Forwarding Anomalies in Bloom Filter Based Multicast

Mikko Särelä, Christian Esteve Rothenberg, Tuomas Aura, Andras Zahemszky, Pekka Nikander, and Jörg Ott

***Abstract*—Several recently proposed multicast protocols use in-packet Bloom filters to encode multicast trees. These mechanisms are in principle highly scalable because no per-flow state is required in the routers and because routing decisions can be made efficiently by simply checking for the presence of outbound links in the filter. Yet, the viability of previous approaches is limited by the possibility of forwarding anomalies caused by false positives inherent in Bloom filters.**

This paper explores such anomalies, namely (1) packets storms, (2) forwarding loops and (3) flow duplication. We propose stateless solutions that increase the robustness and scalability of Bloom filter based multicast protocols. In particular, we show that the parameters of the filter need to be varied to guarantee the stability of the packet forwarding, and we present a bit permutation technique that effectively prevents both accidental and maliciously created anomalies. We evaluate our solutions in the context of BloomCast, a source-specific inter-domain multicast protocol, using analytical methods and simulations.

I. Introduction

Since its the origins in the early 90s, IP multicast has received much attention from researchers (e.g. [5], [18]) and standards bodies (e.g. [9], [2], [14], [10], [8], [3]). These multicast protocols require routers to store forwarding state for each multicast group to which they forward packets. This model is useful for a network with few (potentially large) multicast groups but it breaks down if the number of groups grows. Recent work by Karpilovsky et al. [13] shows that, for the studied large AS, many groups have a low number of receivers (≤ 3) and low bandwidth usage. This indicates that there is, indeed, need for a multicast architecture that can scale to large numbers of groups.

Bloom filters [4] carried in packet headers are a promising approach to making multicast scalable to a large number of multicast groups [19], [12], [7], [27]. The proposed protocols separate the multicast group management and multicast forwarding, thus removing the need for distributed multicast route computation and per group state in multicast routers. Instead of managing the group state in the routing system, the state is maintained either at the source or in a separate group management component. Similar to source routing, the multicast tree is placed in the packet header, where it is encoded as a Bloom filter.

This encoding creates false positives, i.e. some links may match the Bloom filter even when they had not been intentionally added to it. The false positives give rise to anomalies in packet forwarding. We analyze three such forwarding anomalies, namely (1) packet storms, (2) forwarding loops, and (3) duplicate flows. To our knowledge, of these, only forwarding loops are discussed in existing work [6], [12]. We also propose and evaluate effective solutions to these anomalies.

If the average number of false positives per router exceeds one (in some part of the network), the result is a packet storm. Networks with many highly connected nodes, such as the Internet, are especially vulnerable to such phenomena. We show that packet storms can be prevented by varying the number of hash functions as a function of the out-degree of the router. Each router chooses this parameter locally so that the average number of false positives stays below one. Our evaluation shows that in the Internet topology this also reduces the overhead caused by false positives by 50%.

Forwarding loops and flow duplication are caused by false positives that lead to a router in the original forwarding tree. Loops are especially problematic since, every time a packet goes through a loop, a copy is also sent to the legitimate downstream receivers. We propose a per-hop bit permutation scheme of Bloom filters as a stateless solution, and show by means of analysis and simulations that it is an efficient and effective method for preventing both forwarding loops and duplicate flows.

The rest of the paper is organized as follows. Section II gives an overview of Bloom filter based multicast. Packet storms, looping, and duplication anomalies are explained in Section III and the solutions to the anomalies are analyzed in Section IV. In Section V, we introduce the BloomCast architecture and use it for simulation based evaluation in Section VI. Finally, Section VII concludes the paper.

II. Background

A multicast tree can be encoded into a small Bloom filter carried in each packet. Each router locally names the links to its peers with a link mask ; a set of bit positions in the Bloom filter set to 1. A forwarding tree is created by bitwise ORing the link masks of the corresponding links in the multicast tree. Every router that receives a router forwards the packet to each peer When a packet is forwarded, the router checks for each of its peers whether the corresponding bits in the link mask for that peer are all set to 1. It forwards the packet to each peer that matches.

The probability of a false positive is $fpr = \rho^k$ where ρ is the fill factor of the filter (i.e., the percentage of bits set to 1) and k is the number of bits set for each data item added to the filter – typically after the applying k independent hash functions on the item. Previous discussions of the effects of false positives in such in-packet Bloom filter based forwarding applications have revolved around two issues: (1) the extra

bandwidth they consume, and (2) potential forwarding loops they may cause. The basic assumption in the literature is that the false positive rate can be kept low enough for the traffic overhead to be negligible. Ermolinskiy [6] found the sharp decline in bandwidth efficiency once false positive rate exceeded 0.2% to be caused by loops. LIPSIN [12] utilizes a cache in all routers to capture looping packets, requiring routers to maintain short-lived per packet state.

False positives are primarily controlled by adjusting the size and by limiting the capacity of the Bloom filter i.e. the number of data items stored in it. The latter is effectively a limit on the size of the multicast tree (or the length of the forwarding path for unicast). In order to optimize the Bloom filter for the largest trees, the limit can be set so that $\rho \leq 50\%$. In that case, the false positive rate will be $fpr = 2^{-k}$, and the maximum size of the path or multicast tree that can be encoded in the Bloom filter is approximately $n = (m/k) \cdot \ln 2$. If larger multicast groups are needed, they can either be divided into several separate multicast trees [19] or by encoding a path as a virtual link [12].

A. Related work on Bloom filter based forwarding

Free Riding Multicast (FRM) [19] separates group membership management and forwarding by pushing the multicast tree computation to the sender's AS. Each AS maintains a group set for each other AS in the network that describes the groups the AS wants to receive. The sets are piggybacked on BGP advertisements. For each AS-AS link, the link masks are computed from the AS numbers A:B.

LIPSIN [12] uses a Bloom filter based forwarding fabric for a network architecture designed to support publish/subscribe. Each router has a set of d Link ID Tags for each link. For a given multicast tree, a topology manager chooses the d value that minimizes the number of false positives. Z-formation [7] proposes dynamically computed link masks enabling routing Bloom filters to act as capabilities [1]. MPSS [27] uses Bloom filters for VPN multicast services in operator deployments.

