Optimal False-Positive-Free Bloom Filter Design for Scalable Multicast Forwarding

János Tapolcai, *Member, IEEE*, József Bíró, *Member, IEEE*, Péter Babarczi, *Member, IEEE*, András Gulyás, Zalán Heszberger, *Member, IEEE*, and Dirk Trossen

***Abstract*—Large-scale information dissemination in multicast communications has been increasingly attracting attention, be it through uptake in new services or through recent research efforts. In these, the core issues are supporting increased forwarding speed, avoiding state in the forwarding elements, and scaling in terms of the multicast tree size. This paper addresses all these challenges—which are crucial for any scalable multicast scheme to be successful—by revisiting the idea of in-packet Bloom filters and source routing. As opposed to the traditional in-packet Bloom filter concept, we build our Bloom filter by enclosing limited information about the structure of the tree. Analytical investigation is conducted and approximation formulas are provided for optimal-length Bloom filters, in which we got rid of typical Bloom filter illnesses such as false-positive forwarding. These filters can be used in several multicast implementations, which are demonstrated through a prototype. Thorough simulations are conducted to demonstrate the scalability of the proposed Bloom filters compared to its counterparts.**

***Index Terms*—Bloom filter, information-centric networking, multicast addressing, source routing.**

Manuscript received January 08, 2014; revised July 14, 2014; accepted July 16, 2014; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Almeroth. This work was supported by the High Speed Networks Laboratory at the Budapest University of Technology and Economics. The work of J. Tapolcai and P. Babarczi was supported in part by the Hungarian Scientific Research Fund under OTKA Grant K108947. The work of J. Bíró was supported by the TAMOP-4.2.2.C-11/1/KONV-2012-0001 project funded by the European Union, cofinanced by the European Social Fund. The work of P. Babarczi was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences (MTA). The work of D. Trossen was supported by the FP7 PURSUIT project grant. This document has been produced with the financial assistance of the European Union under the FP7 GÉANT project under Grant Agreement No. 605243 as part of the MINERVA Open Call project.

J. Tapolcai and P. Babarczi are with the MTA-BME Future Internet Research Group, Budapest University of Technology and Economics (BME), 1117 Budapest, Hungary (e-mail: tapolcai@tmit.bme.hu; babarczi@tmit.bme.hu).

J. Bíró is with the MTA-BME Future Internet Research Group, Budapest University of Technology and Economics (BME), 1117 Budapest, Hungary, and also with the Inter-University Centre for Telecommunications and Informatics, 4028 Debrecen, Hungary (e-mail: biro@tmit.bme.hu).

A. Gulyás and Z. Heszberger are with the Information System Research Group, Hungarian Academy of Sciences (MTA), 1245 Budapest, Hungary, and also with the MTA-BME Future Internet Research Group, Budapest University of Technology and Economics (BME), 1117 Budapest, Hungary (e-mail: gulyas@tmit.bme.hu; heszberger@tmit.bme.hu).

D. Trossen was with the University of Cambridge, Cambridge CB3 0FD, U.K. He is now with InterDigital Europe, Ltd., London EC2A 3QR, U.K. (e-mail: dirk.trossen@interdigital.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2014.2342155

I. Introduction

In the current Internet, IP multicast is hardly deployed beyond single providers due to incentive, security, and scalability problems. These issues rendered multicast research a cold topic [1] in the last decades. However, recent trends in networking—such as information-centric networking (ICN), which is built on a publish/subscribe service model—urges the need for efficient multicast-based architectures. Common to these trends is that they are leaving the issue of *scalable forwarding* as a major challenge, which makes efficient multicast a core requirement for any successful solution in this space. Therefore, multicast routing research is in its second blossom [2]–[4]. The scalability challenge in these networks is driven by three factors, namely the following:

Goal 1) the need to scale the speed of the forwarding operation with the growing need for speed in the Internet;

Goal 2) the scalability in terms of state to be maintained in the routers for the forwarding operation;

Goal 3) the scalability in terms of supported sizes of multicast trees.

While solutions such as IP multicast generally support multicast trees that span across the Internet, its realization comes at the cost of lookup-based forwarding operations, hindering Goal 1. These operations require appropriately configured entries in forwarding tables within the individual network elements, which clearly hinders Goal 2. On the other hand, recent trends in networking, such as ICN or software defined networking (SDN), are deeply committed to the clear separation of the data and the control plane in order to achieve this goal [5]–[8].

Tackling Goal 2, an appealing approach to minimize, or even avoid, state in the network is that of source routing, which is therefore in its second blossom after decades of ignorance in the IP world. Here, instead of encoding next-hop information of the multicast tree at the intermediate nodes, as done in traditional approaches, efficiently encoding link information of the graph in a compact header allows for avoiding any state at these intermediary elements. Such design issues manifest in recent efforts toward using in-packet Bloom filters [2]–[4], [9] for encoding the edges of the multicast tree into the packet header. Bloom filters are originally designed for membership queries, i.e., for determining whether an element/edge belongs to a set/tree or not. Placed in packet headers, the *in-packet* Bloom filters can effectively address a set of nodes or links [2]. Conceptually, the LIPSIN method in [2] is closest to source-specific multicast (SSM, RFC3569) where the multicast traffic originates from a dedicated source node. However, this flat type of tree representation in LIPSIN (i.e., the structure of the whole tree is represented in an implicit manner as a set of edges) in a single fixed-size (256-bit) filter while keeping the number of *false-positive forwardings*—i.e., sending data on a link that is not involved in the multicast tree—low is very limited (about 20 links). Note that false-positive forwarding, e.g., in IPTV, has a severe affect as unnecessarily sending a high-quality video stream on a link highly degrades network performance. Thus, a single fixed-length header like LIPSIN is clearly not suitable to reach Goal 3.

In this paper, we show that Goal 3, i.e., the scalability problem in terms of multicast tree size, can be tackled by using some topology-related information when composing the in-packet Bloom filter, which was not present in the previous approaches. Although the excellent scalability of our solution would make it suitable for Internet-wide adoption, some technological concerns (e.g., simultaneous deployment of the required functionality at networking hardware) render an incremental domain-by-domain deployment scenario much more realistic. This promotes the thorough intradomain investigation of the problem, which will be made in this paper. First, we propose the so called *multistage Bloom filter* (MSBF) that consists of consecutive Bloom filters encoding only the membership of the edges residing at a given hop-distance (*stages*) from the source. Second, optimal (varying) length *false-positive-free* Bloom filters are introduced in order to eliminate false-positive forwardings, which is of utmost importance in any practical realization of in-packet Bloom-filter-based multicast forwarding. Our analytical investigations and simulations prove that our MSBF solution not only provides excellent space efficiency, but it also remedies several other anomalies besides false-positive forwarding arising when Bloom filters are at use, like forwarding loops and packet storms [4].

The rest of the paper is organized as follows. Section II expands on the related work in this area. In Section III, we define the architectural background in which we embed our solution and introduce the concept of our new multicast architecture. As the main contribution of this paper, the analytical results on creating minimal-length false-positive-free Bloom filters are presented in Section IV. In Section V, we discuss the implementation details of multistage Bloom filters in SDNs, IP routers, and an ICN prototype, followed by Section VI, where we present our simulation and measurement results on the proposed false-positive-free scalable multicast architecture. Finally, Section VII concludes the paper.

II. Related Work

A. Multicasting Routing

Standard IP multicasting can provide a way of transmitting packets from a source to many recipients in a bandwidth economical way. Multicast routing protocols like DVMRP [10], [11] and PIM-SM [12] for performing the transmission maintain multicast forwarding tables (MFTs) in the routers along the multicast trees. This stateful approach is not only in full contradiction of the stateless design of unicast IP, but is also the Achilles' heel of IP multicast. Due to the potentially high number of multicast groups in the network according to massive multiparty applications like video conferencing or networked games, the states to be maintained in a router could be untenable due to the unaggregability of the MFTs.

For better scalability in terms of Goal 1, explicit multicast solutions have been suggested in the literature, in which forwarding information (e.g., a list of destination IP addresses) on the targeted group is encoded in the packet headers. This flat type of stateless multicast routing protocol (like Xcast, Xcast+ [13]) undoubtedly has the drawback that every intermediate node along the multicast tree should process the header, even in the case the node is not a branching point of the tree. To avoid mandatory packet header processing, tree encoding schemes have been proposed to integrate into multicast protocols, like in ERM and Linkcast [14], [15]. In these methods, the entire tree structure is encoded into the headers. Moreover, in the latter case (and its successors [13]), forwarding at intermediate nodes is based on interpreting the tree code, and unicast lookups are completely eliminated. Nevertheless, the tree encoding scheme in Linkcast uses fixed-length link codes based on an assumption of the maximum size of multicast groups and imposes large computational overhead when interpreting the encoded tree.

B. In-Packet Bloom Filters

The *Bloom filter* [16] is a simple yet efficient data structure to answer membership queries. Let $\mathbb{S}$ be a set of elements, assigned with b-bit-long binary codes, in which a maximum of k bits are set in positions indicated by k different *hash functions*. Each of the k hash functions maps the given element onto one of the b-bit positions. The hash functions are assumed to be independent and each position is selected with equal probability. The Bloom filter representing $\mathbb{T} \subseteq \mathbb{S}$ is a b-bit-long binary array consisting of the bitwise OR of the codes of the elements in $\mathbb{T}$. As the main feature, a membership test can simply be performed by checking if all bit positions that are set in the code of the underlying element s are also set in the filter. The performance of the filter is measured by the false-positive rate that is the probability that an element s appears to be included (by test) but actually it has never been added $(s \in \mathbb{S} \setminus \mathbb{T})$.

