom Filter is used, the optimal number of hash functions for every group is $k = \ln 2 \frac{m}{n} = 3.15$. With $k = 3$, the expected traffic leakage ratio is $lr = 9.43\%$; while with $k = 4$, the expected traffic leakage ratio is $lr = 9.84\%$. However, by using different number of hash functions for different groups and setting $k_i = 7(1 \leq i \leq 10), k_i = 2(11 \leq i \leq 20)$, we can get an expected traffic leakage ratio of 2.46%, which is much lower than the minimum value under the standard Bloom Filter. This simple example shows that MBF has a great potential in limiting the traffic leakage in data center Multicast forwarding.

C. Calculating the Number of Hash Functions

As shown in Eq. 1, in order to minimize the traffic leakage ratio of the Bloom Filter forwarding engine on a switch interface, we can optimally calculate k_i for every group. However, it is a non-linear integer programming problem. When there are thousands of or even more Multicast groups in the data center, it is difficult to solve the problem in reasonable time.

We design a simple yet effective algorithm to address this problem. We sort the Multicast groups based on their probabilities to join the switch interface, and then divide the whole set into S disjoint slots. Each slot has the same number of groups. Groups within the same slot j share the same number of hash functions, K_j. In this way, we largely reduce the number of variables to solve. We can also vary the number of S by considering the tradeoff between the resultant traffic leakage ratio and the computation complexity.

[1]We leave as our future work to differentiate group traffic volumes. In this paper, we are concerned with the situation where tens of thousands of Multicast group communications are distributing files of similar size. This is also a typical scenario, e.g., when GFS and Map-Reduce jobs distribute the file chunks sized of 100MB.

We still assume there are in total N Multicast groups in the data center in the steady state. Each of the S slots has G groups. We define the presence probability of a slot j to join a switch interface, P_j, as the average presence probability of all groups within this slot to join this interface. It is set as $P_j = \frac{1}{G}\sum_{i=0}^{G-1} p_i$, where p_i is the presence probability of group i on the interface.

By slotting, the traffic leakage ratio on this interface is expressed as follows.

$$lr = \frac{\sum_{j=0}^{S-1} G*(1-P_j)\{[1-(1-\frac{1}{m})^{\sum_{j=0}^{S-1} P_j*K_j*G}]^{K_j}\}}{\sum_{j=0}^{S-1} G*P_j} = \frac{\sum_{j=0}^{S-1}(1-P_j)\{[1-(1-\frac{1}{m})^{\sum_{j=0}^{S-1} P_j*K_j*G}]^{K_j}\}}{\sum_{j=0}^{S-1} P_j}.$$

We can easily find out that if we use only one slot to put the Multicast groups, MBF degrades to the standard Bloom Filter, which sets the same number of hash functions for all the elements inside. In most cases, it works well enough to set S as a small number. As a result, we can use the simple enumeration method to select the optimal K_j, by limiting K_j as a positive integer less than a relatively small number.

However, if we use the naive enumerating algorithm, the computation complexity is X^S, where S is the number of group slots, and X is the maximum number to enumerate. The exponential computation complexity is usually unacceptable, even for small X and S. Fortunately, we can greatly reduce the computation complexity of the enumeration method based on Theorem 1.

Theorem 1: For two slots i and j, if the presence possibility of slot i is less than that of slot j, i.e., $P_i < P_j$, the optimal number of hash functions of slot i should be greater than or equal to that of slot j, i.e., $K_i \geq K_j$, so as to reduce the traffic leakage ratio of the MBF forwarding engine.

Proof: Please refer to Appendix A. ■

Based on Theorem 1, if we sort all the slots based on their presence probabilities and get $P_0 \geq P_1 \geq ... \geq P_{S-1}$, the number of hash functions we enumerate for these slots should satisfy $K_0 \leq K_1 \leq ... \leq K_{S-1}$. We call this enumeration algorithm *sorted enumeration*.

Theorem 2: The computation complexity of the sorted enumeration is $\binom{X+S-1}{X}$, where S is the number of groups slots and X is the maximum number of hash functions we enumerate for each group.

Proof: In sorted enumeration, there is always $K_0 \leq K_1 \leq ... \leq K_{S-1}$, $1 \leq K_i \leq X$. We build a mapping from K_i to O_i, where $O_i = K_i + i$. Then we get $O_0 < O_1 < ... < O_{S-1}$, $1 \leq O_i \leq (X+S-1)$. Enumerating K_i $(0 \leq i < S)$ from [1,X] is equivalent to enumerating O_i $(0 \leq i < S)$ from [1,$X+S-1$]. Obviously, the computation complexity of the latter is $\binom{X+S-1}{X}$. Therefore, the computation complexity of the sorted enumeration is also $\binom{X+S-1}{X}$. ■

Hence, by sorted enumeration, we reduce the computation complexity of the enumeration algorithm from exponential to polynomial. We will further demonstrate the benefit of the computational time in Section IV.