Another variant was proposed in [22]. Each packet carries a Bloom filter that encodes the set of IP destinations. The proposed scheme ensures loop freeness, but is computationally expensive as each router needs to recompute the Bloom filter for each next hop.

Bloom filter enhancements: Variable-length Signatures [16] and Popularity Conscious Bloom Filters [28] use similar ideas to our varying k approach. The first allows setting partial signatures, where only some of the bits need to be set. An element is reported to be a member of the set if at least $q(\leq k)$ of the bits are set to 1. The latter varies the number of hash functions as a function of the data item popularity to reduce the average number of false positives. Recently, deletable Bloom filters [20] have been proposed as a memory efficient add-on to allow probabilistic element deletions, enabling for instance to remove already processed outgoing links as the in-packet Bloom filter traverses the network.

TABLE I
FALSE POSITIVE RATE, MAXIMUM MULTICAST TREE SIZE AND BANDWIDTH OVERHEAD AS FUNCTIONS OF k (BLOOM FILTER LENGTH $m = 800$, FILL FACTOR $\rho = 50\%$ AND NETWORK NODE DEGREE $d = 10$)

k	*fpr*	max tree size	bandwidth overhead
4	6.3 %	139	114.3 %
5	3.1 %	111	34.8 %
6	1.6 %	92	14.5 %
7	0.8 %	79	6.7 %
8	0.4 %	69	3.2 %
9	0.2 %	62	1.6 %
10	0.1 %	55	0.8 %

B. Traffic amplification and routing loops

While we are not aware of any work on Bloom filter based forwarding anomalies, similar problems have arisen elsewhere. Internet routing loops have been shown to cause traffic amplification [24], [25] causing congestion and enabling denial-of-service attacks even against the backbone network. Icarus [23] uses Bloom filters to prevent unicast loops and multicast implosions in IP forwarding.

III. FORWARDING ANOMALIES

As explained earlier, Bloom filters can cause false positives. In this Section, we analyze the effects of false positives in the forwarding plane and describe three anomalies: (1) packet storms, (2) loops, and (3) flow duplication.

A. Packet storms of false positives

In a network with unicast communication and constant node degree b (each forwarding node has b neighbors), the bandwidth overhead caused by the false positives is

$$\lim_{i \to \infty} (d-2) \cdot fpr \cdot \frac{1 - ((d-1) \cdot fpr)^i}{1 - (d-1) \cdot fpr} = \frac{(d-2) \cdot fpr}{1 - (d-1) \cdot fpr}$$

if $(d-1) \cdot fpr < 1$. If the product of the node degree and false-positive rate exceeds one, $(d-1) \cdot fpr = (d-1) \cdot \rho^k \geq 1$, false positives will cause an unlimited traffic explosion in the network. This is because each false positive will, on the average, cause more than one false positive in the next forwarding node. Theoretically, when we limit the filter fill factor to $\rho \leq 50\%$, the network will be stable only if $(d-1) \cdot 2^{-k} < 1$. (Note that, excluding the ingress link, there are $d-1$ potential outbound links at each forwarding node.)

Table I shows some of these values for an 800-bit Bloom filter. The bandwidth overheads caused by false positive packets have been calculated for the node degree $d = 10$. Based on such estimates, the sensible values of k are usually thought to be between 5 and 8.

Let $k = 5$ and consider high-degree core nodes in the network. For example, if a node has 1000 neighbors, a packet routed to it will cause, on the average, over 30 false positives to be sent out. Moreover, every false positive received by such a high-degree node will cause another over 30 copies of it to be sent. This kind of amplification would cause a packet storm in the highly connected core part of the network.

While practical network topologies have few very-high-degree nodes, inter-domain routing in the Internet has enough of them to be vulnerable to such instability. There are hundreds of ASes with 32 or more peering relations. It can be assumed that most of these are at the core of the network and that they are highly connected to each other. With $k = 5$, the first packet sent through this part of the network would cause an unlimited packet storm as each false positive would induce, on the average, at least one more false positive.

Moreover, the peering and transit relations between ASs are determined by business aspects and technology should not arbitrarily limit them. There is no guarantee that the node degree in inter-domain routing will not grow in the future. It would not be prudent to base the stability of an inter-domain routing protocol on restrictive assumptions about network characteristics that are difficult to measure and control.

B. Forwarding loops

In addition to the bandwidth overhead, the literature identifies forwarding loops as another potential consequence of the false positives. A loop can arise from a single false positive that causes a packet to be sent back to a node which it has already traversed, as shown in Figure 1. Even rare occurrences of such loops can severely disrupt a network if the packet gets stuck in the loop for a long or an unlimited time. In addition the packets congesting the loop, a copy of each packet will be sent to the intended downstream tree every time it goes around the loop.

A further consideration is that a malicious sender may intentionally construct packets that end in such infinite loops: it can simply add the looping link intentionally to the filter. This vulnerability is particularly alarming because one of the promises of Bloom filter based publish-subscribe architectures has been protection against packet-flooding attacks.

FRM [6] finds that false positive rate above 0.2% causes a sharp decline in bandwidth efficiency due to forwarding loops. LIPSIN [12] proposes several solutions: cached state in the routers, a TTL field in the packets, and valley-free network topology. These ideas have major limitations. The cached state would make the protocol less scalable with the number of flows and, thus, also create another denial-of-service vulnerability. The TTL field would end the loop after a finite number of rounds. That might suffice for loops that occur accidentally with low probability. In DoS attacks, however, even a small number of rounds in the loop can create significant traffic amplification to the downstream tree. Valley-free [11] networks prevent loops in theory[1] but are vulnerable to another routing anomaly that will be presented next.

C. Flow duplication

Loops are not the only anomaly that can arise because of the false positives in probabilistic packet forwarding. Figure 2(a) shows how a false positive can cause a packet to be forwarded

[1]In practice even valley free networks experience loops e.g. due to misconfiguration and routing dynamics.