The application of Bloom filters is becoming increasingly popular in future Internet architectures [2], [3] for tackling Goals 1 and 2. Placed in packet headers, the *in-packet Bloom filters* can effectively address a set of nodes or links, hereby qualifying themselves as a strong candidate solution for efficient stateless multicast addressing. When a packet with in-packet Bloom filter arrives to a router, membership testing is performed on the outgoing link identifiers. The bitwise AND and compare (CMP) operation on the Bloom filter placed in the header and on the address of the outgoing link is extremely fast when implemented on a digital signal processor, which leads to a simpler router architecture compared to current IP routers. Realizations of this solution, e.g., in [7], demonstrate that 1-GB/s throughput can be easily achieved in prototypes. Moreover, *Bloom filters are favored for their space efficiency* since the filter requires much less space than listing the identifiers for each link/host in the multicast tree. Furthermore, this allows routers to stay quasi-stateless because the routers only need to know the address of the neighboring nodes and links.

A clear weakness of such an in-packet Bloom filter concept is the total ignorance of the following:

(i) the topology information;

(ii) the tree structure in addressing.

In [4], the first issue (i) was touched by generating the Bloom filter addresses of each link according to the number of adjacent links, which is intuitively beneficial because the links with fewer adjacent links have a smaller chance to give false-positive. However, it is still not able to fully support Goal 3, which is addressed in this paper.

C. *Our Contribution*

In this paper, we propose an addressing scheme for arbitrary-size trees and multicast groups that lies somewhat between the two extreme approaches of explicit routing, i.e., the completely flat solution and the complete tree encoding-based ones, and suits well for the requirements of ICNs by tackling Goal 3. In our solution, link identifiers at equal hop-distances from the source of the multicast tree (referred to as *stages* hereafter) are to be confined into Bloom filters, but without any exact encoding of the tree topology, hereby providing an appealing mixture of space and forwarding efficiency.

Furthermore, we address both weaknesses (i) and (ii) of in-packet Bloom filters by introducing: (i) optimal-length varying-size (ii) multistage Bloom filters, as opposed to the traditional (single-stage) fixed-size in-packet Bloom filter concept [2] (which represents the trees flatly as sets in a fixed-length header). The idea of multistage Bloom filters was first introduced in [17], followed by an ICN prototype implementation in [18]. However, these works contained no clear guidelines for the proper dimensioning of the filters from the theoretical side, nor any tool for processing the required parameters (e.g., the number of hash functions) in an implementation. Thus, in this paper we also introduce the concept of false-positive-free Bloom filters and provide approximation and analytical results for filters with minimal length, which can be directly used in practice. The effectiveness and applicability of the derived approximation formulas are demonstrated through simulations and in our ICN prototype implementation as well.

III. Architectural Background

For the reasons discussed earlier assuming a domain-by-domain introduction of our MSBF approach, in the rest of the paper we assume that information is disseminated across a single domain. As *link identifiers are used instead of endpoint addresses*, from an overall network design perspective, an important consideration is the scheme being used for encoding the link information. Usually, the usage of fixed-size identifiers is favored, enabling line-speed execution of the forwarding operation (see the 256-bit-long identifier in [2] for an example). Such fixed size, however, limits the ability to appropriately encode any given size tree within the limited size of the identifier. While varying-size header approaches have been clearly at a disadvantage compared to fixed-size approaches, it is our contribution in this paper to provide a solution that closes this gap in possible forwarding speed while providing the previously outlined design advantages. For this, we contrast our work against existing source routing approaches and provide insight into the performance and implementation issues of our scheme.

Crucial to our approach is the knowledge of the overall topology over which the multicast tree is formed. Such topology knowledge is not unreasonable to assume, as can be seen in technologies like multiprotocol label switching (MPLS) or software defined networks as well as in proposals envisioning future information-centric solutions. Common here is the existence of a topology manager (or network controller) with knowledge of link information between forwarding elements under the management of this entity. It is the role of this topology manager to compute an appropriate forwarding header (MSBF in our multicast architecture) that can be used within its topology. We argue that the existence of such topology management in today's networks points toward the possibility to improve information delivery in existing IP networks as well.

A. *Architectural Concept of Information-Centric Networks*

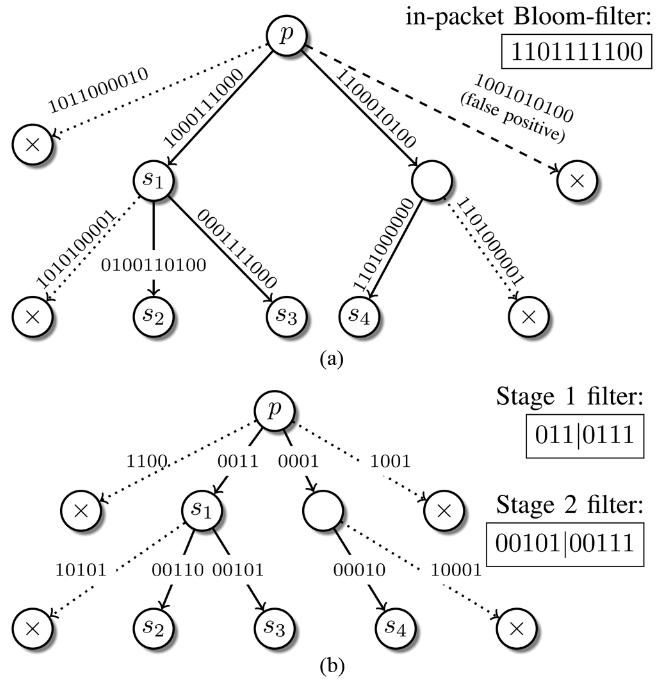
The architectural context of information-centric networks is based on the work in [7] and [19]. Here, we summarize the main design principles onto which we set the foundations for our proposed addressing scheme:

Spatially and Temporally Decouple Communicating Parties: A publish/subscribe service model is exported to all applications and ancillary network components. Hence, the producer of information (publisher) does not need to coexist in time with the consumers (subscribers), i.e., a subscriber can receive information even if the initial information producer is not online, since the information can be replicated or cached throughout the network. Moreover, communicating entities do not know the location of each other, spatially decoupling them.

Clearly Separate Network Functions: As discussed in [19], the core network functions are cleanly separated with each node supporting three network functions. The first one, *rendezvous*, matches demand for and supply of information. This results in some form of (location) information that is used for binding the provisioning of information to a network location. This information is used by the second function, *topology management and formation*, to determine a suitable multicast tree for the transfer of the information, this transfer being executed by the third function, *forwarding*.

B. *Traditional Bloom-Filter-Based Forwarding (1SBF)*

Based on the previous discussion on the ICN architecture, similarly to previous works on Bloom filters [2]–[4], [9], the existence of the *rendezvous* point is assumed in the network. However, it is important to note that this function can be easily implemented in a distributed fashion, e.g., target nodes subscribe to source nodes directly. Similarly to other source routing approaches, we assume that the up-to-date topology of the network is available at the *topology manager*. It is also the task of this topology manager to compute the multicast trees (or DAGs) for each multicast group after consulting with the rendezvous point.



(a)

(b)

Fig. 1. Traditional and multistage in-packet Bloom filters at the forwarding phase (AND and CMP at each interface) on a multicast tree with four subscribers s_1, s_2, s_3, and s_4. (a) Traditional (fixed-length) in-packet Bloom filter with false-positive forwarding, owing to the large number of $\alpha = 5$ in-tree and $\beta = 4$ out-tree links. (b) Multistage (varying-length) Bloom filter on the same topology without false-positive forwarding. At Stage 1, $\alpha_1 = 2$ and $\beta_1 = 2$, while at Stage 2, $\alpha_2 = 3$ and $\beta_2 = 2$.

Following the architecture of [2], the links are assigned by binary addresses, and the topology manager computes the multicast packet headers by combining the addresses of the links residing in the corresponding multicast tree. In order to better understand the details of the proposed multistage filter, let us first recall how this computation is done using the traditional in-packet Bloom filters.

- Each link is assigned by a binary link address (consisting of b bits of which at most k are set to 1).
- The topology manager computes the multicast header by bitwise OR-ing the addresses of the links in the corresponding multicast tree.

At each router, outgoing link addresses are tested against the Bloom filter (i.e., testing at each bit-position where the link address is set if the filter is also set with a binary AND and CMP function) and forwarded if positive. Note that this approach is totally topology-unaware since it simply traces back the forwarding decision to a classical membership testing problem. As an example in the multicast tree of Fig. 1(a), the in-packet Bloom filter contains the bitwise OR of five link addresses (with fixed length $b = 10$ and $k = 4$ hash functions). The filter is tested nine times (against every outgoing interface) during the forwarding process, producing false-positive forwarding even with such low number of links.

C. Varying-Length False-Positive-Free Multistage Bloom Filters

In-packet Bloom-filters are an efficient representation of the links of a multicast tree in an ICN context. However, it was shown in [2] that the multicast tree size that can be efficiently represented in a single fixed-size filter while keeping the false-positive rate low is very limited (about 20 links), contradicting with Goal 3.

Our contribution in this paper is to use a sequence of shorter so-called *stage filters* instead of a single large one. Each stage filter contains only the forwarding information for the edges with the same hop-distance from the source node in the multicast tree. Thus, we improve the completely flat addressing structure of the original in-packet Bloom filter concept, and some topology-related information is added to the packet header through the stage filters for tackling Goal 3. The efficiency of this approach will be demonstrated analytically in Section IV, while empirically in Section VI.

Another desired property of the introduced MSBF approach is that, when forwarding a packet, the *already-used stage filters can be truncated* at each hop one by one, as it contains only information for the already-traversed links. That is, a tree of h hops is represented by h stage Bloom filters, where the ith one contains only the links residing at i hop-distance from the source. When leaving the source, the multicast in-packet Bloom filter header consists of h stage filters, which then shrinks as the packet travels along the tree, highly reducing the overhead introduced by the in-packet Bloom filter.

