isBF: Scalable In-Packet Bloom Filter Based Multicast

Ilya Nikolavskiy[a,*], Andrey Lukyanenko[a], Tatiana Polishchuk[b], Valentin Polishchuk[c], Andrei Gurtov[a,b,d]

[a] *Aalto University, Finland*
[b] *Helsinki Institute for Information Technology HIIT, Finland*
[c] *University of Helsinki, Helsinki, Finland*
[d] *ITMO University, Russia*

Abstract

Bloom-filter (BF) based forwarding was proposed recently in several protocol alternatives to IP multicast. Some of these protocols avoid the state in intermediate routers and leave the burden of scalability management to the multicast source and end-hosts. Still, the existing BF-based protocols have scalability limitations and require explicit network management as well as non-trivial functionality from the network components.

In this work we address the scalability limitations of the BF-based forwarding protocols by partitioning end-hosts into clusters. We propose several algorithms to do the partitioning so as to decrease the overall traffic in the network. We evaluate our algorithms in a real Internet topology, demonstrating the ability of the proposed design to save up to 70% of traffic volume in the large-scale topology for big groups of subscribers, and up to 30% for small groups.

Keywords: In-packet Bloom filters, Multicast, Internet, Architecture

1. Introduction

Today's Internet contains large volumes of network traffic with one common pattern: almost each piece of the data is interesting for multiple recipients (for example, BitTorrent, YouTube, IPTV traffic, etc). The efficiency is extremely hard to achieve in current networks. A few successful examples of traffic savings for networking are web caches and commercial CDNs [11]. There are even attempts to extend CDN publicly to a peer-to-peer (P2P) environment [6]. The main limitation of CDN approach is commercial-primary deployment unacceptable to private users.

Another method for traffic savings is multicast [4] where the network helps to create a minimal amount of traffic until some point in which the traffic can be cloned to multiple copies, and each copy forwarded to interested recipients. As opposed to CDN architecture, multicast benefit small multicast groups. IP multicast, however, still failed to be widely deployed. There are several reasons for that. First, with the current IP network, routers require state for each subscribing group in each router. As the number of possible subscribers may be more than millions and each subscriber has multiple groups in active use, a middle router can easily become overcrowded with heavy lookups over big tables and memory usage. The latter is a severe problem for core routers. Second, multicast deployment requires provider incentives. Fortunately, there is still positive evidence that hardware manufactures are willing to put most of the multicast protocols into routers [18].

Some researchers suggest clean-slate redesign of the current Internet, where multicast is considered as one of the most important network properties [13]. An elegant solution, firstly proposed in [18] as part of

*Corresponding author. Address: PO Box 19800, 00076 Aalto, Finland. Phone: +358505758927
Email addresses: e-mail:ilya.nikolaevskiy@aalto.fi (Ilya Nikolavskiy), andrey.lukyanenko@aalto.fi (Andrey Lukyanenko), tatiana.polishchuk@hiit.fi (Tatiana Polishchuk), valentin.polishchuk@helsinki.fi (Valentin Polishchuk), gurtov@hiit.fi (Andrei Gurtov)

a control plane, was adopted in the form of *in-packet Bloom filters* (*iBF*) as the data plane solution [10]. iBF introduces multicast opportunities without the active router's state; however, iBF still has scalability limitations.

We emphasize that scalability, economical considerations and the complexity of network components create barriers for wide multicast deployment, independently of the architecture. In this work we extend iBF based forwarding for networks with global flat forwarding fabric deployed around the Internet. We propose new mechanisms for scalability and management by introducing a novel *in-packet scalable Bloom filters* (*isBF*) protocol. We attempt to avoid network component complexity by utilizing existing IP fields for our purposes.

The contributions of this work are twofold. First, we design a number of new algorithms based on flat BF labeling, which are able to scale to the whole Internet. Next, we evaluate our algorithms in a real Internet topology (CAIDA data set).

The rest of the paper is organized as follows. In Section 2 we describe the related work and Bloom filters basics. We propose four novel algorithms to achieve scalability in Section 3. Section 4 presents evaluation of the proposed algorithms. In Section 5 we discuss economic incentives to utilize multicast. Section 6 concludes the paper.

2. Background and related work

Over two decades ago, Deering and Cheriton [3, 4] proposed IP multicast as an efficient solution for data dissemination from a single source to multiple receivers. It was deployed experimentally [16], but never adopted in any significant way by service providers. The failure of multicast [5] to achieve the wide spread adoption can be explained by several technical and economic factors, including complexity of the multicast network management and uncertainty in how to appropriately charge for the service in the case when sources and receivers belong to different domains. For many years multicast was implemented only locally within the service providers' networks supporting IPTV[21] and conferencing applications, and also in enterprise networks [12], where the aforementioned issues of management, billing and inter-provider dependencies are mitigated.