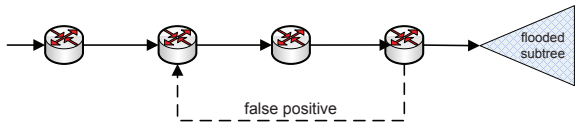

Fig. 1. Forwarding loop

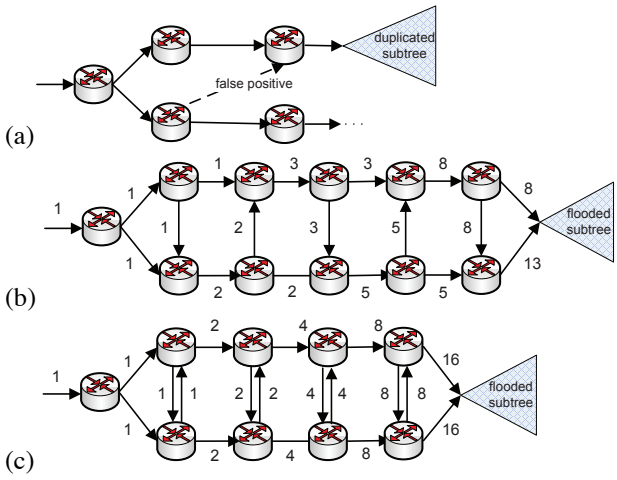

Fig. 2. Flow duplication

for a second time to a subtree even though the forwarding topology is loop-free.

Figures 2(b)-(c) presents rather artificial constructions in which the number of packets grows according to the Fibonacci sequence and as powers of two. The numbers indicate the number of copies of each packet that traverse the link. These examples are interesting for two reasons. First, they shows that exponential growth is possible even when packets have low TTL values and when the valley-free forwarding rule is observed.[2] Second, more importantly, such extremely anomalous cases could be constructed by malicious senders. While accidental flow duplication is clearly not as serious a problem as accidental forwarding loops, the duplication becomes an important consideration when we consider secure ways of preventing intentional attacks.

IV. Increasing robustness and scalability

In this section, we propose and analyze two methods that can be used to prevent the three forwarding anomalies identified in the previous section.

A. Varying Bloom filter parameters

False positives occur randomly, hence, packet storms cannot be detected and stopped reactively. To control them, a stability condition must be preserved, i.e. $d \cdot \rho^k < \alpha$, where $\alpha \leq 1$ is preferred maximum average number of false positives per node, d is the node degree of the forwarding node and k is the number of bits set in the link masks.

[2]The exponent is limited by the number of tiers in the routing architecture.

Given a fill factor, the average number of false positives is independent of the length of the Bloom filter. This suggests a two-fold solution: firstly, each node sets k locally based on the node degree and, secondly, the source sets the length m of the Bloom filter so that the fill factor $\rho \approx \rho_{max}$. (Either technique can also be applied separately.)

1) Varying k: Each node sets k locally based on its degree:

$$k = \lceil \log_2(d-1) \rceil + r \text{ for some small global integer } r.$$

This local condition guarantees that, at any node, the false positive rate is $fpr < \rho_{max}^r/(d-1)$. Thus, a false positive arriving at any node will cause, on the average, fewer than 2^{-r} further false positives to be sent. If we set $\rho_{max} = 50\%$ and r globally to some small value, this limits the bandwidth overhead caused by the false positives to $1/(2^r - 1)$. For example, for $r = 3$, the overhead is limited to 14%.

The variable k does not only prevent traffic storms at the high-degree core parts of the network; it also optimizes the usage of Bloom filter capacity so that there is no unnecessary safety margin and the filter capacity can be used to encode larger multicast trees. This is particularly important if the multicast group size exceeds the maximum capacity of a single filter and the group needs to be split into several trees. The simulations in Section VI confirm this by showing about 50% reduction of *fpr* with variable k compared to static k (holding the Bloom filter size and multicast tree constant).

2) Varying the length of Bloom filter: The number of elements added into the Bloom filter (together with the Bloom filter length m and the number of bits used to identify a link k) determines the fill factor ρ, which then determines the false positive probability. The higher the fill factor, the higher the probability of false positives. In other words, when encoding bigger multicast trees, we can expect larger average number of false positives with a given Bloom filter length m_1. However, if we use a longer filter length m_2, we could reduce the false positive probability for the same multicast tree, thus improving efficiency, with the penalty of larger packet headers. Vice versa, multicast trees containing only a couple of receivers would require fewer bits for efficient packet delivery and would save on per-packet overhead, which is important for applications where average payload sizes are small.

We propose a scheme to implement the variable-length Bloom filters. First, a long filter (e.g. $M = 8000$ bits) is created for the multicast three. Then, the fill factor ρ, i.e. the number of 1-bits divided by the filter length, is computed. This allows us to compute what would have been the the optimal length of the filter: $m = \lceil -M \log_2(1-\rho) \rceil$. A filter of length m would result in a fill factor 50% for the tree. We then fold the long filter into one of length m as follows:

$$iBF[i] = \bigvee_{j=0 \ldots \lfloor M/m \rfloor} longBF[j \cdot m + i] \text{ for } i = 0 \ldots m-1.$$

(Note that the elements beyond the array boundaries are considered to have value 0. Also, the fill factor of the resulting vector may, by change, be above 50%, in which case m should be decremented by one until the fill factor is below the limit.) The forwarding algorithm is modified in such a way that the k hash functions $f_{1 \ldots i}$ used to compute the locations of the of the 1-bits in the link masks are modified to be $f'_i(x) = f_i(x) \bmod m$.

B. Bit permutation of the Bloom filter

In two of the forwarding anomalies introduced above, routing loops and flow duplication, the same nodes forward the same packet (or its exact copy) multiple times. Since we do not want to store any state in the forwarding nodes, it is not possible to detect the recurrence of the packet. However, the packets do differ in their history, i.e. the path which they have already traversed. Only one occurrence has traversed exactly the path that was intended by the routing algorithm while the others have taken some anomalous path and then rejoined the intended route. If we could make the forwarding decision dependent on the path already taken by the packet, then we could prevent the copies with different history from following the same path forward.

