University of Reading

Optimized hash for network path encoding with minimized false positives

Article

Accepted Version

Carrea, L. ORCID: https://orcid.org/0000-0002-3280-2767, Vernitski, A. and Reed, M. (2014) Optimized hash for network path encoding with minimized false positives. Computer Networks, 58. pp. 180-191. ISSN 1389-1286 doi: 10.1016/j.comnet.2013.09.002 Available at https://centaur.reading.ac.uk/51712/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

Published version at: http://www.sciencedirect.com/science/article/pii/S1389128613003046

To link to this article DOI: http://dx.doi.org/10.1016/j.comnet.2013.09.002

Publisher: Elsevier

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

CentAUR

University of Reading

Central Archive at the University of Reading

Reading's research outputs online

Optimized hash for network path encoding with minimized false positives

Laura Carrea[a], Alexei Vernitski[b], Martin Reed[a]

[a]*School of Computer Science and Electronic Engineering, University of Essex, UK*
[b]*Department of Mathematical Sciences, University of Essex, UK*

Abstract

The Bloom filter is a space efficient randomized data structure for representing a set and supporting membership queries. Bloom filters intrinsically allow false positives. However, the space savings they offer outweigh the disadvantage if the false positive rates are kept sufficiently low. Inspired by the recent application of the Bloom filter in a novel multicast forwarding fabric, this paper proposes a variant of the Bloom filter, the *optihash*. The optihash introduces an optimization for the false positive rate at the stage of Bloom filter formation using the same amount of space at the cost of slightly more processing than the classic Bloom filter. Often Bloom filters are used in situations where a fixed amount of space is a primary constraint. We present the optihash as a good alternative to Bloom filters since the amount of space is the same and the improvements in false positives can justify the additional processing. Specifically, we show via simulations and numerical analysis that using the optihash the false positives occurrences can be reduced and controlled at a cost of small additional processing. The simulations are carried out for in-packet forwarding. In this framework, the Bloom filter is used as a compact link/route identifier and it is placed in the packet header to encode

the route. At each node, the Bloom filter is queried for membership in order to make forwarding decisions. A false positive in the forwarding decision is translated into packets forwarded along an unintended outgoing link. By using the optihash, false positives can be reduced. The optimization processing is carried out in an entity termed the Topology Manger which is part of the control plane of the multicast forwarding fabric. This processing is only carried out on a per-session basis, not for every packet. The aim of this paper is to present the optihash and evaluate its false positive performances via simulations in order to measure the influence of different parameters on the false positive rate. The false positive rate for the optihash is then compared with the false positive probability of the classic Bloom filter.
Keywords: Bloom filter, random data structure, packet forwarding, information centric networks, multicast

1. Introduction

Bloom filters are very good data structures to represent concisely a set in order to support membership queries [1]. They are randomized data structures since they require the employment of hash functions for their construction (we will describe them in details in Sec. 2). Consequently, they have some probability of giving false positives when queried; that is, an element may appear to belong to the set when in fact it is not. Because of their succinct size, they have become very popular for network application. As Broder and Mitzenmacher have pointed out in [2], "there are many places in the network where one might like to keep or send a list, but the complete list would require too much space". In fact, Bloom filters offer a representation

which significantly reduces space at a cost of false positives. For many applications false positives can be tolerated or can be made low enough so that the space saving offered by the Bloom filter compensates for the probability of errors. Since Bloom filters have became very popular, having a broad range of applications, many variants of the Bloom filter have been proposed to optimize the data structure with regard to the false positive issue. A thorough survey on Bloom filter variants can be found in [3].

One of the most recent application of Bloom filter in networking is for inpacket forwarding [4], [5]. Contemporary packet forwarding techniques generally fall into two categories: either destination based forwarding utilizing switch or routing tables as in Ethernet or IP; or label swapping techniques as used by MPLS or ATM. However, with recent interests in alternative network architectures, such as content centric networking [6] or information centric networking [7], it is of interest to consider alternative forwarding strategies that might be more suitable for these new architectures or bring more general improvements. One such strategy is to use a Bloom filter [1] as a fixed header identifier that encodes a complete network path in an space efficient manner. While this could be used to replace a strategy such as MPLS labeling [8], it could also be used to implement a forwarding layer replacing IP in a clean-slate network architecture. False positives are a problem that is inherent to Bloom filters. In the case of network path encoding, a false positive means that traffic may pass over links that were not intended to be part of the path. Thus, false positives may cause bandwidth wastage or may generate a loop [9]. Although this may be a very rare event, it can cause major problems in the network. Hence, there is an interest in minimizing the false positive

occurrences or being able to control them for this application while keeping the length as small as possible, since the Bloom filter will occupy part of the packet header. One of the basic design variants of the Bloom filter is its length, a longer filter has generally lower false positives. Consequently, in a given application the length can be adjusted to suit a particular false positive rate. However, for this application the length of the Bloom filter is a critical parameter that can not easily increased.

In this paper, we present the optihash, a new variant of the Bloom filter. When compared to the Bloom filter, we show that it offers an optimization mechanism which reduces the false positives occurrences or alternatively reduces the data structure length for a given false positive rate. If false positive occurrences cannot be eliminated, the mechanism offers the possibility to choose which element would result in a false positive as would best suit a particular network application. The performance of the optihash is tested for the new forwarding mechanism which has been proposed within the framework of the project PSIRP (Publish/Subscribe Internet Routing Paradigm)[1]. This was a substantial effort, which aimed to re-design the whole Internet architecture above the physical layer. For this type of application, it may be useful to generally control the false positive occurrences, not necessarily to simply minimize them. The optihash offers this flexibility. However, the implementation of the optihash for in-packet forwarding is out of the scope of this paper.