Due to its unquestionable capability to significantly reduce network load, multicast remains the most studied research problem in computer networking [17]. According to work [10] the main challenge in efficient *information centric network* (*ICN*) design is how to build a multicast infrastructure that could scale to the general Internet and tolerate its failure modes while achieving both low latency and efficient use of resources. In topic-based ICNs, the number of topics is large, while each topic may have only a few receivers [15]. IP multicast and application level multicast have scalability and efficiency limitations under such conditions. In IP multicast, the amount of routing state is proportional to the number of multicast groups. To address this issue, several multicast proposals [8, 10, 18, 23] implemented the idea of using *Bloom filters* (*BF*) [2] in the packet headers. This way the intermediate routers are purged from the burden of keeping states.

The authors of LIPSIN [10] proposed the multicast fabric, which uses the iBF directly for the forwarding decisions, removing the need for IP-addresses and proposing *Link IDs* (*LIDs*) as a generic indirection primitive. To implement full scale routing in the Internet, authors propose to use two levels of forwarding fabrics - intra-domain and inter-domain. Each AS should have a dedicated node which works as a gateway between intra-domain and inter-domain routing. Those nodes are called rendezvous (RVZ).

The iBF based forwarding fabric was utilized in several works afterwards. Some works are focused on datacenter networks [19]. But they are not extendable to internet-size topologies. Another work [14] utilizes Bloom filters on switch interfaces to encode all the multicast groups on the interface. However, this approach is also limited in scalability. Jokela et al. [9] utilized iBF to reduce multicast state, however this work considers only small number of clients in each multicast group and is not scalable to Internet-size networks.

Tapolcai et al. [22] propose same stateless design as we propose in our work. They utilize multi-stage iBF to encode several levels of data-dissemination tree. However, this solution requires variable size headers in packets. For internet-wide multicast that solution requires headers as long as 2 kilobyte. That greatly limits it's applicability for a wide-scale multicast.

Table 1: Comparison of multicast alternatives.

	Forwarding	Structure	Scalability in the number of groups	Scalability in the number of subscribers	Additional network elements	Traffic economy
IP multicast (ideal)	IP	Flat	No	Yes	No	Perfect
LIPSIN	iBF	Hierarchical	Yes	n/a	Yes (RVZ)	n/a
isBF	iBF+IP	Flat	Yes	Yes	No	Average

Heszberger et al. [7] also suggest to utilize iBF for multicast addressing. To limit size of Bloom filters they propose novel adaptive length Bloom filters. However, their solution does not scale up to whole internet either.

The packet header with iBFs creates false positives, which give rise to forwarding anomalies. Successful methods for their reduction are discussed by Sarela et al. [20], although they do not solve scalability problem in terms of the number of recipients.

Comparison to Our Approach

In Table 1 we compare our isBF protocol with two alternatives. While IP multicast and LIPSIN use stateful routers in the middle, they obviously achieve the most traffic reduction in the network. For LIPSIN this is due to the additional hardware components which should be deployed widely. However, when the number of RVZ servers increases (proportionally to the number of active ASes), the solution stops being scalable in terms of the number of subscribers. Consequently, some additional measures will be needed to cover all recipients with one BF, either a third layer of hierarchy (which will increase the per-packet overhead) or some smart splitting methods. The benefit of our approach is that we use algorithms on flat labels with minimal per-packet overhead, and our protocol can be easily extended with in-network RVZ servers (treated as clients). It is possible to add RVZ servers anywhere in the network for our protocol. RVZ server will be a gateway for all clients in the corresponding part of the network. RVZ in turn will work as a single client for any multicast server or other RVZ. With such RVZ servers our protocol comes closer to the ideal multicast performance. This is not possible for LIPSIN.

Bloom Filter Basics

A Bloom filter (BF) [1] for encoding a set X is a probabilistic data structure for membership queries in X. The filter consists of k independent hash functions $h_1, \ldots, h_k$ and a binary array of length m. Each function maps any element x that may belong to X to an integer chosen uniformly at random from $1 \ldots m$. To add an element x to the set, it is needed to get k array positions using the hash functions, and set bits in the array at these positions to 1. To test whether x is in set, feed x to each of the k functions to get k array positions. If any of the bits at these positions is 0, the element is definitely not in the set; if all are 1, then either the element is in the set, or the bits have by chance been set to 1 during insertion of other elements, resulting in a *false positive*.

In iBF are encoded all links which packet should traverse to reach all clients. The network routers simply forward the packet into all the local links which are encoded in the iBF. Obviously, if we set all bits in iBF (m bits set), then such a packet will be broadcasted to the whole network. For practical and security reasons it is advisable to keep the maximum *fill factor* (the ratio of bits set to m) in BFs below 50% [20].