The history-dependent forwarding can be achieved by accumulating in the packet information about the traversed path. For example, loops could be prevented by adding a hop counter or TTL value in the packet and including it as an input to the computation of the Bloom filter and link masks. This does not, however, prevent flow duplication where the path length is the same for both copies of the flow (e.g. Figure 2(a)). Hence, we need an efficient way to include more information about the history into the packet than just the hop count.

Our solution is to *perform a bit permutation of the Bloom filter on each traversed link*. The cumulative permutation of the filter along the forwarding path, in effect, makes the forwarding decisions dependent on the path already traversed by the packet. Every router should select a random permutation for its inbound links and apply this permutation to the filters of all packets arriving on that link. The forwarding algorithm is shown in Figure 3.

It should be noted that the filters for packet headers can no longer be computed simply as an OR of the link masks of the multicast tree. Instead, they need to be computed starting from the leaves of the tree (i.e. subscribers) towards the root by alternatingly ORing the filter with a link mask and permuting the filter (using the inverse permutation of the up-link). Similar to previous protocols, it is still possible to compute the filters for individual paths and OR them at the sender, or to accumulate the subtree filters along the tree.

There are a few reasons for choosing bit permutations as the technique for making forwarding dependent on the history of the packet. First, (static) bit permutations can be efficiently implemented in hardware. Second, performing a transformation on the filter does not consume any additional space in the packet header. What is needed is a pair of transformations for Bloom filters: f to be performed on forwarded packets and f' for computing the filters in the reverse direction. These transformation must meet the following conditions:

1) The process of forward transformation, setting additional bits, and reverse transformation must not cause any bit

```
Algorithm 1: packet forwarding

Input: LinkMasks of the node;
  Permutations of the node;
  iBF in the packet header;
let π = Permutations[ingress link]
set iBF in packet to π(iBF);
foreach outgoing link l do
  let mask = LinkMasks[ingress link, l]
  if iBF & mask == mask then
    Forward packet on the outgoing link l
  end
end
```

Fig. 3. Pseudocode for packet forwarding

to be unset in the filter: $\forall$ filters $x, y \exists z:\ f'(f(x) \vee y) = x \vee z$.

2) In order to avoid unnecessary false positives and conserve filter capacity, neither transformation should (significantly) increase the number of 1-bits in the filter.

The only transformations that strictly meet the two criteria are bit permutations of the filter, with f' being the inverse permutation of f.

In order to understand the effect of the bit permutations, consider first a forwarding architecture without the permutations. Whether false positives can cause forwarding loops depends on the topology and enforced forwarding policies of the network. For example, tree-structured networks and ones with a valley-free forwarding policy cannot have forwarding loops. On the other hand, in a highly connected network with many redundant routes and few rigid policies, such as the Internet, loops may arise easily. If any node in the multicast tree has a redundant link to a node higher in the same tree, there is a relatively high probability that the packet will loop back. The Bloom filter based forwarding allows false positive rates of up to several percent and all it takes to create a loop is *one false positive* on such a redundant link. Without the bit permutations, every packet in the flow will then keep going around the loop, producing a duplicate to downstream subtree at every loop..

When a bit permutation is applied to the filter at every hop, the first false positive is just as likely to occur but, after that, the packet will usually not match the link masks of its old route. Every further hop along the looping path requires another false positive and, intuitively, we would expect the packet to be dropped soon. This intuition is, however, slightly misleading as infinite loops are still possible: the set of bits tested by the link masks around the loop fall into some cycles of the permutations, and the packet will go into an infinite loop if and only if all bits in these cycles happen to be 1.

1) Upper bound on infinite loop probability:: Assume that a false positive causes a packet to be forwarded back to a node that it has previously traversed. Let us number the nodes in the looping path as $1 \ldots n$ (with the first recurring node numbered 1). Denote with π_i the permutation applied by the ith node in the loop, with β_i the link mask of the egress link from the ith router, with k_i the number of randomly selected bits set to 1 in β_i (because of collisions, the actual number of 1-bits may be lower than k_i), and with F the value of the Bloom filter in the packet before the 1st router of the loop processes it for the second time.

In order for the packet to go around the loop for a second time, specific bits in it must be set. The bit mask indicating those bits is $B = \vee_{i=1\ldots n}(\pi_1^{-1} \circ \ldots \circ \pi_i^{-1})(\beta_i)$. That is, the packet will go around the entire loop for a second time if $F \wedge B = B$. Let us further denote the loop permutation as $\pi = \pi_1 \circ \ldots \circ \pi_n$. In order for the packets to keep going around the loop for an unlimited number of times, the following has to hold: $F \wedge \pi^j B = \pi^j B$ for $j = 0, 1, 2, \ldots$. In other words, the bits set in B must belong to cycles of the permutation π that have all bits set in F. It remains to calculate the probability this.

B is effectively a bit mask of length m where $K = \sum k_i$ randomly selected bits have been set to 1. Since collisions are possible, the actual number of 1-bits in B will be some $1 \leq b \leq K$. We want to know the probability $P(b|K, m)$ of the K random selections from m bits resulting in exactly b distinct values. There are m^K possible mappings from K to m. The number of masks B with exactly b bits set is $\binom{m}{b}$. For each such B, there are $t_b = b^K - \sum_{i=1\ldots b-1}\binom{b}{i}t_i$ ways to map the K selections into the b bits. (There are b^K mappings from K to b and we exclude those that map into fewer than b distinct values.) This means that $\binom{m}{b}t_b$ out of the m^K possible selections result in exactly b 1-bits in B. Thus,

$$P(b|K, m) = \binom{m}{b} t_b / m^K$$

The values of t_b are easy to compute recursively.

Next, given b, we determine an upper bound for the probability $P(\text{infinite loop}|b, m)$ of the b 1-bits in B falling into cycles of π that have only 1-bits. Note that since π is a composition of random permutations, it itself has the statistical properties of a random permutation. The 1-bits of B always fall into some cycles of π, and we denote the $b + \delta$ bits in these cycles with Y.