The rest of the paper is organized as follows. In Sec. 2 we revisit Bloom

[1]http://www.psirp.org/ accessed in July 2013.

filters and we present their application for in-packet forwarding introduced under the aegis of the PSIRP project. In Sec. 3 we describe the optihash, the proposed new variant of the Bloom filter. Sec. 4 introduces the optihash for in-packet forwarding. In Sec. 5 the performances of the optihash are evaluated for in-packet forwarding in a regular network for different parameters and the false positive rates are experimentally estimated. Sec. 6 concludes the paper.

2. Bloom filters for the multicast forwarding fabric

Since the main construction suggested in this paper is inspired by the concept of the Bloom filter, this section reviews the Bloom filter concept and the multicast forwarding scheme based on the Bloom filter. Further details on Bloom filters can be found in the literature, for example in [2].

Bloom filters [1] are space-efficient probabilistic data structures for representing sets and supporting set-membership queries. They were introduced by Bloom [1] in 1970 to represent words in a dictionary. For some time, they have been mainly used in database applications [10], [11]. In the late 90s, Bloom filters started attracting the interest of the networking-research community, especially because of their simplicity and wide applicability in aggregating data sets providing low information processing and networking costs. Surveys on network applications of Bloom filters [2], [3] show employments of Bloom filters for overlay networks, data centric routing, traffic monitoring, caching, security, etc.

Recently, Bloom filters had an important role in the design of the forwarding fabric [4] of the information centric network introduced under the

aegis of the PSIRP/PURSUIT projects[2] [12]. In this scheme, Bloom filters encode links and routes between nodes in a source routing fashion. In this section, the Bloom filter is introduced and its use as the forwarding fabric in PSIRP is described.

2.1. Standard Bloom filters

Consider a subset $A = \{a_1, a_2, ..., a_n\}$ of n elements of a fixed set (called a universe) U of N elements. If it is desired to represent A, a naïve approach would be to represent each element as a fixed length binary identifier, a vector v, of length m and simply concatenate the representation into a binary identifier of length $|A|m$. Determining if an element $a \in A$ is in the identifier can be achieved by a linear search through the concatenated elements to see if they match. An example of this naïve approach in packet forwarding is IP strict source routing, where the IP addresses of the gateways are concatenated in an option header. For a large set this is relatively inefficient but distinct and complete. A Bloom filter is an alternative representation of the set A which encodes the elements into a single fixed-length identifier that is of length m and is thus more space efficient and, as shown below, allows faster testing of membership of an element in the set.

The Bloom filter representation encodes each element as a fixed length binary identifier with at most k bits set to one and the rest set to zero (where k is a parameter that is optimized for a particular application scenario). The bits set to one are chosen using k fixed independent uniform[3] hash

[2]http://www.psirp.org/ and http://www.fp7-pursuit.eu/PursuitWeb/ accessed in July 2013.

[3]Uniform means that the hashes are uniformly selected by the function within the set

functions, all with the same range $(1, \ldots, m)$. Let the set of hash functions be $H = \{h_1, h_2, \ldots, h_k\}$. To represent each separate element a of the set A in the Bloom filter, the bits of $\mathbf{v}$ at the positions $h_1(a)$, $h_2(a)$,..., $h_k(a)$ are set to one (thus, a particular bit of a Bloom filter can be set to one by several elements of the set). Those bits of $\mathbf{v}$ which are not set to one by any of the elements $a \in A$ remain equal to zero.

In order to perform a query to check if an element a belongs to the set A encoded by the Bloom filter $\mathbf{v}$, all the k bits at the positions $h_1(a)$, $h_2(a)$,..., $h_k(a)$ in $\mathbf{v}$ are examined. Obviously, if at least one of the bits is zero then certainly the element a does not belong to A. Now let us consider the situation when all the bits at the positions $h_1(a)$, $h_2(a)$,..., $h_k(a)$ are equal to one. In this case, it is not unreasonable to conclude that the element a belongs to A. However, it is possible that a does not belong to A despite all the relevant bits in the Bloom filter being one giving rise to a false positive occurrence. Summing up, we can say that when testing for membership of an element, a Bloom filter is free from false negatives, but may produce false positives. The probability of false positives is defined in the next section.

Mitzenmacher presents an analysis of the Bloom filter [13] and shows there are three fundamental performance metrics for Bloom Filters that can be traded off: computation time (corresponding to the number of hash functions k), size (corresponding to the array size m), and the probability of error (corresponding to the false positive rate f).

$\{1, \ldots, m\}$

2.2. False positive rate

The *false positive rate* is defined, in the context of a set A and the corresponding Bloom filter, as the number of elements of U giving rise to false positives divided by the number of elements of U not belonging to A.

Simple arguments give an approximate form of the probability of a false positive for an m-bit Bloom filter with k hash functions which encodes an n-element set as follows:

$$f_p(m,n,k) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - \mathrm{e}^{-\frac{kn}{m}}\right)^k. \tag{1}$$

This function has a minimum for k_{min} and its value is $f_{p,min}$:

$$k_{min} = \frac{m}{n}\ln 2 \quad f_{p,min} = \left(\frac{1}{2}\right)^{k_{min}}. \tag{2}$$

The derivation of the expression (1) of the false positive probability appeared first in Mullin [14]. Recently, it has been shown that (1) represents a strict lower bound on the real false positive probability for any $k \geq 2$ [15], [16] and the error can be large, especially for relatively small Bloom filters. A new expression has been proposed by Bose [15] and later by Christensen [16], but it is not in closed form and it is stable only for small Bloom filters.

The false positive probability in (1) is an *a priori* estimation of the false positive rate. Once the Bloom filter is formed, an *a posteriori* estimation of the false positive rate can be obtained as the proportion of ones set in the Bloom filter [16]:

$$f_{pa} = \rho^k, \tag{3}$$

where $\rho = \frac{s}{m}$ is the so-called filling factor of the Bloom filter, with s being the number of bits set to one.

The experimental estimation of the false positive rate is defined as

$$f_{pr} = \frac{p}{q}, \tag{4}$$

where q is the number of queries and p is the number of false positives observed in these queries.