3. Scalability of iBF

In large multicast networks the following issue arises: Any reasonably large set of far situated destinations produces a data delivery tree in which the LIDs are encoded in a iBF almost completely filled with $1s$. Such a BF exceeds the fill factor threshold and will encode all the links in the network causing undesirable packet flooding. One way to address this problem is to divide the set of destinations into several subsets and encode each of them separately as different multicast tree. Figure 1 shows an example: The iBF encoding all destinations is "11111100" which have more than half bits set to 1. The Figure also shows a possible clustering of destinations (E and F for iBF1, and D for iBF2); The iBF for each cluster has fill factor 50%.

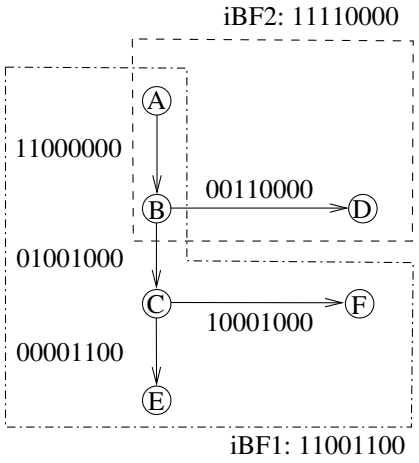

Figure 1: Example of data delivery tree with source A and destinations D, E and F.

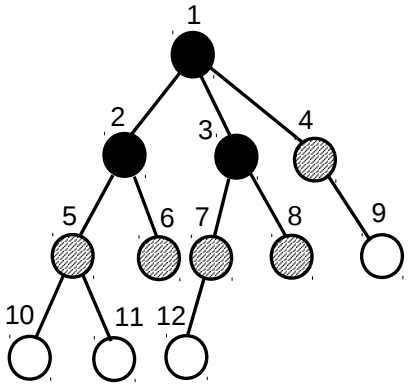

Figure 2: Example of a placement of invalid(1-3), active(4-8) and valid(9-12) nodes in a tree.

3.1. Problem statement

Multicast Tree Splitting Problem. *Given a data dissemination tree with a LID assigned to each link, find a grouping of tree leaves (destinations) that minimizes the total amount of traffic in the network caused by multicast transmission under the constraint that the fill factor threshold condition is satisfied for each of the resulting iBFs.*

It can be shown that our problem is NP-hard, by reduction from a classical NP-hard problem 3-PARTITION: Given $3m$ integers $a_1, \ldots, a_{3m}$, can they be split into m groups of 3 so that the sum in each triplet is the same? From an instance of 3-PARTITION we construct an instance of Multicast Tree Splitting problem. The tree will have $3m$ leaves. All leaves will be connected to a single node u, which is connected to the root r. The BF for a link from u to leaf i will have the number of set bits equal to a_i. BFs for links to different leaves will be disjoint, i.e. they will have set bits in different positions. The ur link will have exactly one bit set. The fill factor threshold is set in such a way that the maximum possible number of set bits in iBF is equal to $1+\sum a_i/3$. It is easy to see that the 3-PARTITION instance has a solution iff the constructed instance of our problem has a solution (with leaves grouped into the m triplets).

This means that finding the optimal solution to our problem is NP-hard. Below we experiment with heuristics for finding feasible solutions to our problem.

3.2. Algorithms

We suggest several tree splitting algorithms and evaluate their performance on a large realistic network topology.

Topology-oblivious random greedy merging

This is the simplest and fastest suggested algorithm. It does not require any knowledge about the network topology. BF encodings for the paths to each destination are given as an input. We add the BFs one by one in a random order to the group until the fill factor threshold of the resulting BF is reached, and then start filling the new group. The pseudocode for this algorithm is presented in the appendix (Algorithm 1). The running time of the proposed algorithm is linear in the number of receivers.

Topology-oblivious random greedy merging with preliminary sort

This method is a modification of the previous algorithm. The idea is to merge close destinations together. In the case when there is no information about the network topology it is impossible to measure the distance between two destination nodes. However, some heuristics may be used for that. We suggest the lexicographical sort to gather close destinations together. Before processing with the greedy merging, all

BFs must be sorted in a lexicographical order. Surprisingly, this simple modification significantly reduces the resulting network load. We refer to Section 4 for further details.

Obviously sorting contributes to the complexity of the algorithm $O(N \log N)$ component, where N is the number of receivers.

Topology-based splitting

This algorithm assumes that the full tree topology data is available at the sender. Alternatively it may be acquired from the *join* request packets. In the latter case, each router on the path from the destination to the source has to append its unique ID information to the *join* packet while processing it.

To minimize the amount of traffic it is reasonable to merge topologically close destinations together because it will allow to encode more destinations in one iBF and will reduce false-positive rate.