In ICNs and also in traditional multicast applications, unnecessarily forwarding a high-bandwidth video stream on a link could result in degraded network performance owing to congestion. In terms of security and business considerations, it is also not desirable if a node receives a stream unintentionally. Thus, in order to keep the number of false-positive forwardings at a tolerable level in arbitrary-size multicast trees, applying *false-positive-free (FPF) varying-length stage filters* could be a solution, discussed in Section IV. For ensuring space efficiency, the length of the FPF filters is optimized at each stage as the function of the number of elements (i.e., links) it contains, which clearly results in varying-size stage filters. Thus, for identifying filter boundaries, we propose to store the length of each filter in the header, e.g., by applying the commonly used Elias gamma universal code [20], where the length γ of a b-bit-long filter is coded as follows (notations can be found in Table I):

- first $\lceil \log_2(b) \rceil - 1$ zero bits are written;
- followed by the binary representation of b on $\lceil \log_2(b) \rceil$ bits.

As it is shown in the example of Fig. 1(b), the packet header at Stage 1 consists of two parts, a $\gamma_1 =$ 3-bit-long Elias gamma code and a $b_1 =$ 4-bit-long Bloom filter. This filter at Stage 1 is tested against four links, out of which two are expected to be chosen. Similarly, at Stage 2, the second filter ($\gamma_2 = 5$, $b_2 = 5$) is tested five times with three expected positive outcomes.

The benefit of our MSBF approach with respect to the introduced overhead is clear, for example, if we investigate Fig. 1(b). Note that the average Bloom filter length is $b_{\text{avg}} = 10$ for 1SBF, while $b_{\text{avg}} = 6.6$ for MSBF. The average overhead bits for our MSBF approach is maximally $m_{\text{avg}} = 17$, while applying the header truncation mechanism it could be reduced to $m_{\text{avg}} = 12.8$ (as the number of overhead bits is 17 on the two links of Stage 1, while 10 for the three links of Stage 2). One can observe that even on this small multicast tree (where

TABLE I
NOTATION LIST

Notations	Description
$G = (V, E)$	the directed graph representation of the topology with nodes V and edge set E
$\mathbb{L}$	multicast tree
α	number of in-tree links ($= \lvert\mathbb{L}\rvert$)
β	number of out-tree links ($\leq \lvert E\backslash\mathbb{L}\rvert$)
b	Bloom filter length
γ	total length of the Elias-gamma function in the varying length in-packet Bloom filters
$m_l = \gamma_l + b_l$	total overhead bits at link (or stage) l
m_{avg}	average number of overhead bits per link in multicast tree $\mathbb{L}$
$\eta(\mathbb{L})$	filter compactness, m_{avg} divided by the number of links in the multicast tree
b_{avg}	average number of in-packet Bloom filter bits per link in multicast tree $\mathbb{L}$
$\lambda(\mathbb{L})$	Bloom filter compactness, b_{avg} divided by the number of links in the multicast tree
δ	test range where FPF Bloom filters are most probable in a practical implementation ($[E[L]-\delta, E[L]+\delta]$)
h	number of stages (maximal hop distance from source)
L	random variable representing the Bloom filter length
$\tilde{L}(\alpha, \beta)$	approximation of the expected FPF Bloom filter length
k^{opt}	optimal (real) number of hash functions
$k^*(\alpha, \beta)$	approximation (integer) number of hash functions
$\mathcal{D}$	set of multicast trees given in the network
η_{avg}	average filter compactness for all multicast trees $\mathcal{D}$ (with header abbreviation, if MSBFs are considered)
μ_{avg}	average filter compactness for all multicast trees $\mathcal{D}$ (without header abbreviation, if MSBFs are considered)
λ_{avg}	average Bloom filter compactness for all multicast trees $\mathcal{D}$

1SBF still performs well), the number of extra overhead bits paid for varying-length multistage filters is negligible (12.8 in contrast with 10 of the 1SBF approach), while with the application of the MSBF filter, false-positive forwarding was completely eliminated.

Note that the main idea behind eliminating false-positive forwarding is to move the uncertainty from the forwarding phase to the Bloom filter generation phase, i.e., generating an FPF filter can be considered as a randomized algorithm (with controllable uncertainties). However, once an appropriate filter is found, the forwarding will be surely FPF, i.e., the forwarding phase is deterministic. Thus, in order to characterize the properties of the FPF filter generation process, in Section IV we will derive and analyze the distribution and the expected value of FPF filter length in several settings and provide some practical guidelines that can be used in an implementation.

IV. Analysis and Design of FPF Bloom Filters

In the case of the original Bloom filter, elements of a certain subset $\mathbb{T}$ of a much larger (often unknown) set $\mathbb{S}$ can be collected (encoded) into the filter in such a way that membership queries can justify the inclusion of the elements of $\mathbb{T}$. Unfortunately, other elements from $\mathbb{T}\backslash\mathbb{S}$ can also happen to be tested positively, although they were never added. The frequency of the events of these false reportings of the membership test is called *false-positive probability*. The design goal is to generate a Bloom filter for $\mathbb{T}$ that can provide the false-positive probability under a prescribed value. After generation, the operation of such a filter can be erroneous. However, the false-positive rate can be kept under a small value.

As opposed to this, in multicast communication, both $\mathbb{T} = \mathbb{L}$ and $\mathbb{S} = E$ are known, and their cardinalities are quite comparable. This fact may be utilized in the filter generation process in such a way that after generating a filter for $\mathbb{L}$, all the elements of $E\backslash\mathbb{L}$ can be tested against the false inclusion. If the inclusion of one or more of the elements of $E\backslash\mathbb{L}$ is reported, then another filter can be generated for $\mathbb{L}$ and tested against false inclusion again. In principle, it is possible to find a filter with appropriate length that does not contain any of the elements of $E\backslash\mathbb{L}$. As opposed to the original Bloom filter, such a filter will be false-positive-free in operation and thus inherently suitable for error-free multicast forwarding.

The generation of such filters at the topology manager for any given multicast tree $\mathbb{L}$ is not a trivial issue and will be discussed in this section. Shortly foreshadowing this, the main tasks in an FPF filter generation can be summarized as follows:

- computing expected filter lengths based on the number of elements α to be included in and the number of elements β to be excluded from the filter (called *Scalability*, discussed in Sections IV-A and IV-B);
- computing/estimating the required test range δ based on the expected filter length (called *Insensitivity*, discussed in Section IV-C);
- testing the filter lengths in increasing order in the test range against the FPF property, stop when the filter tested is FPF using the appropriate number of hash functions (called *Implementability*, discussed in Section IV-D).

We will also demonstrate that breaking down the Bloom filter into stages with optimal length can significantly reduce the overall filter length b compared to the following:

- the traditional (single-stage) Bloom filter with optimal length;
- the LIPSIN approach using fixed-length in-packet Bloom filters.

The notations are summarized in Table I.

A. Expected Length of the False-Positive-Free Bloom Filter

Let α and β denote the number of in-tree (which are contained in the multicast tree $\mathbb{L}$, denoted by bold lines in Fig. 1) and out-tree (which are not contained in the multicast tree $E\backslash\mathbb{L}$, dotted lines in Fig. 1) links in a given stage, respectively, and we assume that there are h stages. For the sake of simplicity, we conduct our analysis for identical α and β values in each stage. However, in Section VI, general settings are used.

First, we recall how to design an optimally sized Bloom filter to meet a prescribed false-positive requirement [21]. Without loss of generality, the false-positive probability for a Bloom filter with length b can be calculated as

$$P_f = \left(1 - \left(1 - \frac{1}{b}\right)^{\alpha k}\right)^{k}. \tag{1}$$

Introducing the notation $p = (1 - (1/b))^{k\alpha}$ and minimizing P_f using the first derivative transforms to

$$\frac{\partial P_f}{\partial k} = (1-p)^{k-1} \cdot [(1-p) \cdot \ln(1-p) - p \ln p] = 0. \tag{2}$$

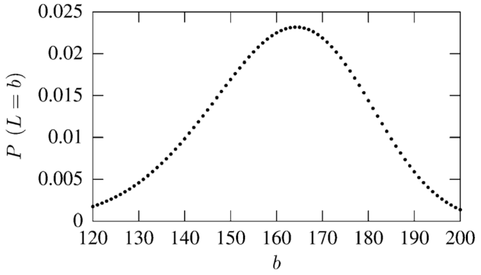


Fig. 2. Probability distribution of FPF filter lengths for $\alpha = 30$ and $\beta = 40$ ($E[L]^{\mathrm{MS}} = 161.2$).

It is easy to see that the optimal k, which is the optimal number of hash functions, is attained when $p = 0.5$

$$k^{\mathrm{opt}} = \frac{1}{\alpha}\frac{\ln(0.5)}{\ln\left(1-\frac{1}{b}\right)} \qquad P_f^{\mathrm{opt}} = 0.5^{\frac{\ln(0.5)}{\alpha\ln\left(1-\frac{1}{b}\right)}} \tag{3}$$

or using

$$\left(1-\frac{k}{b}\right)^{\alpha} \approx e^{-\frac{k\alpha}{b}} \tag{4}$$

we get the approximation

$$P_f^{\mathrm{opt}} \approx 0.5^{\frac{-b\ln 0.5}{\alpha}} = c^{\frac{b}{\alpha}}, \qquad \text{where } c = \frac{1}{2^{\ln 2}}. \tag{5}$$

In the following, the length of false-positive-free Bloom filters is discussed, i.e., a large-enough filter is created for which the expected number of false-positives will be zero. For this, let us define the following probability:

$$P(b,\alpha,\beta) = \left(1 - c^{\frac{b}{\alpha}}\right)^{\beta}. \tag{6}$$

This expresses the probability that in a b-length filter containing α elements, none of the β out-tree links are included in the filter. The probability distribution of the shortest FPF filter length L is shown in Fig. 2, which can be calculated as follows:

$$P(L=b) = P(b,\alpha,\beta)\prod_{i=1}^{b-1}\left(1 - P(i,\alpha,\beta)\right). \tag{7}$$

Now the length L of the false-positive-free Bloom filter comes from the random process of trying with increasing size filters and stopping when false-positive cannot be found. This expected length in case of the traditional (single stage) filter is

$$E[L]^{1\mathrm{S}} = \sum_{b=1}^{\infty} b \cdot P(b,h\alpha,h\beta)\prod_{i=1}^{b-1}\left(1 - P(i,h\alpha,h\beta)\right) \tag{8}$$

because there are $|\mathbb{L}| = h\alpha$ links to be included and $E\backslash\mathbb{L} = h\beta$ links are to be excluded in the filter. The expected length of the false-positive-free multistage filter is

$$E[L]^{\mathrm{MS}} = h\sum_{b=1}^{\infty} b \cdot P(b,\alpha,\beta)\prod_{i=1}^{b-1}\left(1 - P(i,\alpha,\beta)\right) \tag{9}$$

because there are h identical false-positive-free filters (each containing α links and not containing β links) confined in the stage filter. Using the above formulas, Table II shows a comparison between the expected length of the traditional and multistage Bloom filters, when $\alpha_i = 10$, $\beta_i = 30$, and the number of hops increases from 1 to 5. Evidently for a single hop ($h = 1$), (8) and (9) result in the same expected value 54.31, as the numbers of α and β links are the same. However, notice that the improvement due to using more stages can be stunning even if the number of stages is relatively small; for trees with larger depth, 30%–40% improvement can be reached.