$b + \delta < \rho \cdot m$ is a necessary condition for the infinite loop, because otherwise there are not enough 1-bits in F. We want to count the permutations that meet this condition. For any δ, there are $\binom{m-b}{\delta}$ ways to choose Y. For any such choice of Y, there are $(m - b - \delta)!$ permutations of the bits that are not in Y. The number of relevant permutations of Y has the upper bound $(b+\delta)!$ (Some permutations of Y are redundant because they have cycles that contain no 1-bits of B.) Combining these, we get an upper bound for the number of permutations that meet the condition $b + \delta < \rho \cdot m$ for a given δ: $\binom{m-b}{\delta}(m - b - \delta)!(b + \delta)!$. This allows us to compute an upper bound on the probability of infinite loops for a given b. We take into account the fact that π is a random one from the set of all $m!$ permutations and that the δ bits all must have value 1.

TABLE II
THE PROBABILITY OF INFINITE LOOPS WITH VARYING m AND K GIVEN $\rho = 0.5$

K	m = 64	m = 128	m = 256	m = 800
3	$4.1 \cdot 10^{-4}$	$5.1 \cdot 10^{-5}$	$6.4 \cdot 10^{-6}$	$2.1 \cdot 10^{-7}$
5	$7.8 \cdot 10^{-6}$	$2.4 \cdot 10^{-7}$	$7.6 \cdot 10^{-9}$	$2.6 \cdot 10^{-11}$
7	$3.1 \cdot 10^{-7}$	$2.5 \cdot 10^{-9}$	$1.9 \cdot 10^{-11}$	$6.6 \cdot 10^{-15}$
13	$3.5 \cdot 10^{-10}$	$4.3 \cdot 10^{-14}$	$5.2 \cdot 10^{-18}$	$1.9 \cdot 10^{-24}$
17	$1.9 \cdot 10^{-11}$	$1.4 \cdot 10^{-16}$	$1.1 \cdot 10^{-21}$	$4.3 \cdot 10^{-30}$
21	$2.4 \cdot 10^{-12}$	$1.2 \cdot 10^{-18}$	$5.9 \cdot 10^{-25}$	$2.4 \cdot 10^{-35}$

$$
\begin{aligned}
&P(\text{infinite loop}|b,m) \\
&\leq \sum_{\delta=0}^{\rho m-b} \rho^{\delta} \frac{1}{m!} \binom{m-b}{\delta} (m-b-\delta)!(b+\delta)! \\
&= \binom{m}{b}^{-1} \sum_{\delta=0}^{\rho m-b} \rho^{\delta} \binom{b+\delta}{b}
\end{aligned}
$$

Finally, we get an upper bound for the probability that the packet which has looped back once will stay in the loop for ever.

$$
\begin{aligned}
P(\text{infinite loop}|K,m) &= P(\text{infinite loop}|b,m)\ P(b|K,m) \\
&< m^{-K} \cdot \sum_{b=1}^{K} t_b \cdot \sum_{\delta=0}^{\rho \cdot m} \rho^{\delta} \cdot \binom{b+\delta}{\delta}
\end{aligned}
$$

Probabilities for some values m and K are shown in Table II. (Recall that K is the total number of bits se in the link masks around the loop.)

2) Probability of duplicate flows with permutations: A duplicate flow can happen when a false positive causes a packet to be forwarded to a router that is part of the forwarding tree. When the random permutations described above are in use, the false positives packets will not be automatically be forwarded down the tree from the node where they have landed. Because of the randomizing effect, a false positive must occur at every further hop. This is just as unlikely as the propagation of any false positive. Thus, as long as the stability condition of Section III-A is met, flow duplication will not take place. This result applies both to the situations of Figure 2 and to the copies sent downstream from a loop as in Figure 1.

V. INTER-DOMAIN MULTICAST

In this Section, we provide a rough sketch of BloomCast architecture that is designed to interconnect source specific multicast protocols. Unlike traditional IP multicast approaches, where the forwarding information is installed in routers on the delivery tree, in BloomCast, transit routers do not keep any group-specific state. It is influenced by Free Riding Multicast [19] and Automatic IP Multicast Without Explicit Tunnels (AMT) [21].

The rationale for this work is the following. First, we believe the state requirements of traditional multicast to be prohibitive for large scale and widespread use. Second, the analysis in FRM [19] shows that bandwidth requirements for unicast-based large scale inter-domain multicast are unreasonably high. Thirdly, FRM requires that each AS has a correct up-to-date view of the routing topology with full routing vectors for all destinations. Neither assumption holds: route changes propagate slowly and using BGP route aggregation loses the path vector information FRM needs.

We use BloomCast to evaluate our enhancements for Bloom filter-based multicast via simulations.

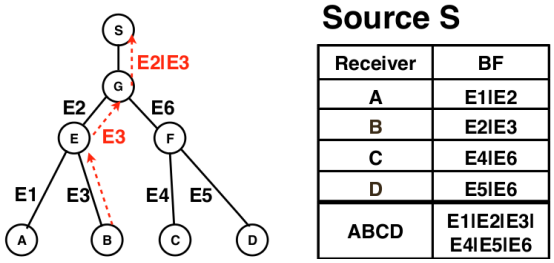

Fig. 4. The left side shows multicast Join message using Bloom filters. The right side shows a simplified Membership Table MT(S) that contains the Bloom filters for A, B, C, and D. The separated bottom row shows how to combine the Bloom filters in to an Bloom filter.

A. BloomCast protocol

BloomCast is an inter-domain protocol, operating between border routers in the AS hosting the source (source AS) and the border routers of the ASes hosting the receivers (receiver ASes). However, for the sake of simplicity, we will treat each AS as a single node.

When an AS joins a multicast group, it sends a BloomCast Join (`BC_JOIN`) message towards the source AS. The message contains an initially empty *collector* Bloom filter. While the message travels upstream towards the source, each AS records forwarding information in the control packet by inserting the corresponding link mask into a *collector*. After this, it performs a bit permutation on the *collector*. An illustration of this process (without the permutations) is shown in Figure 4.

Once the `BC_JOIN` message reaches the AS of the source, the *collector* holds sufficient information so that the source can send source-routing style packets to the recently joined AS. The source AS stores this information in the Membership Table (MT), as shown in Figure 4. The source AS can now combine the individual Bloom filters by bitwise ORing them and send packets to the multicast tree. It can split a large group into several Bloom filters.