Let's consider the sets of destinations induced by some sub-tree in the data dissemination tree. For some node u in the tree we can define a corresponding set of destinations (leaves) in the subtree of this particular node as an *induced set*. It is obvious that for any two nodes where one node is a child of the other node, child's induced set is a subset of its parent's induced set. Thus the fill factor for the corresponding BFs is non-increasing along any path from source to destination. We call the node *valid* if its induced set produces a BF with the valid fill factor, and *invalid* otherwise. In a large tree, the root is an invalid node and all destinations are valid nodes. If the root is a valid node then no tree splitting is needed. If some of the destination nodes are invalid, it is impossible to serve them with the given fill factor threshold at all. Such nodes may be ignored, or processed individually as separate subsets, and are thus excluded from the current problem. It is obvious that an invalid node may have only an invalid parent. We define valid nodes where the parents are invalid as *active*. Those nodes form an edge between invalid and valid nodes (see Figure 2). It is clear that each destination has an active ancestor. Because no active node can be an ancestor to another, all of them induce disjoint sets of destinations.

The Topology-based tree splitting algorithm produces BFs corresponding to each active node. To compute active nodes, we calculate BFs for induced sets for all nodes, starting from the leaves up to the source. The pseudocode for this algorithm is presented in the appendix (Algorithm 2).

The running time of the algorithm is linear in the total number of tree nodes.

Topology-based splitting with greedy merging

This algorithm is a modification of the previous one. After computing active nodes we can greedily merge the resulting filters the same way as in the Topology-oblivious random greedy merging algorithm. The resulting active nodes are processed in such an order that all topologically close siblings are added one after another. It is essential to preserve the locality of the merged vertices. The merging operation does not contribute to the time asymptotic complexity of the algorithm, which stays linear in the number of tree nodes.

The first two algorithms do not use knowledge of the exact network topology. The availability of the topology data at the source requires that the intermediate routers append some extra information to the *join* packet. In case such functionality is not implemented, the first two methods are applicable. Otherwise topology aware algorithms are recommended due to their better performance as demonstrated in the following section.

4. Evaluation

In this section we evaluate performance of the proposed algorithms in the real Internet topology.

4.1. Experimental Network

We use a realistic network topology derived from CAIDA Router Level Topology Measurements computed from ITDK 0304 skitter and ifinder measurements[1]. The dataset represents a directed graph with 192244

[1] www.caida.org/tools/measurement/skitter/router_topology

nodes and 636643 directed links. In order to construct a multicast dissemination tree we choose the vertex from which almost all of the graph vertices are reachable as a source of multicast traffic, and select all the nodes reachable from that vertex. The resulting tree contains 190126 nodes and 606322 directed links. Because in the original data available from CAIDA all the end-hosts were omitted, we introduce 200000 additional nodes into random places to represent receivers. This number of end-hosts were chosen as it is reasonably large yet feasible to simulate.

For iBF size we fix 256 bits which correspond to the combined size of IPv6 source and destination fields. In order to decrease the false-positive rate we set the fill factor threshold at 50% as proposed by Sarela et al. [20].

For each directed link a random LID of the length 256 bits was generated and filled with exactly 8 non-zero bits. Parameter $k = 8$ was chosen because such a quantity produces reasonably small false-positive probability. Larger values of k could provide lower false-positive probability but limit the BF capacity to unacceptably small values. To approve the choice, we repeated all the evaluation experiments described later in this section for $k = 7$ and $k = 9$ non-zero bits, and found out that $k = 8$ is the best choice in respect to all our metrics. In order to reduce false positives, we use TTL field and random permutations as proposed by Sarela et al. [20].

4.2. Simulation Environment

We implemented a simulation framework to evaluate the performance of the proposed algorithms using C++ programming language. The existing network emulators (e.g. ns-3, OMNeT++) did not suit our purposes since they are quite complicated and slow for testing scalability and forwarding issues for the large topologies.

To investigate the performance of the algorithms on different scales, we repeated the experiments several times on smaller dissemination trees. For that purpose a *distance threshold* parameter was introduced, which is the maximum hop distance between two receiving hosts.

For each experiment the end-hosts selection was performed in the following manner. First, all end-hosts were permuted randomly. Then all the end-hosts unreachable from the first client within the distance threshold were disregarded. Finally all hosts except the first 10000 were also omitted. This way all the receivers have equal probability to be selected.

The parent of the lowest common ancestor (LCA) of all destinations was selected as a source. Therefore, we restricted our consideration to the case where the root has only one child. Otherwise there may be clients beyond different links grouped together. In this case, grouping does not give any improvement because the paths to those clients have no shared links and, moreover, the false-positive rate rises due to the larger fill factor.