TABLE II
COMPARISON OF TRADITIONAL (1S) AND MULTISTAGE (MS) BLOOM FILTERS IN EXPECTED FILTER LENGTH WHEN $\alpha_i = 10, \beta_i = 30, i = 1,\ldots,5$

No. of hops	$E[L]^{1S}$ from Eq. (8)	$E[L]^{MS}$ from Eq. (9)
1	54.31	54.31
2	127.94	108.62
3	210.01	162.93
4	297.69	217.24
5	389.61	271.55

Although the expected length of FPF Bloom filter cannot be used directly for filter generation (according to Fig. 2, the filter with length of the expected value will be false-positive-free with only about 2% probability[1]), it plays an important role in finding an appropriate minimal length for obtaining a false-positive-free filter, hence it is worth being analyzed. Based on this analysis, a test region of lengths (δ) around the expected value can be determined, in which the false-positive-free filter can be found with very high probability. For example, according to the minimal length distribution in Fig. 2, the minimal length of the FPF Bloom filter can be found in the region $[80,\ldots,240]$ with probability $1 - 10^{-5}$.

B. Approximations and Analysis of Minimal Filter Length

In this section, we provide analytical approximations for the probability distribution and the expected value of the minimal length of the false-positive-free Bloom filters. Based on these, we analyze how these quantities depend on α and β, and we also provide quantification on improvements in multistage filters against the single-stage FPF Bloom filter.

As previously presented in (6) and (7), the distribution of the filter length L can be characterized as

$$P(L=b) = \left(1 - c^{\frac{b}{\alpha}}\right)^{\beta}\prod_{i=0}^{b-1}\left(1 - \left(1 - c^{\frac{i}{\alpha}}\right)^{\beta}\right). \tag{10}$$

Based on this, the tail of the distribution [complementary cumulative distribution function (CCDF)] of the filter length L can be written as

$$P(L>b) = \prod_{i=0}^{b}\left(1 - \left(1 - c^{\frac{i}{\alpha}}\right)^{\beta}\right). \tag{11}$$

First, we show a useful approximation for this tail probability.

Lemma 1:

$$P(L>b) \approx \operatorname{Exp}\left(-\frac{\alpha \operatorname{Ei}\left(-c^{\frac{b}{\alpha}}\beta\right)}{\ln(c)}\right) \tag{12}$$

[1] This also means that the filter length generated by the FPF filter generation process may largely deviate from the expected FPF filter length.

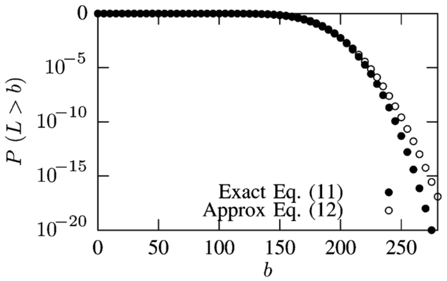


Fig. 3. Tail probability and its approximation for $\alpha = 30, \beta = 40$.

where $c = 1/2^{\ln 2}$ and $\mathrm{Ei}(z) = -\int_{-z}^{\infty}(e^{-t}/t)dt$ is the so-called exponential integral function.

The proof is relegated to the Appendix. Extensive numerical investigation showed that this tail probability approximation is accurate enough for practical-size problems $\alpha = (2, \dots, 100)$, $\beta = (10, \dots, 1000)$, and in the region $1, \dots, 10^{-10}$ of $P(L > b)$. In Fig. 3, one can observe good coincidence of the curves, and even in the range under 10^{-10}, the "horizontal"[2] relative difference between the curves is under a few percent.

In what follows, an accurate approximation is presented on the expected filter length $E[L]$ (either for $E[L]^{1\mathrm{S}}$ or $E[L]^{\mathrm{MS}}$). In principle, $E[L]$ can be expressed as the sum of the tail probabilities as $E[L] = \sum_{b=0}^{\infty} P(L > b)$. Unfortunately, it seems that it cannot be directly used as an integrate approximation sum because the integral $\int_{x=0}^{\infty} \mathrm{Exp}(-(\alpha \mathrm{Ei}(-c^{x/\alpha}\beta)/\ln(c)))\, \mathrm{d}x$ turned out being not characterizable by using elementary and/or special functions. Instead, we utilize the property of the CDF $P(L = b)$ that it is close to a symmetric distribution, therefore the median b^* is an acceptable approximation for the expected value, for which $P(L > b^*) = 1/2$. The next theorem presents the main result of Section IV-B

Theorem 1: The expected false-positive filter length can be approximated by the following formula:

$$E[L] \approx \frac{\alpha \ln\left(-\frac{\beta}{\mathrm{Ei}^{-1}\left(-\frac{\ln^3(2)}{\alpha}\right)}\right)}{\ln^2(2)} =: \tilde{L}(\alpha, \beta) \qquad (13)$$

where $\mathrm{Ei}^{-1}(.)$ is the inverse function of the left branch[3] of $\mathrm{Ei}(.)$.

The statement of the theorem immediately follows from Lemma 1 and the symbolic solution of the equation with respect to b

$$\mathrm{Exp}\left(-\frac{\alpha \mathrm{Ei}\left(-\left(\frac{1}{2^{\ln 2}}\right)^{\frac{b}{\alpha}}\beta\right)}{\ln^2(2)}\right) = \frac{1}{2}. \qquad (14)$$

The approximation $\tilde{L}(\alpha, \beta)$ is worth rephrasing as

$$\tilde{L}(\alpha, \beta) = \frac{\alpha}{\ln^2(2)}\left(\ln \beta - s(\alpha)\right) \qquad (15)$$

where $s(\alpha) = \ln(-\mathrm{Ei}^{-1}(-(\ln^3(2)/\alpha)))$ is a very slowly increasing function of α ($s(\alpha)$ is approximately proportional to $\ln(\ln \alpha)$). This mean filter length approximation is in very good agreement with the "exact" average length numerically computable by (9) or $\sum_{b=0}^{\infty} P(L > b)$. Two figures are presented for illustration: One is on the α dependence with fixed β, and the other is on the β dependence with fixed α (Figs. 4 and 5).[4]

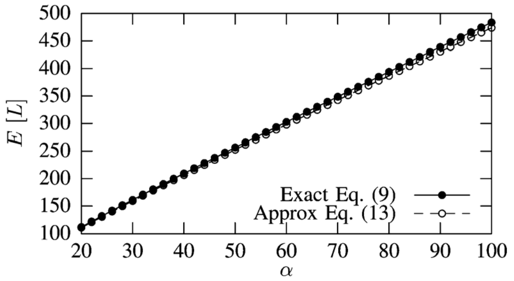


Fig. 4. Expected filter length and its approximation for $\alpha = (20, \dots, 100)$, $\beta = 40$.

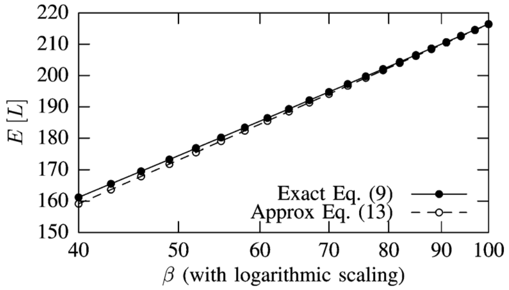


Fig. 5. Expected filter length and its approximation for $\alpha = 30$, $\beta = (40, \dots, 100)$ (log-linear plot).

Now, the main message of Theorem 1 is that the expected false-positive-free filter length is approximately proportional to the number of elements included (α), and to the *logarithm* of the number of elements excluded (β). This is referred to as the property of *Scalability*.

The formula of the expected filter length can also be used to quantify the expected gain in filter length using multistage filters against a single one-stage filter as

$$\tilde{L}\left(\sum_{i=1}^{h} \alpha_i, \sum_{i=1}^{h} \beta_i\right) - \sum_{i=1}^{h} L(\alpha_i, \beta_i). \qquad (16)$$

Taking the simple example of Section IV-A ($h = 5$ and for each stage $\alpha = 10$, $\beta = 30$), this formula gives 121.9 bits, while the previous numerical calculation (see the last row of Table II) provided 118.06 bits as expected gains.