2.3. Bloom filters for a novel forwarding scheme

Within the framework of the PSIRP project[4] (see for example [12]), a new forwarding scheme was introduced. The project aimed to re-invent the way of doing networking, re-defining the whole Internet architecture as far as the physical layer. The architecture is built around the concept of information and dissemination rather than point-to-point communication drives the networking. The primitives *publish* and *subscribe* rather than *send* and *receive* are considered as the primitives of the communication. Therefore, in this architecture stable end-to-end addresses are not needed. Information is the center of the architecture so that information needs to be labeled. End-to-end addresses are replaced by probabilistically unique identifiers of information. The way to reach nodes in the network is an information item as well which, of course, needs an identifier. These identifiers are called forwarding identifiers and they are used to forward packets in the network[5]. In

[4]http://www.psirp.org/ accessed in July 2013.

[5]In the architecture nodes, links and routes have different identifiers. The fact that IP addresses act as a routing locators (where a node is attached to a network) and identity label (who is the node) at the same time, seems to be the root of many Internet's limitations [17]. Unlike IP networks, in the architecture proposed by PSIRP the identity label and the routing locators are distinct.

particular, Bloom filters are used as an encoding to identify routes and links between nodes and they are the base of the forwarding layer of the PSIRP and later PURSUIT[6] architecture (see for example [7]).

Unlike all the network applications of Bloom filters where Bloom filters are stored at the network nodes [2], [4], this approach suggests including a Bloom filter into the packet header, which gives rise to a source-routing-like scheme. However, the choice of the Bloom filter for the packet forwarding scheme proposed in PSIRP offers an approach that combines flexibly elements of source routing (as default case) and stateful routing [4].

In order to be able to construct a forwarding identifier which encodes the route, instead of labeling the nodes for forwarding purposes, each point-to-point link is identified with a probabilistically unique identifier, called a Link-Id. In the nomenclature of the Bloom filter introduced earlier, each Link-Id is itself a Bloom filter. To each point-to-point link two Link-Ids, one for each direction, are assigned so that we have a directed graph. In the PSIRP/PURSUIT architecture the calculation of the forwarding Id is performed by a centralized entity called topology manager. As multicast delivery is possible we may term this *route* a delivery tree as a general term encompassing unicast and multicast. The topology manager is responsible for updating the sending nodes with a new forwarding Id if the topology changes. The PURSUIT project has demonstrated that this is a scalable solution for networks where the topology changes infrequently such as wired networks and proposes other solutions for handling mobility. For the remain-

[6]http://www.fp7-pursuit.eu/PursuitWeb/ accessed in July 2013.

der of the paper we will assume that this architecture is used and that we are operating in a reasonably static network with changes handled by the topology manager.

The Link-Ids are combined to form a Bloom filter encoding the route as a forwarding Id, FId, that is constructed by OR-ing all the Link-Ids $L_0, L_1, L_2, \ldots$ of all the link along the route:

$$\text{FId} = L_0 \vee L_1 \vee L_2 \vee \cdots \quad . \tag{5}$$

The forwarding table at each node is very small and it contains, in the default case of source routing approach, only the Link-Id of the links connected to that node as shown in Fig. 1.

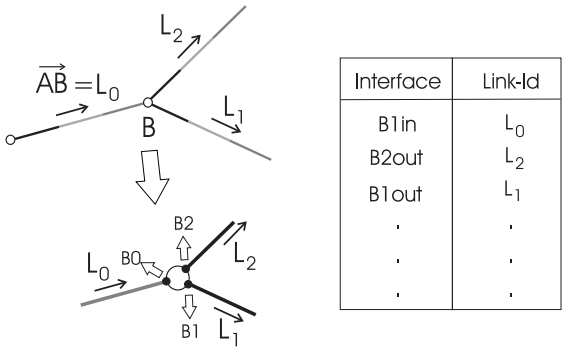

Figure 1: Example of a forwarding table in the PSIRP architecture.

As to querying the Bloom filter, once the packet reaches a node, the FId is *and*-ed with the Link-Ids of all the outgoing links of the node[7]. If the

[7]except the one from which the packet has just arrived, since forwarding the packet backwards does not make sense.

result of the *and* operation is equal to the Link-Id, as follows,

$$\mathrm{FId} \wedge \mathrm{L}_i \equiv \mathrm{L}_i, \tag{6}$$

then the packet is forwarded along that link. These operations can be easily parallelized since there are no memory or shared resource requirements and can be very easily implemented in hardware [4] and also in an optical node[8].

Of course, the match is possible along more than one outgoing link per node. In this way, multicast turns out to be the natural communication paradigm, while unicast becomes just a special case of multicast.

Due to Bloom filters' probabilistic nature, a query may result in a false positive, which translates, in our forwarding scheme, into extra traffic in the network. In other words, a false positive occurrence means that a packet will be forwarded along a link which is not part of the intended route.

False positives may not be significantly harmful for the network. First, the extra traffic in the network will be proportional to the number of packets using a particular Bloom filter. Given two routes with the same number of false positive occurrences, the route which is used for less intense traffic will be less harmful for the network in terms of bandwidth wastage. Second, packets delivered along wrong links could be opportunistically cached for future requests of nearby receivers. Third, primitives like publish/subscribe tolerate false positives, since non-requested content items have a limited life in the network and do not create forwarding states. Moreover, end-nodes

[8]In fact, it is possible to implement logical *and*s with optical components, allowing fast switching operations in full optical network without the need of the electrical node for making forwarding decisions [18].

are expected to effectively process only those pieces of information for which they have explicitly expressed their interest. Finally, packets forwarded due to false positives are highly unlikely to travel far in the wrong direction, due to the large label space and the exponentially decreasing probability of chained false positive forwarding decisions [19].

However, a control over the false positive occurrences is desirable, especially in case of dense multicast trees. Finding mechanisms to reduce false positives can eventually allow the use of a smaller packet header while keeping the false positive probability constant.