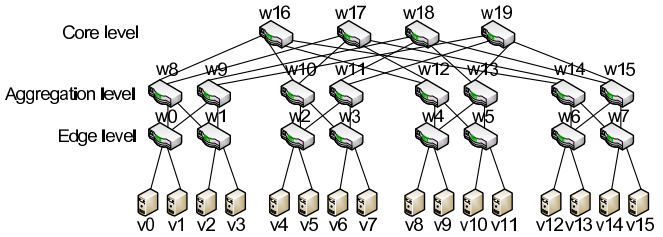

Fig. 2. A Fat-Tree architecture with 16 servers. It has three levels of 4-port switches.

Theorem 3: For the MBF Multicast forwarding engine on a certain switch interface, if we solve the optimal K_i for each slot i $(0 \leq i < S)$ by enumerating K_i from 1 to X, the traffic leakage ratio on this interface satisfies

$$lr \leq max\{lr_{opt1}, \frac{N-n}{n} * (\frac{1}{2})^X\}.$$

Here lr_{opt1} indicates the minimum traffic leakage ratio of the standard Bloom Filter (or 1-slot MBF), N denotes the total number of Multicast groups in the network, and n denotes the expected number of groups on the switch interface.

Proof: Please refer to Appendix B. ■

Mapping Group Addresses to the Group Sizes: We set different number of hash functions for the Multicast groups in the steady state based on their presence probabilities. When a switch forwards a Multicast packet, it needs to decide how many hash functions to use for Bloom Filter checking. Note that we use group size to determine the presence probability. Hence, we divide the Multicast address space into many slots, and intentionally map the Multicast group addresses to group sizes when assigning the group addresses. For example, we set 5 hash functions for a group size of 5000 in the steady state, and map the Multicast address 225.1.1.1 to the group size of 5000. Then, a switch can decide the number of hash functions to use based solely on the Multicast address. It is feasible in the data center Multicast since it is a managed environment. We can depend on a controller for the Multicast address assignment.

IV. Simulation

We conduct simulations to evaluate the MBF based Multicast forwarding scheme in typical data center networks. We compare MBF with the standard Bloom Filter, study its performance when we cannot accurately estimate the number of Multicast groups in the data center, and measure the computation complexity of our enumerating algorithm when solving the optimal number of hash functions.

A. Simulation Setup

We use Fat-Tree as the data center topology to run simulations. Fat-Tree [4] is a recently proposed data center network architecture to replace the traditional tree architecture. As shown in Fig. 2, the Fat-Tree network has three levels of switches, namely, edge-level, aggregation-level and core-level. The links between any two adjacent levels have the

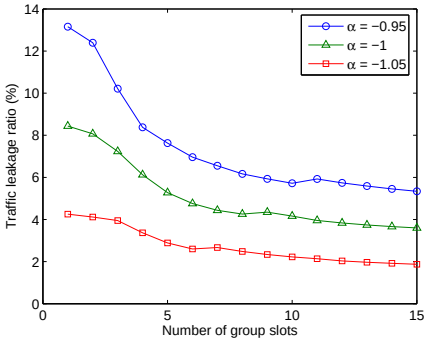

Fig. 3. Traffic leakage ratio against different numbers of slots and group size distributions.

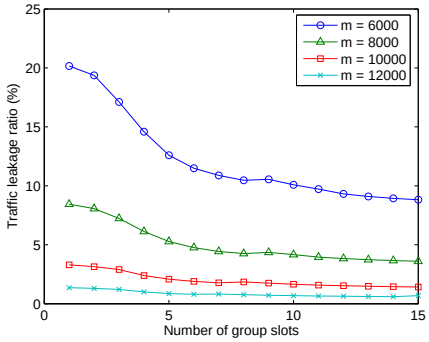

Fig. 4. Traffic leakage ratio against different numbers of slots and Bloom Filter sizes.

same network capacity, hence Fat-Tree network can offer 1:1 oversubscription ratio.

We generate Multicast groups of different sizes according to the simulation requirements. For each group, we select the source node and receiver set randomly from all the data center servers. We run every simulation for 10 times to mitigate the impact of randomness. The presence probability of a group on a certain switch interface can easily be inferred by the group size.