C. Approximation of the Required Test Range

From a practical point of view, it is important to find the optimal-length FPF Bloom filter as quickly as we can. It is straightforward to search for a minimal-length FPF Bloom filter in $[1, E[L]^{MS}]$ with increasing filter length. However, it requires a lot of unnecessary computations, as from Fig. 2, one can observe that the most probable range to find an FPF filter is around the expected value $E[L]^{\mathrm{MS}}$. In this section, we are interested in the size of this *test range* δ, i.e., the test interval $[E[L]^{\mathrm{MS}} - \delta, E[L]^{\mathrm{MS}} + \delta]$, where it is practical to seek an FPF

[2] From a practical point of view, the "horizontal" relative difference has higher significance than the "vertical" one. That is, for given prescribed tail probability ε, the smallest b and $\tilde{b}$ for which $P(L > b) \leq \varepsilon$, $\tilde{P}(L > \tilde{b}) \leq \varepsilon$ are to be found and compared as a "horizontal" relative difference $(b/\tilde{b}) - 1$.

[3] $\mathrm{Ei}(.)$ has a branch cut discontinuity at 0; the left branch (with negative arguments) is a negative valued function.

[4] Note that in Fig. 5, the horizontal axis with β has logarithmic scale.

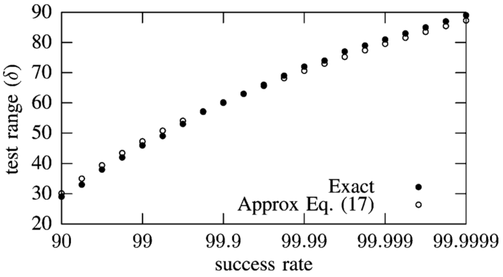


Fig. 6. Required test range and its approximation with success rate $10^{-1}, \ldots, 10^{-6}$.

filter, as a simple and accurate enough approximation can also be given for the required test range based on the CCDF (tail probability) approximation in Lemma 1. If the prescribed success rate for the test range is $1-\varepsilon$, then we find b_1 and b_2 such that the probability of the lower tail $(1-P(L>b_1))$ and the upper tail $(P(L>b_2))$ of the distribution should be at most $\varepsilon/2$. Using our tail probability approximation, the required test range is about

$$b_2 - b_1 \approx \frac{\alpha}{\ln^2(2)}\left(s\left(\alpha, \frac{\varepsilon}{2}\right) - s(\alpha, 1-\varepsilon/2)\right) \tag{17}$$

where $s(\alpha, x) = \ln(-\mathrm{Ei}^{-1}(\ln^2(2)\ln x/\alpha))$.

With $\alpha = 30$ and $\beta = 40$, the required test interval is 160 with 110^{-5} (99.999%) success ratio. The formula in (17) gives us 159. On Fig. 6, one can observe the small differences between (half of) the test interval (previously noted by δ) and its approximation versus the success rate, with the same α and β. Based on extensive numerical investigations, we observed that the analytical formula of the test range provides good approximation when $\beta > \alpha$ (i.e., the multicast tree includes less links than it excludes, which is true in most practical scenarios). Moreover, for these cases, the numerical computation reveals that the test range hardly depends on β. For example, if β is increased from 40 to 4000, in the previous example, the test range increases from 160 to 174. Note that this is also reflected by (17), in which the approximation of the test range $b_2 - b_1$ does not depend on β.

The main message of this section is that the required test range for finding the minimal false-positive-free filter length is approximately *independent* from the number of elements excluded. This is referred to as the property of *Insensitivity*.

D. Approximation of the Number of Hash Functions

Up until this point in the optimized Bloom filter and hence in the analysis of the false-positive-free Bloom filter, the number of hash functions is implicitly treated as a positive real-valued number based on (3). Nevertheless, in implementations, it should be a positive integer number. Recall that the optimal number of hash functions (in case of a b-length filter with α number of elements to be included) is

$$k^{\mathrm{opt}}(\alpha, b) = \ln(0.5)\frac{1}{\alpha \ln\left(1-\frac{1}{b}\right)} \approx \ln(2)\frac{b}{\alpha}. \tag{18}$$

With given parameters, this value should be rounded to a positive integer number. In finding the one with the minimal length among the false-positive-free Bloom filters, the rounding means that for every length to be tested, a rounded value of k^{opt} should be used. Based on thorough numerical assessment of the effect of this rounding procedure, we can state that all the numerically computable formulas and their analytical approximations for the required test range and the expected filter length change negligibly. Moreover, it can also be demonstrated that it is enough to use a *single* integer number of hash functions, and a suitable choice for this is

$$k^* = \mathrm{Round}\left(\ln 2 \frac{E[L]}{\alpha}\right) \approx \mathrm{Round}\left(\frac{1}{\ln 2}(\ln\beta - s(\alpha))\right) \tag{19}$$

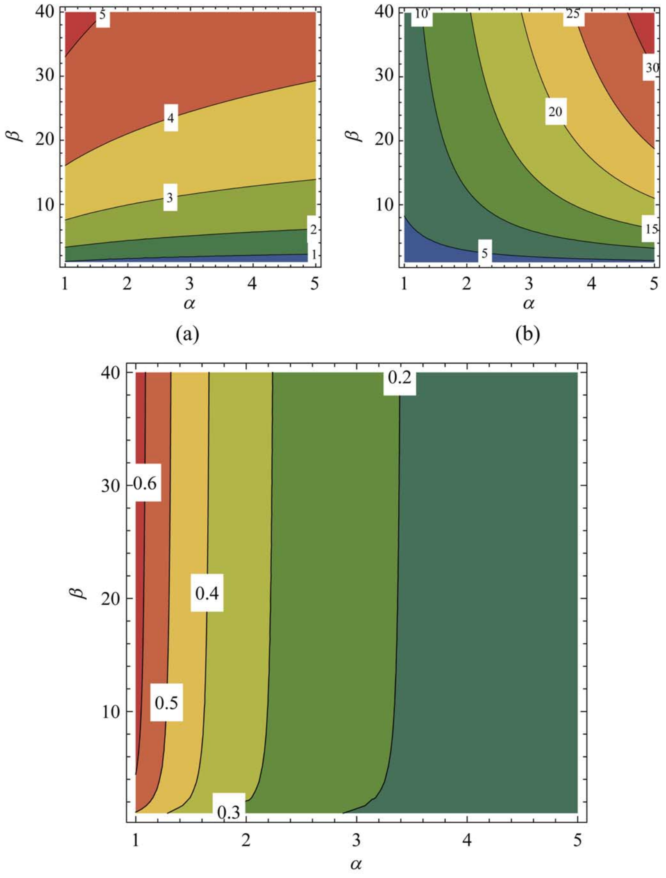
due to the approximation $\tilde{L}(\alpha, \beta)$ in (15). An illustrative example for the robustness of the filter length calculation against the rounding effect is as follows: The expected minimum filter length is 161.2 for $\alpha = 30$, $\beta = 40$ based on (9), and its approximation is 159.1 based on (15) (using real-valued numbers of hash functions). The rounded number of hash functions for this length is $k^* = 4$. Recalculating the expected filter length with this fixed number of hash functions is 162.4. Moreover, if we use 3 instead of 4 for the number of hash functions, the expected length is 162.1. Intuitively, this may be due to the fact that around the mean value there are the most probable minimal filter lengths having principle impact on the mean-value calculation, and in this region, the rounded k^{opt} may be constant or change its value only once. The very similar observations can also be performed for the required test range.

In Fig. 7(a), the contour plots of $k^*(\alpha, \beta)$ are presented on the α, β plane. It can be observed that for a given β, k^* remains intact through a wide range of α. It is also confirmed by the slow increase of the $s(\alpha)$ function. The contours represent the implicit function $\beta(\alpha)$ with fixed k^*, which is approximately logarithmic due to the observation that $s(\alpha) \sim \ln(\ln\alpha)$. In Fig. 7(b), the contour plot of the expected filter length $E[L]$ is presented on the same plane. It can be observed that for fixed expected length, β is quickly decreasing when α is increasing, that is, a filter with given length could exclude fewer elements if the number of elements to be included is larger. It is also justified by the $\tilde{L}(\alpha, \beta)$ approximation formula because, from this, β is approximately $\sim \mathrm{Exp}(-\alpha)$ if $\tilde{L}$ is fixed. In Fig. 7(c), the ratio of the number of hash functions k^* and the expected filter length $E[L]$ is plotted. From the previous figures and considerations, one may expect that this quantity does not significantly depend on β for a given α. This is acknowledged by the contours of the figure, which are almost "vertical."

The main message of this section is that the expected filter length and the required test range remain intact when the unimplementable real (continuous-valued) number of hash functions is replaced by a single integer number (therefore implementable) of hash functions. This is referred to as the property of *Implementability*.

V. Implementation Issues on Bloom-Filter-Based Forwarding

In Section III-C, an efficient multicast addressing mechanism in information-centric networks, called multistage Bloom filters, was introduced. The optimal-length MSBF header was calculated at the topology manager based on publisher/subscriber information of the rendezvous function using our results in Section IV. In this section, the implementation issues of



(a)

(b)

(c)

Fig. 7. Optimal number of hash functions. (a) Contour plot of $k^*(\alpha, \beta)$. (b) Contour plot of $E[L]$. (c) Contour plot of $k^*/E[L]$.

Bloom-filter-based forwarding are discussed in several architectures and environments.

A. Implementation Details in Software Defined Networks

In this section, we discuss the implementation details of single-stage Bloom-filter-based forwarding in OpenFlow. We note here that, to the best of our knowledge, our implementation [8] is the only one that is fully supported by standard OpenFlow v1.0 protocol.[5] Although the extension of this implementation for multistage filters cannot be done in a standard OpenFlow v1.0 conform manner, the basics of Bloom-filter-based forwarding can be easily demonstrated through it. Thus, we shortly summarize this approach.