It is known that Bloom filters are not information-theoretically optimal [20], [2], [21], which means that a data structure exists which gives the same false positive rate but using less space. However, the Bloom filter is intrinsically simple. Since the simplicity of the Bloom filter approach is key for implementation, in this paper we aim to propose a method to improve the Bloom filter's performance regarding false positives, while trying to maintain its other desirable properties.

3. The optihash

The idea of the optihash is inspired by the recent application of Bloom filters in the novel multicast forwarding fabric of PSIRP (see Sec. 2.3) and it is built with the aim of providing a mechanism to control false positives while maintaining the properties of the Bloom filter. False positives for the forwarding application can generate additional traffic which is undesirable, hence there is a strong motivation to reduce, or control, the false positive occurrences in the network. For the forwarding application of the optihash,

as with the Bloom filter, we have a set of edges $\{e_1, e_2, \ldots\}$ which describes a path, or tree as described in Section 2. We require this set to be encoded in the optihash such that by using the optihash in the packet header, a forwarder can use the optihash to determine if a packet should be forwarded over an outgoing link e. For more detail on how the optihash is applied in the network forwarding application see Section 4, in this section we maintain a general approach as the optihash can have wider applications.

In order to be able to control false positives, one of the novel aspect of the optihash scheme is the introduction of a family of functions $f_{\alpha\beta}$ which allows an optimization process at the stage of its formation which involves a little more processing than the simple Bloom filter.

The optihash scheme can be employed for other applications since the mechanism is general enough not to be limited to the specific application. In this section we introduce the concept of the optihash independently on the particular application.

The optihash data structure we propose is a bit array $\mathbf{w}$ which consists of three parts: a bit array $\mathbf{v}$ and two integer parameters α and β. We can notice that in the implementation we propose (see the next section for details) two parameters are needed but the number of parameters can be different. Later we will show, in Section 4 and 5, how these values are used to construct a forwarding identifier which will be used in the packet header as described earlier for the Bloom filter in Section 2. We assume that $\mathbf{w}$ has a fixed length $m_{\mathbf{w}}$, split into $m_{\mathbf{v}}$ bits for storing $\mathbf{v}$ and two shorter fragments consisting of m_α and m_β bits[9], respectively, for storing the values of two integer parameters

[9]Generally we may have a different number of fragments dependently on the particular

α and β:

$$m_{\mathbf{w}} = m_{\mathbf{v}} + m_{\alpha} + m_{\beta}. \tag{7}$$

The values of α and β are, respectively, in the interval $(1, ..2^{m_\alpha})$ and $(1, ..2^{m_\beta})$.

In order to encode elements in the optihash, we initially propose to employ only one hash function h which produces an integer in the interval $(1, ..m_{\mathbf{v}})$; thus, this part of our construction is identical to using a Bloom filter with $k = 1$ and $m = m_{\mathbf{v}}$. The choice of using only one hash function is derived by the fact that the gradient of the false positive probability for the Bloom filter (1) for $k = 1$ is almost constant, while for $k > 1$ the gradient rapidly increases with the number of inserted elements, as shown in Fig. 2. Later we will show that the optimization with the functions $f_{\alpha\beta}$ does not change very much the gradient, but lowers the overall level. Thus, using $k = 1$ means that we are aiming to give significant improvement for a high number of inserted elements, which is the region that needs most attention.

To encode a set $\{e_1, e_2, \ldots\}$ of elements in a optihash, the bits in $\mathbf{v}$ at the positions $h(e_1), h(e_2), \ldots$ are set to one:

$$\mathbf{v}[h(e_i)] = 1. \tag{8}$$

In order to query for an element, that is, to check if an element e is in the set encoded by the optihash, the hash $h(e)$ is calculated and then if the position $h(e)$ contains one in the optihash $\mathbf{v}$ then we conclude that the element e, encoded in $\mathbf{v}_e$, belongs to the set encoded by $\mathbf{v}$. Sometimes it might happen that the element e was not encoded originally in the optihash;

implementation.

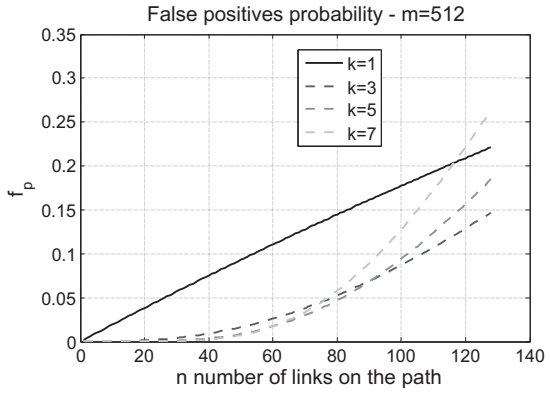

Figure 2: The false positive probability for the Bloom filter with $m = 512$, as a function of the number of inserted elements for different value of k. The curve for $k = 1$ has a almost a constant gradient.

then a false positive occurs. Otherwise, if the position $h(e)$ is filled with a 0, it is certain that the element e does not belong to the set.

The logical operations *and* and *comparison* can be used for testing if the element encoded by the Boolean array $\mathbf{v}_e$ belongs to the set:

$$\mathbf{v} \wedge \mathbf{v}_e = \begin{cases} = \mathbf{v}_e & \text{if } e \in E \\ \neq \mathbf{v}_e & \text{if } e \notin E \end{cases} \tag{9}$$

where $\mathbf{v}_e$ is a bit array of length $m_{\mathbf{v}}$ in which only the bit at the position $h(e)$ is set to one.