The edge-level switches in the Fat-Tree network are the bottleneck switches to hold the Multicast forwarding entries because they are the dominate parts in the Multicast trees. Hence, we only evaluate the traffic leakage ratio on the *edge-level switches*. In all our simulations, we assume there are in total 10,000 Multicast groups in the steady state.

B. The Number of Group Slots

We measure the traffic leakage ratio on the edge-level switches by varying the number of group slots we divide from 1 to 15. Note that MBF degrades to the standard Bloom Filter when we use only 1 slot. We also evaluate the impacts of other conditions, namely, the group size distribution pattern, the Bloom Filter size, as well as the network size.

Group Size Distribution: We run the simulation in a Fat-Tree network built by 48-port switches. The total number of servers is 27648. Assume the switch has a memory space of 100KB for fast-path forwarding, half of which is used for Multicast forwarding (the other half for Unicast). Hence, the Bloom Filter size for Multicast forwarding on each interface is about $50K * 8/48 \approx 8000$ bits. Note that in data centers, small-sized groups, such as file-chunk replication, should be much more than large-sized groups, such as Map-Reduce binary distribution [15]. Hence, we assume the group size follows a power-law distribution pattern. More formally, there is $y = a * x^{\alpha}$, where x is the group size between [3,27648] and y is the number of groups. We consider three cases, i.e., $\alpha = -1.05$, $\alpha = -1$ and $\alpha = -0.95$.

Fig. 3 shows the simulation results. We find that for all the distribution patterns, the traffic leakage ratio generally decreases with more group slots. But there is no obvious improvement when the number of group slots becomes larger than 10. There are cases that the traffic leakage ratio slightly increases when there are more group slots, because the different combinations of groups in each slot also affect the result. But it rarely occurs. It is noticeable that when the power-law distribution is less skewed, the benefit of MBF is more obvious. For instance, when $\alpha = -0.95$, MBF can reduce the traffic leakage ratio by more than 60% compared with the standard Bloom Filter, i.e., when using only 1 slot.

Bloom Filter Size: We study the impact of Bloom Filter size by choosing the Fat-Tree network with 48-port switches, and assuming the group size follows a power-law distribution pattern with $\alpha = -1$ between [3,27648]. We vary the Bloom Filter size on the edge-level switch interface as 6000, 8000, 10000 and 12000 bits. In practice it corresponds to switches with different fast memory spaces for the Multicast forwarding entry maintenance.

The simulation results are shown in Fig. 4. As expected, a longer Bloom Filter results in lower traffic leakage ratio because of the less false-positive matching. But we also find that when the Bloom Filter size is smaller, the traffic leakage reduction by MBF is more obvious. For example, when the Bloom Filter size is 6000 bits, the traffic leakage ratio in the standard Bloom Filter is above 20%, while that in MBF is as low as about 8%. It suggests that MBF is especially helpful when the fast memory space on switches is very limited.

Network Size: To evaluate the impact of network size, we choose Fat-Tree network composed of 16-port switches, 32-port switches, and 48-port switches, respectively. The Bloom Filter size on the edge-level interface is fixed as 8000 bits, and the group sizes also follow the power-law distribution with $\alpha = -1$ between [3,Z], where Z is the total number of servers in the Fat-Tree network.

Fig. 5 demonstrates the results. From the figure, the larger networks tend to have higher traffic leakage ratios. This is because when a switch has more links, there is more chance that a packet is false-positively forwarded to other interfaces. On the other hand, larger networks also benefit more from MBF based forwarding.

C. Inaccurate Estimation on the Number of Groups

We configure the MBF based Multicast forwarding engine on data center switches for steady states, and do not frequently update the forwarding engine. But there is high probability

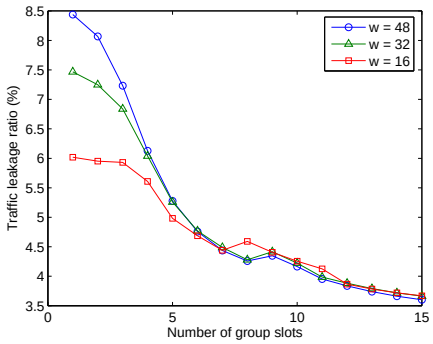

Fig. 5. Traffic leakage ratio against different numbers of slots and network sizes.

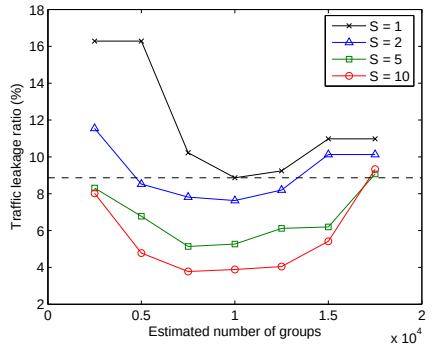

Fig. 6. Traffic leakage ratio under inaccurate estimation of the number of steady-state groups.

that the number of groups we estimate for some steady state is inaccurate. If this is the case, the number of hash functions we set for each group may not be optimal to control the traffic leakage ratio.