By introducing a distance threshold we emulated a multicast behavior in a single autonomous system, or some geographical area. The distance threshold parameter defining the size of the dissemination tree varied in the range 2 to 21. The distances more than 21 were not taken into consideration since most of the end-hosts within the full network are reachable within 21 hops. Each test was repeated 100 times.

4.3. Metrics

A fundamental metric in evaluation of the packet level transmission protocols is the number of packets sent within the network which defines the *network load* L. The lower bound for this metric is the number of links needed to cover all the receivers L_{min}. It is theoretically possible to achieve this lower bound value using a stateful multicast protocol. BF based multicast could reach the lower bound only in the case where all the receivers were encoded with a single BF which produced no false-positive traffic. In large networks this is impractical since such a setup results in either unacceptably large packet header sizes or critical false positive rates. Therefore, the increase of the network load is a trade-off for maintaining the stateless multicast protocol. The upper bound on the network load L_{max} is the sum of lengths of all paths from the source to destinations. In the worst case scenario, each destination is encoded with a separate BF leading to the unicast design.

For performance evaluation we use the following metrics (i) *network overhead*, (ii) *network economy* and (iii) *false-positive rate*:

$$Overhead = \frac{L - L_{min}}{L_{min}}; \quad Economy = \frac{L_{max} - L}{L_{max}}; \quad fpa = \frac{\#\text{Unintended packets}}{\#\text{All sent packets}}$$

The network overhead shows how bad the algorithm performs in comparison to the stateful multicast protocol. Network economy defines the multicast protocol gain in comparison to the unicast transmission. Calculated during experiments, fpa defines the ratio of the number of unintended packets to the total number of packets sent within the network. Unintended packets include packets in cycles, duplicated packets and packets sent through the links not included in the data dissemination tree.

Additionally, we calculate a *filter density*, which is the number of links encoded in each BF. This value helps to determine whether there is a room left for further improvement of the algorithms.

4.4. Results

First we compare the network economy resulting from different tree splitting methods. Figure 3 confirms that algorithms utilizing topology information are more efficient than the algorithms that are oblivious of that information. For the largest networks (with the distance threshold equaling 21), random merging gives only 15.3% of network economy. Random merging with preliminary sort achieves 32.8% of network economy, which is just slightly better than the topology-based splitting without merging (32.6%). Applying topology-based splitting with random merging resulted in 57% of network economy for the largest network size.

Another observation is that all the algorithms perform better in the average network sizes (with the threshold value between 7 and 9), but slightly worse in the smaller scales (distance threshold less than 6). Obviously, smaller networks involve fewer clients served (Figure 4) and consequently there are fewer ways to merge them in groups. This explains why all the methods are unable to give the maximum network economy in the smallest networks.

For the topologies with the threshold values larger than 9, a slight decrease in performance is observed due to the fact that for the fixed number of clients in larger topologies there are fewer common links in source-to-client paths.

Figure 5 shows how different methods of destinations grouping affect the false-positive rate. Plain topology-based splitting results in the minimal false-positive rate (0.5% in average). Topology-based splitting with random greedy merging shows an average false-positive rate of about 2%. As expected, topology oblivious methods result in the larger false-positive rates. In cases where far situated clients are grouped together and total number of nodes and links involved in the forwarding is large, more iBF check operations are performed which increases the probability of false positives.

The random merging algorithm results in about 10% of undesirable traffic. A preliminary sort with random merging reduces the false-positive rate to 8%. It is clear that smaller network scales result in less false positives.

Additionally, we calculate the average number of links covered by a single iBF for each algorithm. Figure 6 shows that topology-based splitting results in a very low density. This fact confirms that the method can be improved by additional merging of the resulting iBFs. For the 256-bit BF with 8 non-zero bits the optimal number of encoded links is 22. The other three algorithms almost reach this number.

We investigated performance of the proposed algorithms for different numbers of clients (from 10 to 10^6). The distance threshold was fixed at 21 so that the full network was used in this experiment. In order to place more than 10^4 clients within the network, we inserted more nodes into the network so that the total number of added end-hosts in the largest experiment was $2,000,000$. Additionally, we present the $5th$ and $95th$ percentiles for the network economy in Table 2.

All other methods achieve better network economy for a larger number of clients except the topology-based tree splitting. This method works better in middle-sized networks, while we observe the dramatic decrease in the performance in the largest topology with 1 million clients. This is due to the fact that in this case many clients have a shared parent; these parents become invalid and many active nodes are clients themselves. This results in the situation where many single clients are encoded by individual iBFs, and the decrease in the iBF density confirms this assumption.