When an AS expresses its desire to leave a multicast group, it will send a BloomCast Leave (`BC_LEAVE`) message specifying (S,G) to the source AS. Intermediate routers do not need to process this packet. Hence, unlike pruning packets in IP multicast, control packets are transparently routed to the source of the tree. Upon receiving the message, the source removes the receiver AS from the MT and reconstructs the forwarding Bloom filter for the group. An example of a combined Bloom filter for the group is shown in Figure 4 at the separated bottom row of the table.

When source AS sends a packet to a multicast group, it adds a BloomCast Forwarding (`BC_FW`) header to the packet, which contains the *forwarding* Bloom filter. Each intermediate AS makes its forwarding decision by performing a reverse bit permutation on the *forwarding* Bloom filter and then querying it with the question: which of my outgoing links are present in the Bloom filter?

For each match found, the packet is forwarded to the corresponding neighboring ASes. Eventually, the packet reaches all the destination ASes, following the sequence of ASes the `BC_JOIN` packets traversed in reverse order.

The BloomCast architecture shifts the group and tree management to the source and, hence, requires less processing and state at the transit routers. Computing the link mask can be done on line speed using e.g. a fast spreading hash function (e.g. [15], [26]). BloomCast requires more processing and state at the border routers in the source domain. However, we believe that the additional complexity is reasonable considering that the source domain is the main beneficiary of the inter-domain multicast architecture.

VI. Evaluation

In this section, we evaluate our solutions using the BloomCast architecture and an inter-domain AS topology scenario. We evaluated the effects of *varying k scheme* and *bit permutations* with the CAIDA AS relationship data set[3] and using shortest valley free paths [11] for every pair of ASes. Loop prevention was stress tested with an artificial simulation setup of three nodes forming a ring.

A. Scalability with variable k

The multicast forwarding process was simulated with constant and variable k for group sizes between 5 and 30 random ASes and multicast trees based on valley free shortest paths to each destination AS. For each parameter set, 5000 rounds were run.

In the *varying k scheme*, each AS_i sets $k_i = \lceil log_2(d_i) \rceil$, with d_i being the AS degree, i.e., the number of neighboring domains. The k_i hashes are generated using the double hashing technique, with MD5 and SHA1 as the independent hash functions.

Forwarding efficiency was measured as the ratio of Bloom filter edges to the total number of edges traversed (including false positives and packet duplications). False positives were recursively tested on the forwarding tree using the CAIDA AS topology, i.e. whenever a false positive was found, additional false positives were recursively tested by querying for Bloom filter presence of adjacent domains.

Figure 5(a) shows the forwarding efficiency for varying Bloom filter lengths and multicast group sizes. 20–25 receiving ASes ($\approx$ 40–50 edge-pair labels) is a practical limit to the group size with Bloom filter of 800-1024 bits. This keeps the fill ratio $\rho < 0.5$ and the forwarding efficiency at an acceptable level ($\geq$ 60–80%).

[3]Data set: as-rel.20091215.a0.01000.txt, http://as-rank.caida.org/

TABLE III
Simulation results, constant k vs. variable k.

group size	# transit AS $avg.$ (95^{th})	hashes k	fpr (%)		fill factor ρ	
			$avg.$	95^{th}	$avg.$	95^{th}
15	33.5 (38.5)	$k_c = 9$	0.027	0.050	0.31	0.35
		$k_{var}(b)$	0.015	0.031	0.32	0.35
20	43.5 (49.1)	$k_c = 9$	0.062	0.109	0.38	0.43
		$k_{var}(b)$	0.032	0.057	0.39	0.42
25	53.1 (59.5)	$k_c = 9$	0.143	0.252	0.45	0.49
		$k_{var}(b)$	0.069	0.119	0.45	0.49

TABLE IV
Simulation results, permutation vs. non-permutation. For $m = 800, k_c = 9$.

group size	rdm. perm.	fp per domain		duplicates (#)	
		$avg.$	95^{th}	$avg.$	95^{th}
15	NO	0.22	0.45	0.005	0.061
	YES	0.22	0.45	0.004	0.060
25	NO	0.61	0.83	0.032	0.121
	YES	0.61	0.81	0.023	0.061

Figure 5(b) shows the rank distribution of false positives per domain with 100-byte Bloom filters and varying group sizes when using both the bit permutation technique and the varying k scheme. As expected, the majority of transit ASes are false-positive-free, whereas more than 5 false positives per domain correspond to less than 1% of the cases.

Table III compares the observed false positive rate (fpr) between constant k_c and variable k_{var} for multicast group sizes between 15 and 25 AS receivers. The results show that, in inter-domain AS topologies, the logarithmic computation of k_i results in a factor 2 improvement in fpr for the same receiver group while keeping the average fill factor approximately the same as in the constant k scheme.

B. Duplication prevention

Using the same simulation setup, we now focus on the incidence of duplicate flows, i.e., the amount of duplicated traffic delivered to the destination AS due to false positives. The number of duplicate flows was measured as the sum of falsely forwarded flows each receiver AS received. Our results in Table IV show that using bit permutations contributes to a reduction in the number of duplications.

The drop in flow duplications is relatively small because many of the flow duplications encountered are caused by false positives directly to a recipient AS (case shown in Fig. 2(a)), instead of causing the flow to return to the forwarding tree.