In this simulation, we evaluate the impact of inaccurate estimation on the traffic leakage ratio. The network we use is a Fat-Tree network composed of 48-port switches. The Bloom Filter size is 8000 bits. The actual number of Multicast groups is still 10000, but we vary the estimated number from 2500 to 17500. The group sizes follow the power-law distribution with $\alpha = -1.0$ between [3,27648]. We test the cases of dividing the group set into different number of group slots, i.e., $S = 1$, $S = 2$, $S = 5$ and $S = 10$. $S = 1$ also serves as the baseline of the standard Bloom Filter.

Fig. 6 shows the simulation results. We observe that generally, the more group slots we divide, the lower traffic leakage ratio we get. It is consistent with the previous simulations, but we validate that it also holds even when the estimation is inaccurate. For the same number of group slots, the traffic leakage ratio first decreases, then reaches the minimum point, and then gradually increases with the even larger estimated number. But it is interesting to find that the minimum traffic leakage ratio is not obtained when the estimated number of groups is exactly equal to the actual number. Instead, the estimated number at the magic point is a little less than the actual number. It is because when we design MBF, we assume that all the N Multicast groups in the data center have chances to be forwarded to the edge-level switch where this interface lies. However, in practice the number should be smaller than N. The traffic leakage ratio is minimized when the estimated number is equal to the number of groups forwarded to the switch.

Moreover, we observe that even when the estimation is inaccurate, the traffic leakage ratio using MBF still outperforms the optimal value of the standard Bloom Filter if the estimation falls into a reasonable range. In our simulation, when the estimated number of groups is between [5000,15000], the traffic leakage ratios in $S = 5$ and $S = 10$ curves are less than the minimum value with the standard Bloom Filter, which is shown by the dashed line in the figure. This property of MBF indicates that we can relax the precision requirement on the estimated number of steady-state groups. Rough estimation usually works well enough in practice.

TABLE I
COMPUTATION TIME OF THE ENUMERATION ALGORITHMS. THE MAXIMUM NUMBER OF HASH FUNCTIONS TO ENUMERATE (X) IS 10.

Number of slots (S)	Sorted Enumeration	Naive enumeration
1	≈ 0	≈ 0
2	≈ 0	≈ 0
3	≈ 0	≈ 0
4	10ms	50ms
5	20ms	430ms
6	40ms	4920ms
7	80ms	58240ms
8	190ms	691955ms

D. Computation Time of the Enumerating Algorithm

We use the enumeration method to calculate the optimal number of hash functions for each group slot. As shown in Section III, we can reduce the computation complexity from X^S to $\binom{S+X-1}{X}$ by choosing sorted enumeration other than naive enumeration, where S is the number of group slots, and X is the maximum number of hash functions we enumerate for each group. In this simulation, we compare the actual computation time between the two enumeration algorithms.

The topology is still a Fat-Tree network composed of 48-port switches. 10000 groups are generated and the group sizes follow the power-law distribution with $\alpha = -1$ between [3,27648]. We divide all the groups into S ($1 \leq S \leq 8$) slots, and the maximum number of hash functions to enumerate is 10. Both algorithms are run on a Desktop with AMD Athlon(tm) II X2 245 2.91G CPU and 2GB DRAM.

Table I shows the results. We find that our sorted enumeration can greatly reduce the computation time. When we divide the groups into 8 slots, the computation time of the sorted enumeration is three orders of magnitude lower than the naive enumeration. The difference will be even magnified when we divide the Multicast groups into more slots, or the number of hash functions to enumerate is higher.

V. IMPLEMENTATION AND EXPERIMENT

In this section, we present the implementation of the prototyped MBF based Multicast forwarding engine and the experiments atop.

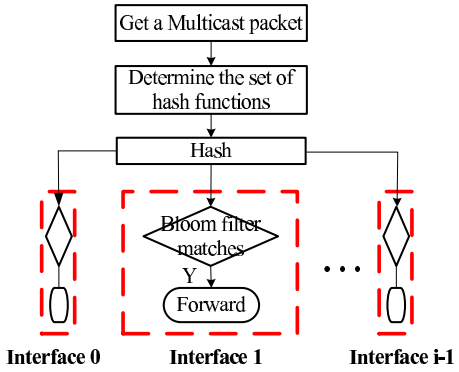

Fig. 7. The flowchart on the MBF forwarding engine.