If E is a set of elements encoded in the optihash, the hash function h maps the elements in E to a generally smaller set R. The inverse h^{-1} maps the set of hashes R to a set of elements which is bigger than E. Namely, $h^{-1}(R)$ is the full set of elements that have hashes in R whereas E is only a subset, $E \subseteq h^{-1}(R)$. If Q is the set of elements which are queried, the set of elements F which give false positives can be defined as the intersection between the set of elements $h^{-1}(R)$ which have hashes in R and the set of

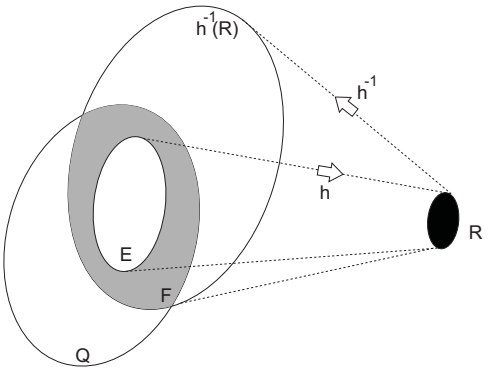

Figure 3: Given the set of hashes in $R = h(E)$ of the set of elements E in the optihash, the set of elements in $h^{-1}(R)$ which have hashes in R is generally a superset of E, namely $E \subseteq h^{-1}(R)$. Given Q, the set of elements to be queried, the set of the elements which produce false positives F is the intersection between Q and $h^{-1}(R)$ excluding the elements of $h^{-1}(R)$ which are already in the optihash.

elements Q which are queried minus the set of elements E used to build the set of hashes R:

$$F = \left(h^{-1}(R) \cap Q\right) \setminus E. \tag{10}$$

The sets are shown in Fig. 3. We define the set D as the set of elements to be queried which may generate false positives $D = Q \setminus E$.

So far, our exposition is similar to the Bloom filter construction with $k = 1$. However, our concept of the optihash includes an optimization specific to the particular set E of elements to be encoded and to the set D of elements to be queried.

Given the set E of elements to encode in the optihash and the set D of elements to be queried, the number of false positive occurrences is the set $h^{-1}(R)$ of elements which have hashes in R which are also in D (see Fig. 3). The optihash we propose involves calculating initially the set F of elements

which give false positive. If F is empty, there are no false positives in this instance, then the optihash is constructed with the hashes in R and with $\alpha = 0$, $\beta = 0$. If F is not empty, there are false positives in this instance, then we propose to use a family of functions

$$f_{\alpha\beta}[\lambda](\mu) \tag{11}$$

where μ is the hash of elements in E and D, λ is a hash that can be chosen according to the application and α and β are two parameters. We can notice that in the implementation we propose (see the next section for details) two parameters are needed but the number of parameters can be different. The output of the function f is an integer in the interval $(1, ..., m_{\mathbf{v}})$. For each value of α and β, the function f generates $2^{m_\alpha}2^{m_\beta}$ new sets of hashes for the elements in E and in D. In fact, the output of the function f is a transformed hash. In total we will obtain $2^{m_\alpha}2^{m_\beta}$ new sets of hashes for E and D. Then $2^{m_\alpha}2^{m_\beta}$ sets F are constructed evaluating the elements generating false positives. Among all the new sets F, the sets with the minimum number of elements is selected and the correspondent value of α and β is recorded.

The allocations α β that corresponds to a minimum number of false positives are considered and one pair is chosen. The pair is then used to recalculate the hashes of the elements to encode in E and of the elements to compare in D using the function f. Note that these hashes calculated with the selected values of $\alpha = a$ and $\beta = b$ are specific to the particular sets E and D. Since it is necessary to know the value of a and b for membership queries, the optihash encoding finally the set E is formed with:

- $\mathbf{v}$, the hash array built with the selected value of the pair $\alpha = a$ and $\beta = b$,

- a,
- b.

Since we are searching for the minimum, this scheme provides an optimization process which leads to a reduction of the false positive rate. However, it also provides the opportunity to choose which elements should offer false positives and also how many. The form of the family of functions $f_{\alpha\beta}$ is crucial for the optimization since it has to provide a big variety of different outputs. Actually, the properties of the functions $f_{\alpha\beta}$ do not differ much from the properties of hash functions. Hash functions have the properties of being deterministic, having the output which is uniformly distributed and of mapping a larger set to a smaller one. Regarding $f_{\alpha\beta}$, we require determinism since we want to apply $f_{\alpha\beta}$ more than once, uniformity since we want to map the inputs as evenly as possible over the output range to guarantee diversity. However, unlike hash functions $f_{\alpha\beta}$ maps a set of integers to itself.

We investigate the performance of the optihash scheme for a specific application: the packet forwarding in the PSIRP architecture described in the previous section.

4. The optihash for packet forwarding

The application by which the optihash has been inspired and for which the evaluation of the optihash performances is carried out is the packet forwarding scheme described in Sec. 2.3.

To illustrate the use of the optihash in the in-packet forwarding scenario, a network graph is considered, and initially a fixed hash value, $h(l_i)$, is assigned

to each link, l_i, in the network and the correspondent $\mathbf{v}_{l_i}$ is constructed as in (8).

A route is selected by the control entity of the network from a source (publisher) node to one or more destination (subscriber) nodes. In the PSIRP/PURSUIT architecture this operation is performed by a centralized entity called topology manager. As multicast delivery is possible we may term this *route* a delivery tree as a general term encompassing unicast and multicast. The topology manager is responsible for maintaining the network graph and for calculating the multicast tree when a publisher and subscriber are matched. Moreover it is responsible for generating the forwarding Is which is the optihash in this case. Consequently, the topology manager also carries out the optimization to create the optihash as described in Section 3. We should note, that as there is no state information required for the optihash calculation other than the knowledge of the network topology, it is possible to replicate the topology manager function for scalability if required.