TABLE III
FLOW ENTRIES IN THE iTH FLOW TABLE IN OUR OPENFLOW BLOOM-FILTER-BASED FORWARDING APPROACH

match condition	associated actions

(in-packet filter AND Bloom ID$_i$) CMP Bloom ID$_i$	output(port$_i$)
	goto-table(i+1)

(always match)	goto-table(i+1)

In the model of [8], Bloom IDs are assigned to switch ports. These Bloom IDs are used to generate the in-packet Bloom filter, i.e., port IDs, which the packet needs to pass through are bitwise OR-ed and put into the packet's destination Ethernet address field. In our implementation, we need to slightly deviate from the regular usage of flow entries. Normally, flow entries correspond to a specific set of *flows*, whereas we proactively configure a separate flow table for each *port* as shown in Table III. The first entry in the table matches if the packet's Bloom filter contains the port's Bloom ID. In case of a match, the switch forwards the packet via the corresponding port and, independently of the outcome, the packet is matched against the next port's flow table, hereby providing the stateless multicast switching capability. Hence, the packet is matched against the port ID of each outgoing interface and forwarded on all necessary ports.

B. Implementation Issues in Intermediary Forwarding Elements

Next, we discuss the implementation issues of the proposed MSBF framework [17]. The forwarding process of the multistage filter runs in each relay node along the multicast tree and consists of the following steps.

1) The header of the incoming multicast packet is loaded into a fast access type of memory.
2) The starting number of 0's is counted, and the *length* field is loaded accordingly (Elias-gamma function).
3) The following *length* number of bits is identified as the current stage filter and prepared for Bloom filter membership testing.
4) The Bloom filter membership testing is executed in parallel at each link interface card.
5) If the test is positive, then at the current bit position, the header is truncated—that is, the current stage filter with the length field is removed from the header.
6) If the header size is greater than zero, the packet is forwarded using the new header.

The membership testing requires a bitwise logical AND operation of the Bloom filter and the link address. The result is then tested (bitwise CMP) if it equals to the link address itself. Recall that the filters at different stages can have different sizes, which requires link addresses of varying sizes. To store the addresses of all possible lengths for each link can be overwhelmingly memory-consuming; prompt generation can provide a better solution instead. For this purpose, [22] proposes a lightweight mechanism, where two uniformly distributed random hash functions (even two can be enough) are used to generate varying-size hash codes to implement Bloom filters without any loss in the asymptotic false-positive probability. In this case, two hash functions $h_1(x)$ and $h_2(x)$ are stored at line cards, where x is the link address. The ith hash function $g_i(x)$ is generated by the formula $g_i(x) = h_1(x) + i \cdot h_2(x) \bmod b$, where b is the length of the hash code of the link to be established, and $i = 1, \ldots, k$. Finally, it is tested whether the $(g_i(x))$th bit in the stage filter is 1 for every i. Note that in such implementation, membership testing depends mainly on the number of hash functions and less on the size of Bloom filters, allowing fast forwarding even for large Bloom filters.

With respect to the processing power required by the proposed method, only $h_1(x)$ and $h_2(x)$ should be stored for each

[5] Our implementation is open-source and can be downloaded at http://sb.tmit.bme.hu/mediawiki/index.php/Sigcomm2012.

port, which can be stored in the local Level 1 (L1) cache of the CPU. L1 cache can be accessed in just a few CPU cycles, and its typical size is tens of kilobytes. Recall that in the proposed method, instead of address lookup, we need to decode the length of the first stage, performing 2–3 modulo divisions with remainder using the two hash functions of $h_1(x)$ and $h_2(x)$, and test the related bit-positions in the header. These operations should not require more than a few tens of CPU cycles,[6] which is not considered to be overwhelmingly expensive.

In comparison, e.g., forwarding an IPv4 packet requires 6–8 memory access operations (and up to 16 for IPv6) taking up most of the forwarding time. Note that a 3-GHz processor has a CPU cycle of 0.3 ns, while a single DDR SDRAM memory access operation requires ~20 ns (that is about 66 CPU cycles) for each memory access, summing up to 396–528 CPU cycles for the whole IPv4 address lookup strongly depending on the applied advanced cache technology [24]. In traditional label switching technologies, no extra processing is needed besides changing the label in the header. However, this comes for the price of keeping states in the switches (clearly not satisfying Goal 2).

C. Implementation Details in an ICN Prototype

We realize our solution in an available ICN prototype [7] for evaluation and demonstration. The *rendezvous* component implements the respective network function, as outlined in Section III-A. All publish/subscribe requests finally reach this element, which matches publishers with subscribers and triggers the formation of a forwarding path/tree. The *topology manager* manages the network topology and, upon request by a rendezvous component, creates forwarding paths from one or more publishers to one or more subscribers. These paths are sent to the respective publishers that use them when publishing information for a specific information identifier, utilizing the *forwarding* component for the delivery across the network.

We integrate our solution into the platform in [7] by adapting two modules [18]. First, we replace the forwarding component, which is currently based on [2], with a realization of the forwarding operation described in Section V-B. The platform design allows for forwarding functions to coexist, delimited by the dissemination strategy for a particular part of the information structure. Second, we also adapt the topology manager by extending the current minimal spanning tree mechanism with the header encoding, i.e., the topology management function splits the tree into stages and calculates stage Bloom filters by executing the following steps (initializing the filter length to $b := b_{\min}$).

Step 1) Create the stage filter BF as the bitwise OR of the first b bits of the α in-tree links (if $b \in [b_{\min}, b_{\max}]$ calculated in Section IV-C).

Step 2) Check whether BF contains any of the β out-tree link identifiers.

Step 3) If BF is false-positive-free, return BF, else go to Step 1 with $b := b + 1$.

Note that the test interval $[b_{\min}, b_{\max}]$ can be efficiently computed based on the *Scalability* and *Insensitivity* properties (presented in Sections IV-B and IV-C), that is given α and β, the expected filter length can be approximated by (15), and the required test range (given a prescribed success ratio $1 - \varepsilon$) around this mean value can be determined by (17).

Observe also that the three-step loop above succeeds to find the false-positive-free Bloom Filter in $[b_{\min}, b_{\max}]$ with probability $1 - \varepsilon$.

In the stage filter creation BF in Step 1, a single integer number of hash functions is used [according to (19)], and its applicability is credited with the *Implementability* property discussed in Section IV-D.

The length of the FPF Bloom filter (b) is included into the stage filter using an Elias-gamma encoding with $\gamma = 2 \cdot \lceil \log_2(b) \rceil + 1$ bits used for the length encoding per stage.

VI. Simulation and Measurement Results

A. Measures on Filter Length

In order to ensure the fair comparison of different Bloom-filter-based approaches, in this section we define performance measures to compare the header length of different forwarding schemes. Remember that the filter length m_l at link l in the proposed architecture stands for

$$m_l = \gamma_l + b_l \tag{20}$$

where γ_l is the total length of the Elias-gamma codes of the stage filters, while b_l is the total bits consumed by (stage) Bloom filters in the header at link l.

Definition 1: The *filter compactness* of a multicast tree $\mathbb{L}$ is denoted by $\eta(\mathbb{L})$, which is the sum of the header overhead along each link in the multicast tree divided by the square of the number of links $|\mathbb{L}|$ in the multicast tree, formally

$$\eta(\mathbb{L}) = \frac{\sum_{l \in \mathbb{L}} m_l}{|\mathbb{L}|^2} = \frac{m_{\text{avg}}}{|\mathbb{L}|}.$$

Obviously, a lower $\eta(\mathbb{L})$ refers to a more compact filter. To further explain, filter compactness refers to the average filter length divided by the number of tree links stored in the filter. Using this definition, the performance of an architecture does not depend on the tree size, and the average filter length can be obtained by $\eta(\mathbb{L}) \cdot |\mathbb{L}|$.

To compare the overall performance of different filters, in the simulations, a set of multicast trees is generated for a given network topology $G = (V, E)$, denoted by $\mathcal{D}$, and the *average filter compactness*

$$\eta_{\text{avg}} = \frac{\sum_{\mathbb{L} \in \mathcal{D}} \eta(\mathbb{L})}{|\mathcal{D}|}$$

of these trees is evaluated in the investigated architecture.

Recall that in our multicast architecture, the headers are truncated at forwarding (i.e., the route information already traversed

[6] Note that as $h_1(x), h_2(x) \leq b$, computing the remainder requires at most k subtractions and k comparisons in the worst case even without applying any software or hardware optimized algorithm. In the general case, a native and single (i.e., not considering pipeline execution) 32-bit unsigned integer division operation costs around 26–38 cycles in IA-32 or 64-bit Intel processor architectures [23].

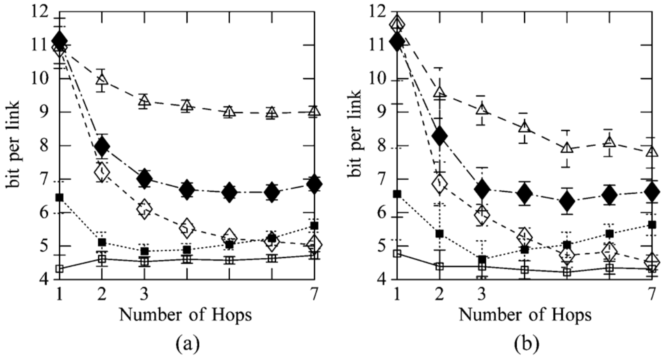


Fig. 8. Header length versus the number of hops in the multicast tree for the 37-node European reference network (legends are shown in Fig. 9). (a) Unicast demands. (b) Multicast demands.

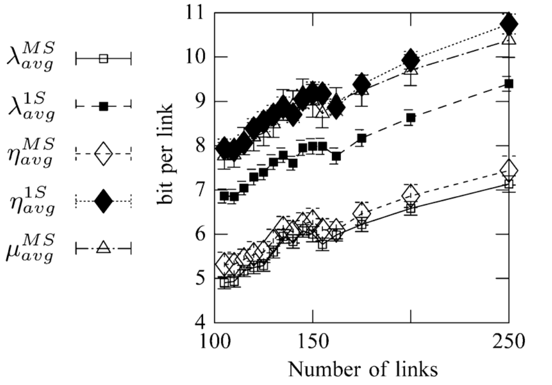


Fig. 9. Results of 2-connected 50-node networks with different number of links.