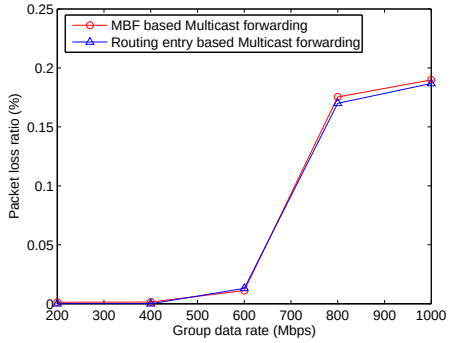

Fig. 8. Packet loss ratio under the MBF based Multicast forwarding and the routing entry based Multicast forwarding.

A. Implementation

MBF based Multicast forwarding engine requires modifying the data plane of switches. It has been shown that, if employing OpenFlow [17] framework which has already been ported to run on a variety of hardware platforms, such as switches from Cisco, Hewlett Packard, and NEC, only minor modifications (several lines of codes) on the data path of switches are required to encompass Bloom Filter based forwarding [14]. This supports our belief that the MBF forwarding scheme can be well incorporated into existing commodity switches.

At current stage, we have implemented a software based MBF forwarding engine on Linux platform leveraging the NetFilter module. Fig. 7 shows the flowchart for the forwarding engine. When a Multicast arrives, the forwarding engine checks the Multicast group address, and then determines the set of hash functions to use for this group. Note that we make the mapping from group address to group size, and further to the set of hash functions, as discussed in Section III. The Bloom Filter on each interface will be checked based on the hashed bits to decide whether to forward the Multicast packet to that interface.

B. Experiments

We conduct experiments to study the system overhead brought by MBF based forwarding engine. We use a small test bed composed of 5 hosts: 1 Multicast sender, 3 Multicast receivers and 1 forwarder. The forwarder has one 2.8G Intel(R) Core(TM)2 Duo CPU, 2GB DRAM, and four Realtek RTL8111/8168B PCI Express Gigabit Ethernet NICs, each connecting one of the other four hosts. The forwarder is installed with Ubuntu Linux 9.10 (karmic) with kernel 2.6.31-14-generic. The sender sends out UDP Multicast packets for a group at different speeds (from 200Mbps to 1000Mbps), and the 3 receivers join the group to receive the packets.

We first measure the CPU overhead of the MBF forwarding engine. At all network speeds we test (from 200Mbps to 1Gbps), we do not find obvious rise on the CPU utilization. Then we compare the packet loss ratio between MBF forwarding engine and the traditional routing entry based Multicast forwarding engine. Fig. 8 shows the results. It suggests that at all packet speeds, the packet loss ratios under the two forwarding engines are similar. Even when we send the Multicast packet at the full network speed of 1Gbps, the packet loss ratio is only 0.2%. Overall, the MBF based Multicast forwarding brings little system overhead.

VI. Related Work

In this section we discuss the related work in scalable Multicast forwarding as well as the Bloom Filter improvements.

A. Scalable Multicast Forwarding

As one of the deploying obstacles of Multicast in the Internet, scalable Multicast forwarding has attracted much attention in the research community. For data center networks where low-end switches with limited forwarding table memory space are used, the problem is even more challenging. One possible solution is to aggregate multiple Multicast forwarding entries into a single one, as used in Unicast. However, it is difficult for Multicast aggregation because Multicast group address is logical without any topological information [18].

Bloom Filter can be used to compress in-switch Multicast forwarding entries. In FRM [12], Bloom Filter based group information is maintained at border routers to help determine inter-domain Multicast packet forwarding. A similar idea is adopted in BUFFALO [16], though it is primarily designed for the scalable Unicast forwarding. They both use standard Bloom Filter, which requires the exact membership information.

Another solution is to encode the tree information into in-packet Bloom Filter, and thus there is no need to install any Multicast forwarding entries in network equipments. For instance, LIPSIN [13] adopts this scheme. However, the bandwidth overhead of this solution comes not only from the false-positive forwarding of Bloom Filters, but also from the in-packet Bloom Filter field. Besides, in-packet Bloom Filter based Multicast forwarding also relies on the actual Multicast tree information.

In the recent work of MCMD [19], scalable data center Multicast is realized in the way that only partial groups are supported by Multicast according to the hardware capacity, and the other groups are translated into Unicast communications. Though this approach solves the scalability problem of data center Multicast, it cannot fully utilize the advantage of Multicast in bandwidth saving.

B. Bloom Filter Improvements

To accommodate different application scenarios, there are many improvements on the standard Bloom Filter.