Once a deliver tree is selected it is encoded in $\mathbf{v}$ *or*-ing the $\mathbf{v}_{l_i}$ of the composing links obtained using the hashes $h(l_i)$:

$$\mathbf{v} = \mathbf{v}_{l_1} \vee \mathbf{v}_{l_2} \vee .. \tag{12}$$

Then the number of false positive occurrences that arise along the specific route and the false positive rate (4) are evaluated *and*-ing and *comparing* the array $\mathbf{v}$ encoding the route and the arrays $\mathbf{v}_{l_j}$ of the off-route links l_j also constructed using the hashes $h(l_j)$. In the case the number of false positive occurrences is not zero, the optimization described in the previous section is performed within the topology manager function in order to construct the forwarding Id (see (5)). The array $\mathbf{v}$ encoding the route is built for various

values of the parameters α and β; in $\mathbf{v}_{\alpha,\beta}$ the bits at the positions $f_{\alpha\beta}(h(l))$ are set to one for each link l in the delivery tree:

$$\mathbf{v}_{\alpha,\beta}[f[\alpha,\beta](h(l))] = 1. \tag{13}$$

The form of the function f we selected is:

$$f_{\alpha\beta}[\lambda](\mu) = (\mu + \mu\lambda\alpha + \lambda\beta) \bmod m_{\mathbf{v}}, \tag{14}$$

where μ is the hash $h(l)$ of the link to recalculate, λ is the hash of the link where the packet was coming from, and α and β are the two parameters used for the optimization. We can notice that the number of parameters is specific to the particular form of the family of functions $f_{\alpha\beta}$. In particular, since we are using two operations (addition and multiplication) we need two parameters in order to tap into the complex relationship between addition and multiplication in modular arithmetic.

We have chosen to base the function on multiplication in the modulo arithmetic because this operation, despite being a relatively simple to calculate, is known to produce results which are versatile and unpredictable; in particular, this is why modular arithmetic is used in the RSA cipher [22]. The exact form of each term in this function is intended to provide as much variability as possible, given a very restricted range of parameters.

At this point, the number of false positive occurrences is evaluated for all the values of the parameters α and β comparing $\mathbf{v}_{\alpha\beta}$ with the off-route link identifiers $\mathbf{v}_{l_j,\alpha\beta}$. Since more than one pair (α, β) producing the smallest number of false positive occurrences is normally found, a further degree of freedom is offered by the scheme. For example, the choice can be made in

terms of network state: the pair selected will be the one which offers false positives[10] along the least congested links.

Once the optimization is performed, the optihash is composed from the bit array $\mathbf{v}$ and the selected values of $\alpha = a$ and $\beta = b$. The optihash is placed in the packet header to be used as the forwarding identifier; then the packet is ready to be forwarded. Each node maintains a forwarding table containing the hash arrays $\mathbf{v}_{l_i}$ constructed with the hashes $h(l_i)$ for each incoming and outgoing link; also, the node should be able to calculate the values of the function f.

When the packet reaches a node, the fields containing the values of the selected a and b are read. The forwarding nodes contain only the function f and the Link-Id (hashes) of the outgoing links. No state is necessary in the forwarding node. The link hashes of the outgoing links of the node and the hash of the previous link are calculated with the function f using the received values a and b; thus, the link hashes $f_{ab}(h(l_j))$ are formed and the correspondent bit array $\mathbf{v}_{l_j,ab}$ for each outgoing link is built. The $\mathbf{v}_{l_j,ab}$ array is used with the packet header information to make the forwarding decision for the incoming packet. Namely, the *and* and the *comparison* operations (9) are performed for each outgoing link. For the links such that (9) is true, the packet is forwarded.

The optihash scheme requires more processing at the stage of forming the forwarding identifier, since an optimization takes place. The complexity is related to the number of links $|D|$ to compare, the number of links $|E|$ along

[10]if the minimum of false positive is not zero.

the path and the number N of pairs (α, β) used for the optimization.

In total the complexity of the algorithm can be written as:

$$O(N|E| + N|D| + |D| + N) \approx O(N(|E| + |D|)) \tag{15}$$

since $|D| + N << N(|E| + |D|)$. Consequently, the complexity is directly proportional to the number of pairs N, and the sum of the number of elements stored in the optihash $|E|$ and the number of element to be queried $|D|$.

Also, slightly more processing is needed at the nodes for each packet that is forwarded since the function f has to be applied to the hashes (Link-Ids). The function f can be implemented with lookup-up tables accessible in one clock cycle. The amount of memory needed depends on the node degree:

$$\begin{aligned} OH_{memory} &= d(d-1)\ 2^{15} \text{ bytes} = \\ &= d(d-1)\ 32 \text{ kilobytes} \end{aligned} \tag{16}$$

where d is the node degree. For example, for $d = 4$, $OH_{memory} = 384\,kB$. Alternatively, we can consider to implement the function f in hardware but this is future work.

However, the size of the forwarding table remains unchanged and no state has to be placed in the forwarding tables. We have now to evaluate the false positive rate of the optihash and whether the reduction in false positive rate is significant considering the additional processing time.

5. Evaluation of the optihash false positive performance

The evaluation of the false positive performance of the optihash is carried out numerically through simulations. The benefits of using the optihash,

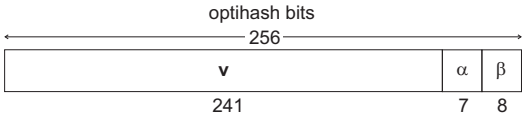

Figure 4: The optihash data structure for in-packet forwarding mechanism.

rather than the simple Bloom filter are quantified against the performance of the Bloom filter of size $m = 256$ bits; this is a practicable size used in other literature [4]. Therefore, the size of the optihash is selected to be the same with $m_{\mathbf{w}} = 256$ bits (see (7)) and the following sizes for $\mathbf{v}$, α and β are considered:

- $m_{\mathbf{v}} = 241$ bits.
- $m_{\alpha} = 7$ bits.
- $m_{\beta} = 8$ bits.

Thus, the size of the optihash is 256 bits and it is shown in Fig. 4.

Using simulations we perform our experimental evaluation to determine the properties which may influence the false positive rate of the optihash.

To this aim, we consider an artificial graph consisting of a variable size set of nodes with the same node degree as shown in Fig. 5.

For evaluation purposes, we model the optihash using the hashes $h(l)$ of the links rather than using the bit array $\mathbf{v}$. This implies that each link is identified by an hash $h(l)$ and each route is identified by a list of hashes which is the set R (see Sec. 3).