C. Loop-freeness using bit permutations

Bit permutation-based loop prevention was stress tested with an artificial topology of three nodes, interconnected as a directional ring. An m-bit Bloom filter was generated by inserting random link masks from a pool of 3M unique randomly generated 32-bit identifiers until ρ_{max} was reached. 100 different test link masks were randomly chosen for each

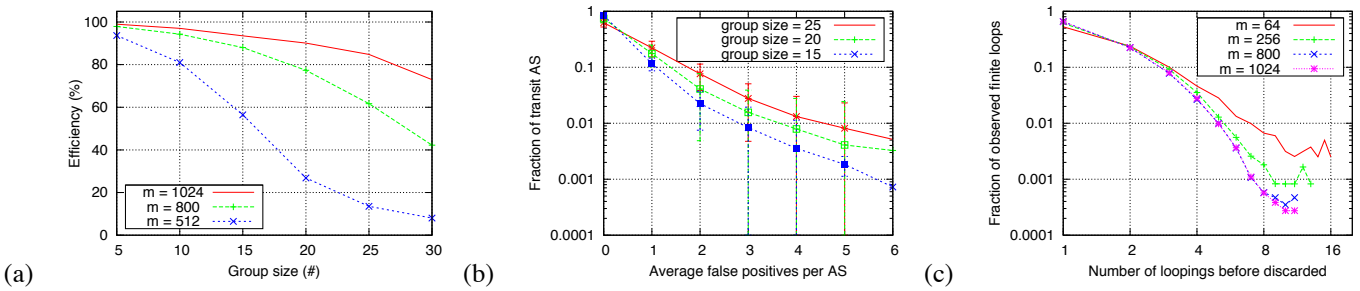

Fig. 5. (a) Average forwarding efficiency for different filter sizes m in function of the multicast group size. (b) Rank distribution of the average false positives per domain using *varying* k, bit permutations and $m = 800$. (c) Distribution of the number of finite loops observed when $\rho = 0.95$ and $k = 7$.

multicast node inserted in the Bloom filter. The presence of false positives was tested recursively starting with an initial router. The incidence of loops was tested using packets with randomly filled Bloom filters both with and without bit per-router permutations. This setup resulted in at least 10.000 rounds per parameter set; for some configurations over 10M rounds were run so that some loops could be observed.

Table VI-C shows the results in terms of observed loop rate with and without per-router bit permutations for varying target fill factors ($\rho_i \in [0.5, 0.75, 0.95, 0.98]$),[4] number of hash functions ($k_c \in [5, 7, 13, 17, 21]$), and 800-bit Bloom filters.

Not a single infinite loop was observed when bit permutations were used. In comparison, loop events of plain Bloom filters were rare but observable for $\rho = 0.5$ and $k \leq 7$ and become common ($> 1\%$) for $\rho \geq 0.75$.

The results are consistent with the theoretical predictions. Loop events were effectively contained due to per-node bit permutations for practical fill factors ($\rho < 0.75$). Only for *very* filled Bloom filters ($\rho \geq 0.95$) some amount of *finite loop* instances could be observed. In those cases, as expected, increasing k and m reduces the amount of loopings (i.e., finite loops before the packet is discarded. This effect and the relation to m can be observed in Fig. 5(c) for $\rho = 0.95$. Even for 800-bit Bloom filters filled with 760 ones, the majority of the loopings lasted only for 1 or 2 cycles. The longest finite loop observed was for $\rho = 0.98$ (i.e., 784 bits set) where a packet looped as many as 58 times before dying out. There were only anecdotic instances of infinite loops when using permutations for small filters ($m = 256$ and $m = 64$) and always for unrealistic fill factors $\rho \geq 0.9$.

D. Discussion

The three discussed anomalies – packet storms, forwarding loops, and flow duplication – are especially problematic because, if they occur, they do not just affect a single packet but every packet in a flow. Hence, even if the probability of a problem is low, a few rare instances can cause major problems in a network.

Packet storms can be prevented by local decisions, i.e. varying the k_i to ensure that the average number of false positives is below the threshold of one. Allowing each AS to vary k_i gives it a policy tool that can be used to control local false positives. First, peering policies often restrict the set of possible outbound links depending on the ingress link of a packet. For this reason, the out-degree of the node will be smaller if it is calculated individually for each inbound link. Consequently, determining the value of k separately for each inbound link may help save filter capacity. Second, an AS can vary the value of k for inbound/outbound combinations e.g. based on traffic volumes, or importance of a particular link. Decisions about such optimizations are matters of local policy as long as every router meets the stability condition of not amplifying false positives.

[4]While ρ values larger than 0.5 are not practical due instability packet storms as discussed earlier, we use them solely for the purpose of illustrating the effectiveness of permutations in preventing loops.

TABLE V
FRACTION OF OBSERVED INFINITE LOOPS WITH PERMUTATIONS (P) AND WITHOUT (NP) FOR VARYING ρ, K AND $m = 800$. NORMALIZED BY THE NUMBER OF SYNTHETIC LOOP TRIALS.

K	$\rho = 0.5$		$\rho = 0.75$		$\rho = 0.95$		$\rho = 0.98$	
	P	nP	P	nP	P	nP	P	nP
5	0	$2.8 \cdot 10^{-5}$	0	$1.37 \cdot 10^{-2}$	0	0.46	0	0.74
7	0	$2.8 \cdot 10^{-7}$	0	$2.4 \cdot 10^{-2}$	0	0.34	0	0.66
13	0	0	0	$1.5 \cdot 10^{-5}$	0	0.14	0	0.46
17	0	0	0	0	0	0.08	0	0.36
21	0	0	0	0	0	0.05	0	0.29

In Bloom filter based multicast, loops cause every packet in the flow to loop in the network and for every loop, a copy will be sent to downstream receivers. In theory, local routing policies prevent loops in BGP. In practice, however, BGP instabilities and configuration errors do cause routing loops in the Internet (see e.g. [24]). For these reasons, local routing policies should not be the only technique used to prevent loops

1) Using varying k and bit permutations: Bit permutations are easy to use in FRM [19], LIPSIN [12], and MPSS [27]. FRM uses AS number pairs for Bloom mask computation and the source is assumed to know the AS level path to destination. The same AS pairs can be used for computing the bit permutation performed on the Bloom filter. In LIPSIN and MPSS, the routers can communicate the permutation they use to the topology manager or PCE, respectively.

Varying k is easy to utilize both when there is a centralized topology manager and when the Bloom filter is collected hop by hop. This makes it simple for LIPSIN and MPSS. In FRM, either a new protocol is needed or BGP would need to be

augmented to distribute the k values of each AS.

2) Route failures: A route failure can cause multiple receivers to disconnect simultaneously. The Bloom filter in the packet can be used to reverse route an ICMP error packet back to the source AS, since the use of both previous hop and next hop AS ensures that the Bloom filter can be used bidirectionally. Multipath routing that gives source control over paths (see e.g. [17]) could be used to provide alternative paths to the destinations in advance.