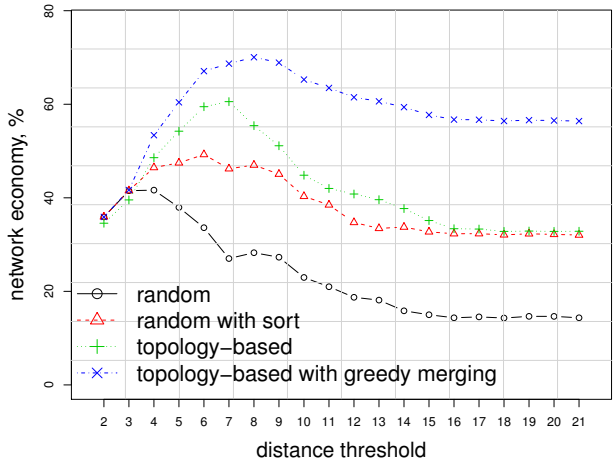

Figure 3: Average network economy for proposed algorithms with different distance threshold.

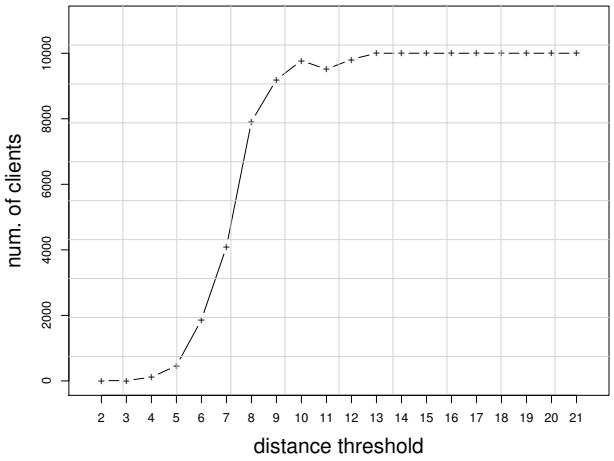

Figure 4: Average number of clients for each distance threshold.

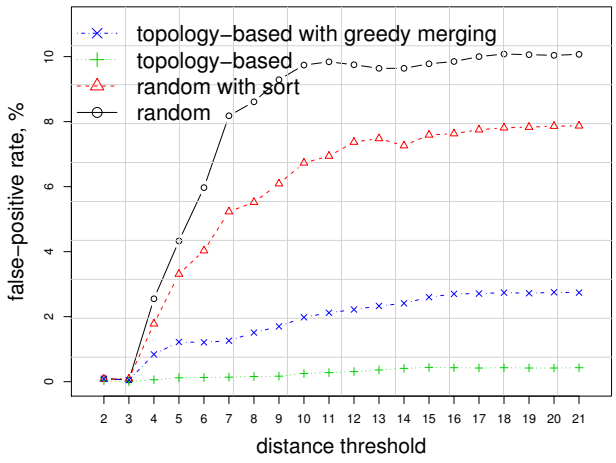

Figure 5: Average false-positive rate for each distance threshold.

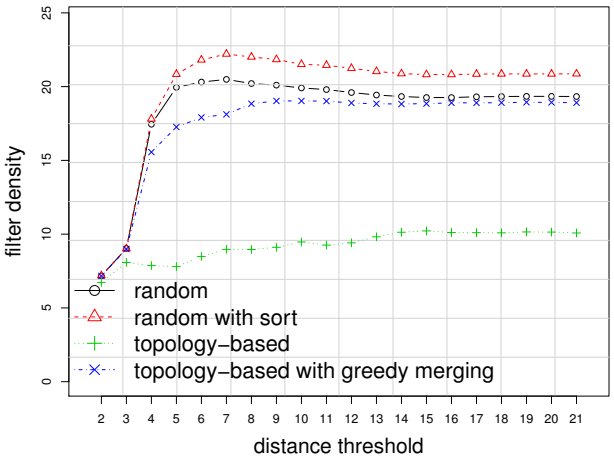

Figure 6: Average iBF density for each distance threshold.

Table 2: Evaluation of the proposed four algorithms for different numbers of clients.