We consider routes of different length and we assign initially link hashes to each link which is either in the route or adjacent to the route (*off-route links*), as for example in Fig. 5. Note that the mechanism of using hashes

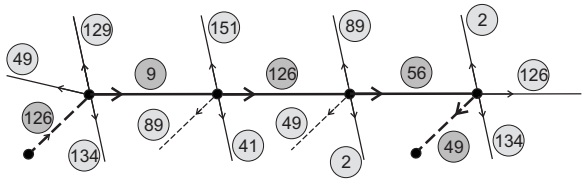

Figure 5: Through the link badges $h(l)$ the link identifiers $\mathbf{v}$ are built.

of links only gives *statistically* uniqueness of the link labels. However, we choose to enforce the uniqueness of outgoing links from a single node. In Fig. 5 there are examples of duplicated link labels due to the relaxation of global uniqueness through the use of statistical uniqueness provided by the hash function.

In order to be able to understand the dependencies of the false positive rate and properly design the experiment, we consider in detail the set of hashes, their sizes and we analyze how the false positive rate is calculated after the optimization to see which are the parameter which may influence its growth.

The set R used to build the route identifier $\mathbf{v}$, contains the link hashes of the links along the route. In contrast, the list $h(D)$ of the hashes of the off-route links to compare contains values that can appear more than once; consequently we refer to such a list as a *multiset* [23]. We note that R is truly a set, since even if along the route more than one link has the same hash, in the list encoding the route, the hash is represented only once. In this way, the cardinality $|R|$ of the set R, is not necessarily equal to the number of links along the route, i.e. $|R| \leq n$.

The number of false positive occurrences is the number of those hashes of the multiset $h(D)$ that are also present in the set R. The hashes with

this property constitute the multiset $h(F)$ of the hashes which generate false positives. Its cardinality is the number of false positive occurrences. The false positive rate is calculated according to (4) dividing the cardinality $|h(F)|$ of the multiset $h(F)$ by the cardinality $|h(D)|$ of the multiset $h(D)$:

$$f_{pr} = \frac{|h(F)|}{|h(D)|}. \tag{17}$$

To perform the optimization, new hashes $f[\alpha, \beta](R)$ and $f[\alpha, \beta](h(D))$ are calculated for all the values of the pair (α, β). The number of false positive occurrences and $f_{pr}^{[\alpha,\beta]}$ (17) are evaluated for each value of the pair. Now considering the set P of the pairs (α, β),

$$P = \{\, (1,1), (1,2), (1,3), ..., (2^{m_\alpha}, 2^{m_\beta}) \,\}, \tag{18}$$

the subset $M \subseteq P$ for which $f_{pr}^{[\alpha,\beta]}$ is a minimum created. The elements of M are the candidates for constructing the optihash. Clearly, the minimum value of $f_{pr}^{[\alpha,\beta]}$ depends on $|h(D)|$ and $|R|$. In fact, if $|h(D)|$ is high there will be more chances that one element of $h(D)$ is also in R, generating false positives. This implies that if there are more elements in $h(D)$ it is less probable to obtain a low value of the minimum of the false positive occurrences and also that it is likely that M will have fewer elements. Also, if $|R|$ is high, which means that the route contains a high number of links with different hashes, there will be more chances to have false positive occurrences and consequently it will be less probable to obtain a low value of the minimum of the false positive rate.

For this particular application, the cardinality $|R|$ is the number of on-route links with different hashes while $|h(D)|$ is related to the node degree and to the unicast or multicast nature of the traffic. If a node has a higher node degree, the number of elements in $h(D)$ will be higher than for a lower

degree node. In the same way, a unicast route through the same nodes will contribute more elements in $h(D)$ than a multicast route.

The false positive rate for the optihash is calculated as the ratio of the *minimum* of all $|h(F)|$ obtained with all the different values of the pair (α, β) and of $|h(D)|$. Namely, the minimum of the false positive occurrences is used to calculate the false positive rate of the optihash. Since the false positive rate of the optihash is defined as the minimum obtained through enumeration of a specific delivery tree, it will depends on the node degree, the path length, the hash function, and the unicast/multicast delivery because the minimum does. For these reasons, we evaluate the performances of the optihash in a regular network with different node degrees and for unicast and multicast routes.

We note that another way to carry out the optimization would be to calculate minimum of the *a posteriori* estimation of false positive rate for a particular value of (α, β) using ρ, the filling factor of $\mathbf{v}$, as defined by (3) with $k = 1$. In this case, the positive rate would not depend on $|h(D)|$ but only on how populated in the vector $\mathbf{v}$, namely on the cardinality of the set R. However, the use of the *a posteriori* estimation of false positive rate, during the optimization phase, gives rise to a much higher value than can actually be obtained for many particular instances for the specific delivery tree. Regarding the Bloom filter, the false positive rate does not depend on $|h(D)|$, as it does not encompass any optimization mechanism, unlike the optihash.

As an example of how the optihash works, with reference to the scenario of Fig. 5, the sets R and $h(D)$ are

$$\begin{aligned} R &= \{126, 9, 56, 49\} \\ h(D) &= \{49, 134, 129, 89, 151, 41, 49, 89, 2, 2, 126, 134\} \end{aligned} \tag{19}$$

and the multiset of false positives is

$$h(F) = \{49, 49, 126\}; \tag{20}$$

thus, the number of false positive occurrences is 3 and the false positive rate is $f_{pr} = \frac{|h(F)|}{|h(D)|} = \frac{3}{12} = 0.25$.