TABLE IV
IN-TREE AND OUT-TREE LINKS PER STAGE IN THE COST 266 PAN-EUROPEAN BACKBONE NETWORK WITH 37 NODES AND 57 LINKS ($\mathcal{D} = 2000$)

	Hop-distance (stage h) from source							1SBF
	1	2	3	4	5	6	7	Total
α	1.85	2.66	2.67	2.90	2.28	1.87	1.32	15.55
β	1.08	3.90	7.01	6.86	8.26	6.17	6.24	39.52

by the packet is erased), which leads to a better filter compactness η_{avg}. Therefore, as a reference, average filter compactness is also evaluated without abbreviating the headers at forwarding, denoted by μ_{avg}. In order to obtain a fair comparison of the traditional and our stage filters, similarly to the filter compactness, we define the following metric.

Definition 2: The *Bloom filter compactness* of a multicast tree $\mathbb{L}$ is denoted by $\lambda(\mathbb{L})$, which is the sum of the lengths of the in-packet Bloom-filters along each link in the multicast tree divided by the square of the number of links $|\mathbb{L}|$ in the multicast tree, formally

$$\lambda(\mathbb{L}) = \frac{\sum_{l \in \mathbb{L}} b_l}{|\mathbb{L}|^2} = \frac{b_{\text{avg}}}{|\mathbb{L}|}.$$

Similarly, *average Bloom filter compactness* without abbreviating the headers is defined as λ_{avg} for a set of $\mathcal{D}$ multicast trees.

B. False-Positive-Free Bloom Filter Length Analysis

In the simulation, we compare the two Bloom-filter-based forwarding approaches, namely the traditional in-packet Bloom filters [2] that were modified to handle varying-size headers (drawn with filled marks on the charts) and the proposed MSBF approach (drawn with empty marks on the charts). We compare their performance in terms of the metrics proposed in Section VI-A, i.e., η_{avg} and μ_{avg} to investigate the effect of abbreviating the header, and μ_{avg} and λ_{avg} to investigate the effect of varying-size headers.

Fig. 8 shows the result on the COST 266 pan-European backbone network with 37 nodes and 57 links. The demands were classified according to the maximum hop distance in the multicast tree. $\mathcal{D} = 2000$ random demands were generated with unicast traffic only in Fig. 8(a), and multicast traffic with up to 10 terminal nodes in Fig. 8(b). Coinciding with our analytical evaluation, the multistage Bloom filter has significantly shorter filter sizes. Surprisingly, this is true even when the multistage Bloom filter consists of one Bloom filter in one-hop trees. This is because, in multistage Bloom filters, the number of links on which false-positive can occur is smaller since none of the links, which are two hops away from the source node, can generate a false-positive due to the header truncation mechanism, i.e., zero-sized header prevents the forwarding of a packet in the multistage case. Note that the size of the multicast trees increases as the number of hops increases and shows the great scalability multistage Bloom filter can achieve. This is validated by the number of in-tree and out-tree links per stage as well in Table IV.

In the case of traditional in-packet Bloom filters, encoding the size adds small overhead, especially for large multicast trees. Since multistage Bloom filters consist of several consecutive Bloom filters, one may argue that encoding these boundaries may end up in a larger overhead. Our results support that this is not the case since, due to the truncation of the header at forwarding and the space efficiency of the multistage filter, in average header length the multistage Bloom filters remarkably outperform the original approach. The advantage of the MSBF approach is even more significant for unicast demands.

Next, we investigate the performance with respect to the network density. Twenty random 50-node, 2-connected networks are generated with different network density. Fig. 9 shows the results of $\mathcal{D} = 500$ demands where the horizontal axis corresponds to the number of links in the graph. One can observe that the performance gain of multistage Bloom filter is always 3–4 bits per link and does not depend on the density of the topology.

Finally, we investigate the scalability of the approaches as the network grows. Table V shows the results on topologies with different sizes and with multicast demands of terminal nodes at most 10. As a reference, an Xcast-based solution is also added to the table, in which the header consists of a series of IPv4 addresses with 32 bits for each destination. However, note that IPv4 and Xcast are not source routing solutions and require large routing tables at the routers (i.e., clearly not supporting Goals 1 and 2). We were surprised to see that, although the forwarding decision for an in-packet Bloom filter is significantly simpler and faster compared to traditional IPv4 forwarding, using in-packet Bloom filters even at the AS level has similar performance than Xcast. Further note that using the

TABLE V
TOPOLOGIES USED FOR THE NUMERICAL EVALUATIONS WITH FILTER LENGTH MEASURES FOR THE TRADITIONAL IN-PACKET BLOOM FILTERS (1SBF) AND FOR THE MULTISTAGE (MSBF) USING THE EXACT AND APPROXIMATION FORMULAS

			Xcast	η_{avg}			μ_{avg}	λ_{avg}		δ	
Topology	Nodes ($\|V\|$)	Edges ($\|E\|$)	IPv4	MSBF	Approx	1SBF	MSBF	MSBF	1SBF	Incremental	$E[L] \pm \delta$
Cost266 [25]	37	57	18.77	4.33	4.33	10.97	6.79	3.90	9.90	52.38	34.28
Germany [25]	50	88	17.88	5.97	5.97	11.79	8.14	5.13	10.42	66.69	40.15
Deltacom [26]	113	161	9.89	4.18	4.07	12.11	6.87	3.92	11.27	105.38	70.53
Random E-R	1001	1997	150.15	5.15	4.97	5.77	7.41	3.15	3.73	13.38	21.21
AS level	34306	71448	10.23	10.88	10.93	19.75	16.39	13.53	18.78	366.27	151.84

TABLE VI
HEADER LENGTH AND FORWARDING EXECUTION TIMES

	Header Length at Forwarding				Forwarding time	
Stage (h)	Elias (γ)	Filter (b)	MSBF ($m = \gamma + b$)	LIPSIN ($m = b$)	MSBF (μs)	LIPSIN (μs)
1	5	7	12	256	8.0	16.1
2	15	9	24	256	11.8	16.1
3	18	11	29	256	12.4	16.1
Total	38	27	65	256	32.3	48.3
1	2	3	5	256	4.9	16.1
2	21	11	32	256	12.9	16.1
3	31	11	42	256	14.2	16.1
4	18	11	29	256	12.4	16.1
Total	72	36	108	256	44.4	64.4
1	2	3	5	256	4.9	16.1
2	17	11	28	256	12.2	16.1
3	56	13	69	256	16.3	16.1
4	59	13	72	256	16.5	16.1
5	37	13	50	256	14.9	16.1
Total	171	53	224	256	64.8	80.5

approximation formula developed in Section IV-C for the test range δ requires a lower number of tested filter lengths in order to find an FPF Bloom filter, which is important from a practical point of view. Furthermore, these filters are comparable to those in η_{avg}, which were generated with the greedy approach.

C. Forwarding Complexity in the ICN Prototype

We compare the complexity of our scalable multicast architecture in each forwarding element with LIPSIN [2], based on the MSBF implementation within our ICN prototype [18]. Referring to Section V-B, the MSBF forwarding complexity not only includes the Bloom-filter-based membership test, but also the on-demand hash creation as well as the extraction of the stage FPF Bloom filter. On the other hand, the traditional Bloom filter approach (LIPSIN) only performs a constant-length Bloom filter membership test. For comparison, we created three different multicast trees on the same topology with three, four, and five stages, respectively. We repeatedly executed the forwarding function in our prototype (processing the LIPSIN header at each stage as well) and determined the average of the execution times at the various stage boundaries. Table VI shows the results of these tests, compared to the constant filter length and forwarding time of the LIPSIN. One can observe that the execution times for stage Bloom filters with length smaller than 50 are faster than comparable LIPSIN times. This is due to the membership test operating on smaller bit sets compared to the 256 bits used in the LIPSIN alternative. Again, these results can only be indicative since our current implementation is not optimized through, e.g., replacing the on-demand hash function through table lookups or utilizing hardware assistance could be further improve the performance of our MSBF approach based on varying-length FPF Bloom filters.

D. Discussion on Forwarding Loops and Packet Storms

A well-known problem of Bloom filters is that through a chain of false-positives, a packet can loop back to a previously visited node where it generates a false-positive again and is stuck in an infinite loop. In extreme situations, such a behavior may cause even packets storms. In [4], a bit permutation technique was proposed to prevent such anomalies with very high probability. Our proposed multistage Bloom filters remedy these illnesses in a fairly natural way (even without using FPF filters) since, due to the truncation of the filter after every stage, the packets cannot go further than a few hops. In such a way our MSBF approach certainly prevents the formation of infinite loops by encoding the stage decomposition into the header. Note that the flow duplication is indirectly prevented by limiting the multicast routing on tree without false-positive forwarding.

VII. CONCLUSION

In this paper, we addressed the scalable multicast forwarding problem and introduced a novel multistage Bloom-filter-based architecture, which tackles Goals 1–3. Although in ICN architectures and with the application of in-packet Bloom filters, Goals 1 and 2 have been addressed, the scalability in terms of supported multicast tree size was lost in these concepts owing to neglecting topology information when creating the in-packet filter. Furthermore, false-positive forwarding was a serious issue in these environments. We gave a thorough analysis on the design of minimal-length false-positive-free Bloom filter both for the traditional Bloom filter approach with varying-size filters and also for our novel multistage architecture in order to avoid such anomalies. Approximation formulas were proposed, which can be directly used in a practical implementation of, e.g., an ICN prototype. Our simulation results suggest that encoding topology-related information into varying-length multistage filters results in much better efficiency in several topologies than its previous counterparts.