#	Algo.		isBF			Economy (%)			over.
	topol.	extra	#	dens.	fpa(%)	5_{th}	mean	95_{th}	(%)
10	-	-	3.5	17.8	10.4	-2.5	9.4	19.5	34.2
	-	sort	3.4	17.9	10.5	0.0	10.1	18.2	33.2
	+	-	6.5	10.1	1.5	4.6	14.8	23.5	26.1
	+	merge	3.3	17.2	7.9	7.2	17.7	27.3	21.8
100	-	-	31.9	19.1	11.0	6.9	11.3	14.3	75.0
	-	sort	28.9	19.6	10.6	13.5	17.0	21.2	63.8
	+	-	69.6	9.3	1.0	12.5	16.9	20.7	64.0
	+	merge	26.6	18.3	6.2	29.0	31.5	34.1	35.0
1K	-	-	319.6	19.2	11.2	9.8	11.9	15.9	121.2
	-	sort	265.8	20.3	10.0	20.5	22.6	27.0	94.4
	+	-	630.1	9.5	0.6	22.1	24.5	26.5	89.7
	+	merge	224.0	18.6	4.4	42.4	43.6	46.0	41.7
10K	-	-	3402.5	19.3	10.1	10.3	14.3	16.0	198.7
	-	sort	2506.1	20.9	7.9	29.6	32.0	33.1	137.1
	+	-	5753.3	10.1	0.4	32.2	32.8	33.7	134.5
	+	merge	1879.7	18.9	2.7	54.6	56.4	57.2	52.1
100K	-	-	33501.8	19.3	11.3	15.2	15.3	15.4	382.2
	-	sort	20938.3	21.4	7.3	41.7	41.9	42.0	231.0
	+	-	36071.5	10.8	0.3	55.2	55.4	55.5	154.1
	+	merge	12241.3	19.1	1.7	71.7	71.8	71.8	60.8
1M	-	-	291939.1	17.5	10.2	16.3	14.7	16.4	520.4
	-	sort	160881.0	22.2	6.6	52.5	52.5	52.5	285.2
	+	-	976066.4	8.7	0.1	2.2	2.2	2.2	692.9
	+	merge	79679.2	21.1	1.8	78.9	78.9	78.9	70.9

For all the methods network overhead increases with the number of clients. This is because the links closer to the source are included in the paths to more clients. Since each iBF can encode a limited number of clients, such links experience more overhead copies of multicast packets.

Our evaluation shows that topology-based tree splitting with merging achieves the best results and is suitable for large-scale networks with a large number of clients. When the topology data is unavailable, random merging with preliminary sort is a second best choice. Moreover, with an increasing number of clients this method performs better.

4.5. Dynamic Topology View

In previous sections we tested several tree splitting algorithms in the scenarios where all clients are static. If one client decides to leave the multicast group or a new client emerges, computations have to be restarted. We applied several modifications to the proposed algorithms and conducted a set of additional experiments which confirm that it is possible to update corresponding iBFs quite quickly. Our implementation takes no more than 0.02 ms for processing joining or leaving client on consumer level hardware even for large scale tests. This enables processing of hundreds of thousands of clients per second. In extreme cases when this is not enough several servers have to be used in parallel and clients have to be distributed between them randomly or geographically. It comes at the cost of reduced traffic economy.

5. Economic considerations

The last but not the least problem, is the provider or domain authority incentives to utilize multicast. First of all, while the domains where the source of multicast is situated and core ASes benefit from multicast, the closest of higher level ISPs, which receives money for the total traffic may have disincentives to support multicast. We do not make any explicit statements about the pricing mechanisms among providers, but we notice that one way to overcome this limitation is to construct tunnelling for the isBF infrastructure. These tunnels are constructed through non-participating intermediate providers. Even with such disincentive providers, isBF domains may construct connected isBF topology and benefit from the multicast savings.

We propose the following economical variant of the multicast deployment. First, AS starts isBF in its own domain, using policy on BGP routers it removes any outgoing join packets outside of the domain. When

the AS finds another AS with the same multicast deployment, they may form a peering agreement, and allow join packets in both directions using BGP policy for directly connected routers or tunnelled connections.

Multicast traffic reduces only needs for end-user downstream traffic, while upstream traffic remains the same. isBF reduces downstream traffic of the clients (from Table 2) 3-4 times. The ASes will benefit from this approach. On the one hand, big multicast service providers (e.g., IPTV) will still need some CDN/RVZ server to reduce the outgoing traffic. The latter service is provided by ISPs with additional costs. On the other hand, small clients do not normally benefit from multicast traffic, and their payment model is based on downstream traffic rather then upstream. Therefore, the end-users can stay with the same bandwidth contracts of the ISPs, while the ISPs experience lower volumes of traffic and may provide additional service.

6. Conclusions

In this work we presented a novel multicast architecture (isBF), which by leveraging stateless network approach proposed previously for local domains, achieves high efficiency on a large scale. We presented 4 algorithms of destination. We measured performance of these algorithms on the Internet-wide topology acquired directly from the Internet traceroutes. Topology agnostic algorithms allow to reduce traffic up-to 2 times compared to unicast. Topology aware algorithms could reduce traffic almost 5 times in large networks, but are resource demanding for senders because part of the network topology with all clients has to be stored in memory. While the isBF loses out by a small fraction of overhead to the ideal stateful multicast, it has a potential to simplify the wide multicast deployment.

References

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

[2] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1:485–509, June 2004.

[3] S. Deering and D. Cheriton. Host groups: A multicast extension to the Internet Protocol. RFC 966, December 1985. Obsoleted by RFC 988.