3) Security: The three anomalies described (packet storms, forwarding loops, and flow duplication) present opportunities for denial of service attacks. Each of them can be used for traffic amplification attack: packet storms against the forwarding infrastructure and forwarding loops and flow duplication both against the network and a target recipient. The combination of varying k and globally enforced maximum fill factor efficiently prevents an attacker from causing packet storms. Random bit permutations make it harder for an attacker to cause a packet to loop or a flow duplication. They also ensure that even if a packet loops, the duplicates will not necessarily be forwarded to the downstream subtree.

VII. Conclusions

Bloom filters have been proposed as a scalable solution for multicast. Their nature causes random false positives that cause packets to be forwarded over some additional links. We identified three anomalies in the forwarding plane that arise from the probabilistic multicast: packet storms, forwarding loops, and flow duplication. It is noteworthy that all packets sent to the same multicast group have the same set of false positives (rather than being randomly distributed), exacerbating the problem. The identified anomalies could also occur in other forwarding technologies that follow a similar probabilistic approach. Our evaluation shows that packet storms can be efficiently mitigated by adapting the number of hash functions to the node degree while forwarding loops and flow duplication can be prevented using per hop bit permutations on the Bloom filter.

We also introduced BloomCast, a source-specific inter-domain multicast architecture based on in-packet Bloom filters. We intend to continue our work by improving fault tolerance via multipath routing and further evaluating scalability and fault tolerance of BloomCast. The possibilities for preventing denial-of-service attacks will be studied in the future. We also plan to evaluate the impact of ASes varying the size of the link mask per inbound/outbound link pair using local policies and experiment with safe (false-negative-free) bit deletions. Finally, incentives for adoption by the different stakeholders along with overlay and partial deployment strategies are also part of our research agenda.

References

[1] T. Anderson, T. Roscoe, and D. Wetherall. Preventing Internet denial-of-service with capabilities. In *Hotnets II*. ACM, 2004.

[2] N. Bhaskar, A. Gall, J. Lingard, and S. Venaas. Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM). RFC 5059 (Proposed Standard), Jan. 2008.

[3] S. Bhattacharyya. An Overview of Source-Specific Multicast (SSM). RFC 3569 (Informational), July 2003.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[5] C. Diot, W. Dabbous, and J. Crowcroft. Multipoint communication: A survey of protocols, functions, and mechanisms. *IEEE Journal on Selected Areas in Communications*, 15(3):277–290, 1997.

[6] A. Ermolinskiy. The Design and Implementation of Free Riding Multicast. Master's thesis, UCB, May 2007.

[7] C. Esteve, P. Jokela, P. Nikander, M. Särelä, and J. Ylitalo. Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters. *EC2ND*, 2009.

[8] D. Farinacci and Y. Cai. Anycast-RP Using Protocol Independent Multicast (PIM). RFC 4610 (Proposed Standard), Aug. 2006.

[9] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601 (Proposed Standard), Aug. 2006.

[10] B. Fenner and D. Meyer. Multicast Source Discovery Protocol (MSDP). RFC 3618 (Experimental), Oct. 2003.

[11] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions On Networking*, 9(6):733–745, 2001.

[12] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander. LIPSIN: Line speed publish/subscribe inter-networking. In *SIGCOMM*, 2009.

[13] E. Karpilovsky, L. Breslau, A. Gerber, and S. Sen. Multicast redux: a first look at enterprise multicast traffic. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009.

[14] D. Kim, D. Meyer, H. Kilmer, and D. Farinacci. Anycast Rendevous Point (RP) mechanism using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP). RFC 3446 (Informational), Jan. 2003.

[15] H. Krawczyk. LFSR-based hashing and authentication. In *Advances in Cryptology–CRYPTO'94*, pages 129–139. Springer, 1994.

[16] Y. Lu, B. Prabhakar, and F. Bonomi. Bloom filters: Design innovations and novel applications. In *the Annual Allerton Conference*, 2005.

[17] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala. Path splicing. In *ACM SIGCOMM*. ACM, 2008.

[18] P. Paul and S. V. Raghavan. Survey of multicast routing algorithms and protocols. In *ICCC*, pages 902–926, 2002.

[19] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. *ACM SIGCOMM Computer Communication Review*, 36(4):26, 2006.

[20] C. E. Rothenberg, C. Macapuna, F. Verdi, and M. Magalhes. The deletable bloom filter: a new member of the bloom family. *IEEE Communications Letters*, 14(6):557 – 559, June 2010.

[21] D. Thaler, M. Talwar, A. Aggarwal, L. Vicisano, and T. Pusateri. Automatic IP Multicast Without Explicit Tunnels (AMT). Internet-Draft draft-ietf-mboned-auto-multicast-10, IETF, Mar 2010. Work in progress.

[22] X. Tian, Y. Cheng, and B. Liu. Design of a scalable multicast scheme with an application-network cross-layer approach. *IEEE Transactions on Multimedia*, 11(6), 2009.

[23] A. Whitaker and D. Wetherall. Forwarding without Loops in Icarus. In *Open Architectures and Network Programming*, pages 63–75, 2002.

[24] J. Xia, L. Gao, and T. Fei. Flooding attacks by exploiting persistent forwarding loops. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, page 36. USENIX Association, 2005.

[25] J. Xia, L. Gao, and T. Fei. A measurement study of persistent forwarding loops on the Internet. *Computer Networks*, 51(17):4780–4796, 2007.

[26] K. Yuksel. *Universal hashing for ultra-low-power cryptographic hardware applications*. PhD thesis, Citeseer, 2004.

[27] A. Zahemszky, P. Jokela, M. Särelä, S. Ruponen, J. Kempf, and P. Nikander. MPSS: Multiprotocol Stateless Switching. In *Global Internet Symposium 2010*, 2010.

[28] M. Zhong, P. Lu, K. Shen, and J. Seiferas. Optimizing data popularity conscious bloom filters. In *ACM PODC*, pages 355–364, 2008.