APPENDIX

A. Proof of Lemma 1

First, we rewrite and approximate $(1 - c^{i/\alpha})^{\beta}$ as

$$\left(1 - c^{\frac{i}{\alpha}}\right)^{\beta} = \operatorname{Exp}\left(\beta \ln\left(1 - c^{\frac{i}{\alpha}}\right)\right) \approx \operatorname{Exp}\left(-\beta c^{\frac{i}{\alpha}}\right) \quad (21)$$

due to $\ln(1-x) \approx -x$ for small x. Based on this, consider the following approximation of the logarithm of $P(L > b)$:

$$\ln P(L > b) = \sum_{i=0}^{b} \ln\left(1 - \left(1 - c^{\frac{i}{\alpha}}\right)^{\beta}\right)$$

$$\approx \sum_{i=0}^{b} \ln\left(1 - \mathrm{Exp}\left(-\beta c^{\frac{i}{\alpha}}\right)\right) \approx \sum_{i=0}^{b} \left(-\mathrm{Exp}\left(-\beta c^{\frac{i}{\alpha}}\right)\right) \tag{22}$$

where the last approximation is again due to $\ln(1-x) \approx -x$. This sum above can also be viewed as an integrate approximation sum, hence

$$\sum_{i=0}^{b} \left(-\mathrm{Exp}\left(-\beta c^{\frac{i}{\alpha}}\right)\right) \approx \int_{x=0}^{b} \left(-\mathrm{Exp}\left(-\beta c^{\frac{x}{\alpha}}\right)\right) \mathrm{d}x. \tag{23}$$

The integrate on the right-hand side can be expressed by the exponential integral $\mathrm{Ei}(z)$ function as

$$\frac{\alpha \mathrm{Ei}\left(-c^{\frac{1}{\alpha}}\beta\right)}{\ln c} - \frac{\alpha \mathrm{Ei}\left(-c^{\frac{b}{\alpha}}\beta\right)}{\ln c} \approx -\frac{\alpha \mathrm{Ei}\left(-c^{\frac{b}{\alpha}}\beta\right)}{\ln c} \tag{24}$$

for larger b, from which the statement follows. ■

References

[1] J. Crowcroft, "Cold topics in networking," *Comput. Commun. Rev.*, vol. 38, no. 1, pp. 45–47, Jan. 2008.

[2] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," *Comput. Commun. Rev.*, vol. 39, no. 4, pp. 195–206, 2009.

[3] S. Ratnasamy, A. Ermolinskiy, and S. Shenker, "Revisiting IP multicast," *Comput. Commun. Rev.*, vol. 36, no. 4, p. 26, 2006.

[4] M. Särelä *et al.*, "Forwarding anomalies in Bloom filter-based multicast," in *Proc. IEEE INFOCOM*, Shanghai, China, 2011, pp. 2399–2407.

[5] V. Jacobson *et al.*, "Networking named content," in *Proc. ACM CoNEXT*, 2009, pp. 1–12.

[6] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," *Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, 2007.

[7] D. Trossen and G. Parisis, "Designing and realizing an information-centric internet," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 60–67, Jul. 2012.

[8] F. Németh, Á. Stipkovits, B. Sonkoly, and A. Gulyás, "Towards SmartFlow: Case studies on enhanced programmable forwarding in OpenFlow switches," *Comput. Commun. Rev.*, vol. 42, no. 4, pp. 85–86, 2012.

[9] C. Rothenberg, C. Macapuna, F. Verdi, M. Magalhães, and A. Zahemszky, "Data center networking with in-packet Bloom filters," in *Proc. Brazilian Symp. Comput. Netw. (SBRC)*, Gramado, Brazil, 2010, pp. 553–566.

[10] S. Deering and D. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *Trans. Comput. Syst.*, vol. 8, no. 2, pp. 85–110, 1990.

[11] S. Deering, C. Partridge, and D. Waitzman, "Distance vector multicast routing protocol," Internet Request For Comments RFC-1075, 1988.

[12] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol independent multicast-sparse mode (PIM-SM): Protocol specification (Revised)," Internet Engineering Task Force, RFC 4601 (Proposed Standard), Aug. 2006, updated by RFCs 5059 and 5796.

[13] M. Bag-Mohammadi and N. Yazdani, "A fast and efficient explicit multicast routing protocol," *IEICE Trans. Commun.*, vol. E-88B, no. 10, pp. 4000–4007, 2005.

[14] J. Bion, D. Farinacci, M. Shand, and A. Tweedly, "Explicit Route Multicast (ERM)," Internet Engineering Task Force, Jun. 2000.

[15] M. Bag-Mohammadi, S. Samadian-Barzoki, and N. Yazdani, "Linkcast: Fast and scalable multicast routing protocol," in *Proc. IFIP Netw.*, 2004, pp. 1282–1287.

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

[17] J. Tapolcai *et al.*, "Stateless multi-stage dissemination of information: Source routing revisited," in *Proc. IEEE GLOBECOM*, 2012, pp. 2797–2802.

[18] W. Yang, D. Trossen, and J. Tapolcai, "Scalable forwarding for information-centric networks," in *Proc. IEEE ICC-NGN*, Budapest, Hungary, 2013, pp. 3639–3644.

[19] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an information-centric internetworking architecture," *Comput. Commun. Rev.*, vol. 40, no. 2, pp. 26–33, Apr. 2010.

[20] P. Elias, "Universal codeword sets and representations of the integers," *IEEE Trans. Inf. Theory*, vol. IT-21, no. 2, pp. 194–203, Mar. 1975.

[21] P. Bose *et al.*, "On the false-positive rate of Bloom filters," *Inf. Process. Lett.*, vol. 108, no. 4, pp. 210–213, 2008.

[22] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better Bloom filter," in *Proc. ESA*, 2005, vol. 14, pp. 456–467.

[23] *"Intel 64 and IA-32 Architectures Optimization Reference Manual"* 2014 [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[24] T.-C. Chiueh and P. Pradhan, "High-performance IP routing table lookup using CPU caching," in *Proc. 18th. Annu. IEEE INFOCOM*, 1999, vol. 3, pp. 1421–1428.

[25] SNDlib, "Survivable fixed telecommunication network design library," 2006 [Online]. Available: http://sndlib.zib.de

[26] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," 2013 [Online]. Available: http://www.topology-zoo.org

János Tapolcai (M'09) received the M.Sc. degree in technical informatics and Ph.D. degree in computer science from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 2000 and 2005, respectively, and the D.Sc. degree in engineering science from the Hungarian Academy of Sciences (MTA), Budapest, Hungary, in 2013.

Currently, he is an Associate Professor with the High-Speed Networks Laboratory, Department of Telecommunications and Media Informatics, BME. He is an author of over 100 scientific publications. His research interests include applied mathematics, combinatorial optimization, optical networks and IP routing, addressing, and survivability.

Dr. Tapolcai is a TPC member of leading conferences such as IEEE INFOCOM (2012–2015). He is a winner of the MTA Lendület Program and Google Faculty Award and the Best Paper Award in ICC 2006 and DRCN 2011.

József Bíró (M'96) received the M.Sc. and Ph.D. degrees in electrical engineering and computer science from the Budapest University of Technology and Economics, Budapest, Hungary, in 1993 and 1998, respectively. He holds the Doctor of Sciences degree of the Hungarian Academy of Sciences, Budapest, Hungary.

Since 2009, he has been a Professor of network performance with the Department of Telecommunications and Media Informatics and a Senior Research Fellow with the Inter-University Centre for Telecommunications and Informatics. His recent research interests are stochastic modeling and performance evaluation of networks in general, hyperbolic random graphs and routing analysis in particular.

Péter Babarczi (M'11) received the M.Sc. and Ph.D. (*summa cum laude*) degrees in computer science from the Budapest University of Technology and Economics, Budapest, Hungary, in 2008 and 2012, respectively.

He is currently working as an Assistant Professor with the High-Speed Networks Laboratory, Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics. In 2012, he held an appointment as a Post-Doctoral Research Associate with the University of Waterloo, Waterloo, ON, Canada, and the University of Oklahoma, Norman, OK, USA. His research interests include all-optical failure localization, combinatorial optimization, dedicated protection, and network coding.

Dr. Babarczi won the János Bolyai Research Scholarship of the Hungarian Academy of Sciences in 2013.

András Gulyás received the M.Sc. and Ph.D. degrees in informatics from the Budapest University of Technology and Economics, Budapest, Hungary, in 2002 and 2008 respectively.

Currently, he is a Research Fellow with the Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics. His research interests are complex and self-organizing networks, network calculus, and software defined networking.

Zalán Heszberger (S'99–A'01–M'14)received the M.Sc. and Ph.D. degrees in electrical engineering from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 1997 and 2007, respectively.

Currently, he is an Associate Professor with the Department of Telecommunications and Media Informatics, BME. His main research interests are future Internet technologies and complex networking. Currently, he is working on clean-slate design Internet routing and network management algorithms.

Dirk Trossen received the Ph.D. degree in computer science from the Technical University of Aachen, Aachen, Germany, in 2000.

He is a Principal Engineer with InterDigital Europe, London, U.K., the European branch of InterDigital, Inc. His main responsibility lies in establishing the European presence of InterDigital through engagements within the EU-funded Horizon 2020 work program as well as within U.K.-funded efforts. He has more than 15 years of experience in network architectures, services, and wireless technology. His main contributions can be found in the area of interdomain networking as well as seamless handovers, physical network overlays, and new service concepts for operators. He was the technical lead of the European efforts PSIRP and PURSUIT, performing research on large-scale publish–subscribe information-centric systems in the context of the Future Internet and possible novel internetworking solutions. Prior to joining InterDigital, he was co-founder of TecVis LP, a U.K.-based software solution company in the mobile, context-aware solution space, and he held prior positions as a Senior Researcher with Cambridge University, Cambridge, U.K.; Chief Researcher with BT Research, Ipswich, U.K.; and a Principal Scientist with Nokia Research in Boston, MA, USA, and Helsinki, Finland. He is also a research affiliate with the Advanced Network Architecture Group, MIT CSAIL, Cambridge, MA, USA. He published more than 75 peer-reviewed papers in international conferences and journals and currently has 29 international patents.