[4] S. E. Deering and D. R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8:85–110, 1990.

[5] C. Diot, B. Neil, L. Bryan, and K. D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14:78–88, 2000.

[6] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[7] Z. Heszberger, J. Tapolcai, A. Gulyas, J. Biro, A. Zahemszky, and P.-H. Ho. Adaptive bloom filters for multicast addressing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 174–179, April 2011.

[8] Z. Jerzak and C. Fetzer. Bloom filter based routing for content-based publish/subscribe. In *Proceedings of DEBS '08*, pages 71–81, New York, NY, USA, 2008.

[9] P. Jokela, H. Mahkonen, C. Esteve Rothenberg, and J. Ott. (deployable) reduction of multicast state with in-packet bloom filters. In *IFIP Networking Conference, 2013*, pages 1–9, May 2013.

[10] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: line speed publish/subscribe inter-networking. *SIGCOMM Comput. Commun. Rev.*, 39:195–206, August 2009.

[11] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[12] E. Karpilovsky, L. Breslau, A. Gerber, and S. Sen. Multicast Redux: a First Look at Enterprise Multicast Traffic. In *WREN*, pages 55–64, 2009.

[13] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37:181–192, August 2007.

[14] D. Li, H. Cui, Y. Hu, Y. Xia, and X. Wang. Scalable data center multicast using multi-class bloom filter. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pages 266–275, Oct 2011.

[15] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews. In *IMC'05: Proceedings of the Internet Measurement Conference 2005 on Internet Measurement Conference*. USENIX Association, 2005.

[16] O'Sullivan. The Internet Multicast Backbone. `http://ntrg.cs.tcd.ie/undergrad/4ba2/multicast/bryan/index.html`. Last accessed Jan 2012.

[17] P. Paul and S. V. Raghavan. Survey of multicast routing algorithms and protocols. In *Proceedings of the 15th international conference on Computer communication*, ICCC '02, pages 902–926, Washington, DC, USA, 2002. International Council for Computer Communication.

[18] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. *SIGCOMM Comput. Commun. Rev.*, 36:15–26, August 2006.

[19] C. E. Rothenberg, C. A. B. Macapuna, M. F. Magalhes, F. L. Verdi, and A. Wiesmaier. In-packet bloom filters: Design and networking applications. *Computer Networks*, 55(6):1364 – 1378, 2011.

[20] M. Särelä, C. E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott. Forwarding Anomalies in Bloom Filter-based Multicast. In *IEEE INFOCOM'11*, 2011.

[21] T. H. Szymanski and D. Gilbert. Design of an IPTV Multicast System for Internet Backbone Networks. *Int. J. Digital Multimedia Broadcasting*, 2010.

[22] J. Tapolcai, A. Gulyas, Z. Heszberger, J. Biro, P. Babarczi, and D. Trossen. Stateless multi-stage dissemination of information: Source routing revisited. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2797–2802, Dec 2012.

[23] A. Zahemszky, P. Jokela, M. Sarela, S. Ruponen, J. Kempf, and P. Nikander. MPSS: Multiprotocol Stateless Switching. In *INFOCOM 2010, IEEE*, pages 1 – 6, May 2010.

Appendix A. Appendix

Algorithm 1 Greedy BF merging

Require: iBF_d, $1 \leq d \leq D$ — iBFs encoding paths to all destinations.

Begin
$cur \leftarrow iBF_1$
for $d = 2 \ldots D$ **do**
    $nxt \leftarrow cur \vee iBF_d$
    // $Ones(x)$ is a number of ones in x
    if $Ones(cur) \leq m/2$ **then**
        $cur \leftarrow nxt$
    else
        OUTPUT(cur)
        $cur \leftarrow iBF_d$
    end if
end for
OUTPUT(cur)
End

Algorithm 2 Topology-based tree splitting

Require: $H > 1$ — number of nodes; $V = \{v_i | 1 \leq i \leq H\}$ — nodes of data dissemination tree; $par(u), \forall u \in V$ — parents for all nodes in the tree; $iBF_d, \forall$ destination d — iBFs encoding paths to all destinations.

Input: $\forall i, j : v_i = par(v_j) \Rightarrow i < j$ (topological order)

Begin
$acc[1 \ldots H] \leftarrow$ empty BFs
for $\forall d$ — destination **do**
    $acc[d] \leftarrow iBF_d$
end for
for $i = H \ldots 2$ **do**
    $acc[par(i)] \leftarrow acc[par(i)] \vee acc[i]$
end for
for $i = H \ldots 2$ **do**
    if $Ones(acc[par(i)]) > m/2$ & $Ones(acc[i]) \leq m/2$ **then**
        OUTPUT($acc[i]$)
    end if
end for
End