Efforts have been made on efficiently allocating the memory for a dynamic Bloom Filter, the size of which can vary. In the case of standard Bloom Filter, the memory space required for a Bloom Filter should increase linearly with the maximum possible number of elements in the set to control the maximum false positive rate, and thus much memory is wasted. Dynamic Bloom Filter [20], or DBF, uses a variable number of standard Bloom Filters to represent a Bloom Filter set. i-DBF [21] improves DBF in the addition operation with a new algorithm. Incremental Bloom Filter [22] considers the problem of minimizing the memory requirement in case where the number of elements in the set is unknown in advance but the distribution of the number of elements is known.

Counting Bloom Filter [23], or CBF, uses a counter instead of a single bit in each Bloom Filter cell in order to allow element deletion operation. When inserting an element into the Bloom Filter, we increment the hashed counters by 1; while deleting an element from the Bloom Filter, we decrement the hashed counters by 1. There are also some enhancements based on the basic idea of CBF. Spectral Bloom Filter [24] makes the length of the counter of each cell different in order to optimize the counter space allocation. Dynamic Count Filters [25], or DCF, makes further improvements by using two vectors, namely, a CBF vector and an OPV vector, to represent a Bloom Filter set. dlCBF [26] provides a simple hashing-based alternative based on d-left hashing, which offers the same functionality as a CBF, but uses much less space.

Compressed Bloom Filter [27] targets to compress the Bloom Filter bits to save memory space. Instead of optimizing the number of hash functions for a given Bloom Filter size, compressed Bloom Filter optimizes the number of hash functions for the compressed size of the Bloom Filter. The cost is the processing time for compression and decompression, which can use simple arithmetic coding. Multi-Layer compressed CBF was also proposed to reduce the memory requirements and the lookup complexity of CBF [28].

Space-code Bloom Filter [29] is an approximate representation of a multiset Bloom Filter, which allows for the query "How many occurrences of an element are there in a set". Distance-Sensitive Bloom Filters [30] generalizes Bloom Filters to answer queries of the form, "Is x close to an element of S?", where closeness is measured under a suitable metric. Such a data structure would have several natural applications in networking and database applications.

VII. Conclusion

In this paper, we designed MBF, which extends the standard Bloom Filter by embracing element uncertainty. MBF sets the number of hash functions in a per-element level to minimize the expected false positive of the Bloom Filter. We applied MBF into the Multicast forwarding engine on switches to achieve scalable Multicast, and developed a low-complexity algorithm to calculate the optimal number of hash functions for each Multicast group. Simulation results showed that MBF can significantly reduce the traffic leakage ratio compared with the standard Bloom Filter, especially when the group sizes are various, the Bloom Filter size is narrow, and the network size is large. We prototyped a software based MBF forwarding engine on the Linux platform. The experiments on the testbed demonstrated that MBF brings little system overhead.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", In *Proceedings of OSDI'04*, 2004

[2] C. Guo, H. Wu, K. Tan and etc., "DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers", In *Proceedings of ACM SIGCOMM'08*, Aug 2008

[3] C. Guo, G. Lu, D. Li and etc., "BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers", In *Proceedings of ACM SIGCOMM'09*, Aug 2009

[4] M. Al-Fares, A. Loukissas and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture", In *Proceedings of ACM SIGCOMM'08*, Aug 2008

[5] A.Greenberg, J. Hamilton, N. Jain and etc., "VL2: A Scalable and Flexible Data Center Network", In *Proceedings of ACM SIGCOMM'09*, Aug 2009

[6] D. Li, C. Guo, H. Wu and etc., "Scalable and Cost-effective Interconnection of Data Center Servers using Dual Server Ports", *IEEE/ACM Transactions on Networking*, 19(1):102-114, 2011.

[7] L. Barroso, J. Dean and U. Holzle, "Web Search For a Planet: The Google Cluster Architecture", *IEEE Micro*, 23(2):22-28, 2003

[8] Hadoop, http://hadoop.apache.org/

[9] S. Ghemawat, H. Gobioff and S. Leung, "The Google File System", In *Proceedings of SOSP'03*, Oct 2003

[10] M. Isard, M. Budiu, Y. Yu and etc., "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks", In *Proceedings of EuroSys'07*, Mar 2007

[11] D. Newman, "10 Gig access switches: Not just packetpushers anymore", *Network World*, 25(12), Mar 2008

[12] S. Ratnasamy, A. Ermolinskiy and S. Shenker, "Revisiting IP Multicast", In *Proceedings of ACM SIGCOMM'06*, Aug 2006

[13] P. Jokela, A. Zahemszky, C. Rothenberg, and etc., "LIPSIN: Line Speed Publish/Subscribe Inter-Networking", In *Proceedings of ACM SIGCOMM'09*, Aug 2009

[14] C. Rothenberg, C. Macapuna, F. Verdi, and etc., "Data center networking with in-packet Bloom filters", In *Proceedings of SBRC'10*, May 2009

[15] D. Li, J. Yu, J. Yu and etc., "Exploring Efficient and Scalable Multicast Routing in Future Data Center Networks", In *Proceedings of IEEE INFOCOM'11*, Apr 2011

[16] M. Yu, A. Fabrikant and J. Rexford, "BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations", In *Proceedings of ACM CoNext'09*, Dec 2009

[17] OpenFlow, http://www.openflowswitch.org/

[18] D. Thaler and M. Handley, "On the Aggregatability of Multicast Forwarding State", In *Proceedings of IEEE INFOCOM'00*, Mar 2000

[19] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, and etc., "Dr. Multicast: Rx for Data Center Communication Scalability", In *Proceedings of ACM Eurosys'10*, Apr 2010

[20] D. Guo, J. Wu, H. Chen, and etc., "Theory and Network Applications of Dynamic Bloom Filters", In *Proceedings of IEEE INFOCOM'06*, Apr 2006

[21] J. Wang, M. Xiao, J. Jiang and etc., "i-DBF: an Improved Filter Representation Method on Dynamic Set", In *Proceedings of GCCW'06*, Oct 2006

[22] F. Hao, M. Kodialam, and T. Lakshman, "Incremental Bloom Filters", In *Proceedings of IEEE INFOCOM'08*, Apr 2008

[23] L. Fan, P. Cao, J. Almeida, and etc, "Summary cache: a scalable wide-area web cache sharing protocol", *IEEE/ACM Transactions on Networking*, 8(3):281-293, 2000

[24] S. Cohen and Y. Matias, "Spectral bloom filters", In *Proceedings of SIGMOD'03*, Jun 2003

[25] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and etc., "Dynamic count filters", *ACM SIGMOD Record*, 35(1):26-32, 2006.

[26] F. Bonomi, M. Mitzenmacher, R. Panigrahy, and etc., "An improved construction for counting bloom filters", In *Proceedings of ESA'06*, 2006.

[27] M. Mitzenmacher, "Compressed Bloom Filters", *IEEE/ACM Transactions on Networking*, 10(5):604C612, 2002

[28] D. Ficara, S. Giordano, G. Procissi, and etc., "MultiLayer Compressed Counting Bloom Filters", In *Proceedings of IEEE INFOCOM'08*, Apr 2008
[29] A. Kumar, J. Xu, J. Wang and etc., "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement", In *Proceedings of IEEE INFOCOM'04*, Apr 2004
[30] A. Kirsch and M. Mitzenmacher, "Distance-sensitive Bloom Filters", In *Proceedings of ALENEX'06*, Jal 2006

APPENDIX

A. Proof of Theorem 1.

Without loss of generality, we assume that $i = 0$ and $j = 1$. The number of Multicast groups within each slot is G. We assume that the traffic leakage ratio of the MBF forwarding engine, lr, gets the minimum value when the number of hash functions for slot j, K_j, is set as the optimal value of K'_j. We further assume $K'_0 = K_a$ and $K'_1 = K_b$.

Then we use a contradictory condition to prove Theorem 1. When there is $P_0 < P_1$ and $K_a < K_b$, we can get a smaller traffic leakage ratio by swapping K_a and K_b.

We use $f(K_0, ..., K_{S-1})$ to stand for the probability that a certain bit in the Bloom Filter is set by 1, when the number of hash functions for slot i is K_i. There should be

$$f(K_0, ..., K_{S-1}) = 1 - (1 - \frac{1}{m})^{G*\sum_{i=0}^{S-1} P_i * K_i}.$$

Let $l_{x,y}$ denote the leaked traffic of the MBF forwarding engine when slot 0 uses x hash functions and slot 1 uses y hash functions, under the condition that all other slots use the optimal number of hash functions. Then we compare l_{K_a,K_b} and l_{K_b,K_a} when there is $P_0 < P_1$.

We further denote $f_{K_a,K_b} = f(K_a, K_b, K'_2, ..., K'_{S-1})$ and $f_{K_b,K_a} = f(K_b, K_a, K'_2, ..., K'_{S-1})$. Since $P_0 < P_1$ and $K_a < K_b$, we have $P_0 * K_a + P_1 * K_b > P_0 * K_b + P_1 * K_a$. So there is $f_{K_a,K_b} > f_{K_b,K_a}$.

According to the properties of the MBF, we have

$$\begin{aligned} l_{K_a,K_b} &= G * [\sum_{i=2}^{S-1} (1 - P_i) * {f_{K_a,K_b}}^{K'_i} \\ &+ (1 - P_0) * {f_{K_a,K_b}}^{K_a} + (1 - P_1) * {f_{K_a,K_b}}^{K_b}] \\ &> G * [\sum_{i=2}^{S-1} (1 - P_i) * {f_{K_b,K_a}}^{K'_i} \\ &+ (1 - P_0) * {f_{K_b,K_a}}^{K_a} + (1 - P_1) * {f_{K_b,K_a}}^{K_b}] \\ &> G * [\sum_{i=2}^{S-1} (1 - P_i) * {f_{K_b,K_a}}^{K'_i} \\ &+ (1 - P_0) * {f_{K_b,K_a}}^{K_b} + (1 - P_1) * {f_{K_b,K_a}}^{K_a}] \\ &= l_{K_b,K_a}. \end{aligned}$$

Therefore, when there is $P_0 < P_1$ and $K_a < K_b$, we can get smaller leaked traffic by swapping K_a and K_b, which also results in a smaller traffic leakage ratio. This is a contradict.

In summary, when there is $P_i < P_j$, we should set $K'_i \geq K'_j$ to minimize the traffic leakage ratio.

B. Proof of Theorem 3.

We use $lr(K_0, K_1, ..., K_{S-1})$ to denote the traffic leakage ratio when slot 0 uses K_0 hash functions, slot 1 uses K_1 hash functions, ..., and slot $S-1$ uses K_{S-1} hash functions. Assume in the standard Bloom Filter, K is the number of hash functions assigned to each group, and we get the minimum traffic leakage ratio, lr_{opt1}, when there is $K = K_{opt1}$. Hence we also have $lr_{opt1} = lr(K_{opt1}, K_{opt1}, ..., K_{opt1})$ for any number of S.

There are two cases when we use enumeration to solve the optimal K_i for each slot i.

Case 1: $K_{opt1} \leq X$.

Since we have enumerated each K_i from 1 to X, so there is

$$lr \leq lr(K_0, K_1, ..., K_{S-1}), 1 \leq K_i \leq X.$$

We assume $K_{opt1} \leq X$, so we have

$$lr \leq lr(K_{opt1}, K_{opt1}, ..., K_{opt1}) = lr_{opt1}.$$

Case 2: $K_{opt1} > X$.

Let P_i be the presence probability of slot i, and m be the Bloom Filter size. There is

$$\begin{aligned} lr &\leq lr(X, X, ..., X) \\ &= \frac{\sum_{i=0}^{S-1} (1 - P_i) * [1 - (1 - \frac{1}{m})^{G*\sum_{i=0}^{S-1} P_i * X}]^X}{\sum_{i=0}^{S-1} P_i} \\ &\leq \frac{\sum_{i=0}^{S-1} (1 - P_i) * [1 - (1 - \frac{1}{m})^{G*\sum_{i=0}^{S-1} P_i * K_{opt1}}]^X}{\sum_{i=0}^{S-1} P_i}. \end{aligned}$$

Note that in the standard Bloom Filter, there is $K_{opt1} = \ln 2 * \frac{m}{n}$. Here $n = G * \sum_{i=0}^{S-1} P_i$.

So we have

$$\begin{aligned} lr &\leq \frac{\sum_{i=0}^{S-1} (1 - P_i) * [1 - (1 - \frac{1}{m})^{\sum_{i=0}^{S-1} G*P_i*\ln 2*\frac{m}{n}}]^X}{\sum_{i=0}^{S-1} P_i} \\ &= \frac{\sum_{i=0}^{S-1} (1 - P_i) * [1 - (1 - \frac{1}{m})^{m*\ln 2}]^X}{\sum_{i=0}^{S-1} P_i} \\ &\approx \frac{\sum_{i=0}^{S-1} (1 - P_i) * [1 - (1 - e^{\frac{-m \ln 2}{m}})]^X}{\sum_{i=0}^{S-1} P_i} \\ &= \frac{\sum_{i=0}^{S-1} (1 - P_i) * [1 - (1 - \frac{1}{2})]^X}{\sum_{i=0}^{S-1} P_i} \\ &= \frac{\sum_{i=0}^{S-1} G * (1 - P_i)}{\sum_{i=0}^{S-1} G * P_i} * \frac{1}{2}^X \\ &= \frac{N - n}{n} * (\frac{1}{2})^X. \end{aligned}$$

In summary, we get

$$lr \leq max\{lr_{opt1}, \frac{N - n}{n} * (\frac{1}{2})^X\}.$$