When the false positive rate is not zero, the function f is used to obtain alternative link hashes for the on- and off-route links and for each value of α and β the set R, and the multisets $h(D)$ and $h(F)$ are built. The distribution of the cardinality of $h(F)$ for the example considered is shown on the histogram in Fig. 6. In this example, there are 27487 pairs (α, β which result in 0 false positives, namely the cardinality of the set M is 27487; we can notice in this case, that $|M|$ is quite high. For practical implementation, the pair (α, β) in the set M can be chosen according to some heuristics intended to optimize some parameters of the network, for example the network state. A pair which gives the minimum of false positive rate (in this case the minimum is 0) is, for example, $(1, 2)$:

$$\begin{aligned} R_{12} &= \{126, 191, 74, 135, 14\} \\ h(D)_{12} &= \{210, 160, 7, 186, 83, 188, \\ &\qquad 210, 229, 25, 226, 65, 39\} \\ h(F)_{12} &= \{\}. \end{aligned} \tag{21}$$

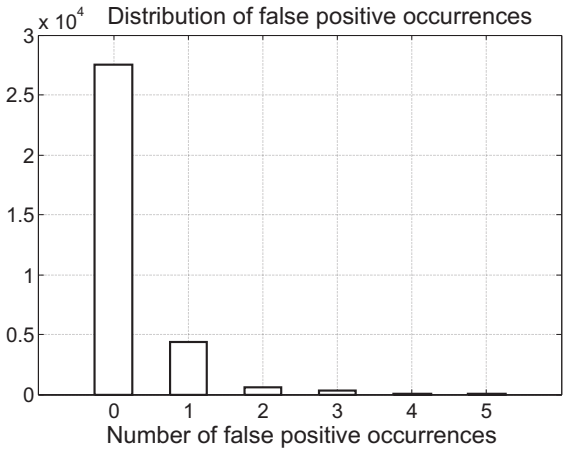

Figure 6: Distribution of the false positive occurrences for all the values of α and β for the example in Fig. 5. The majority of the pairs α and β gives link badges with no false positives occurrences.

For the experimental evaluation the false positive rate for the optihash is calculated using $\min(f_{pr})$, the minimum of the value defined in (17) over all values of α and β. The results are averaging over different trials obtained with new hashes in order to obtain stable values. Given that the false positive rate is a number around n, the number of trials is chosen to be $100\,\frac{1}{n}$. The results for the different parameters which influence the optihash performances are described below.

5.1. False positive rate for unicast routes of variable length through nodes with a fixed degree

Now, we shall consider unicast routes of different number of links along nodes with the same node degree equal to 5. First of all, we can demonstrate the impact of the optimization on the scheme from Fig. 7 where the false positive rate is shown for the scheme with and without the optimization. We

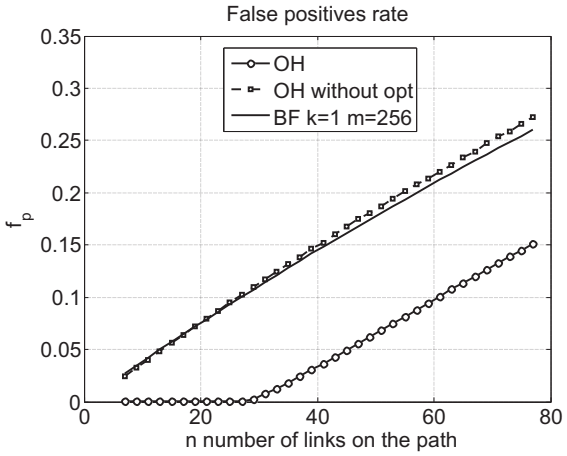

Figure 7: The false positive rate of the optihash with and without optimization for unicast routes of different length along nodes with node degree equal to 5. The analytical false positive probability for the Bloom filter with $m = 256$ and $k = 1$ is plotted.

can notice that the optimization does not influence the gradient of the curve.

The analytical expression (1) of the false positive rate for the Bloom filter with $m = 256$ and $k = 1$ is also shown, for comparison. The optimization offers a steady improvement of the false positive rate for routes with more than 30 links while for a lower number of links the false positive rate is nearly zero ($< 10^{-4}$) for the optihash model.

Next, the false positive rate of the optihash for different lengths of unicast routes is plotted in Fig. 7. It is compared with the false positive probability of the Bloom filter (1) with $m = 256$ and the optimal value of $k = k_{min}$ as defined in (2). Thus, for the Bloom filter part of the graph, for each n a new k_{min} (2) is calculated and a new $f_{p,min}$ (2) is shown on the plot; the value $f_{p,min}$ offers a lower bound of a realistically achievable false positive rate. The value $f_{p,min}$ is not realistic for a practical implementation, since it is not

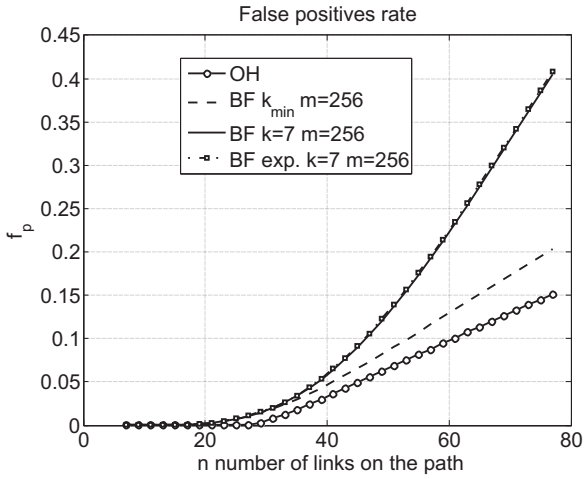

Figure 8: The false positive rate for the Bloom filter and for the optihash for unicast routes of different length along nodes with node degree equal to 5. The false positive probability has been computed for the Bloom filter using $k = 7$ and $k_{\min}$ and experimentally evaluated for $k = 7$.

feasible to use a different k for each route as it is has to be chosen as an *a priori* network constant. For a fairer comparison, the false positive rate of the Bloom filter is also plotted for a realistic fixed value of $k = 7$, which is the average over all the k_{min} calculated. In addition, the experimental values of the false positive rate for a Bloom filter with $m = 256$ and $k = 7$ are shown, since the (1) represents a lower bound for the probability (see Sec. 2.2).

We can observe from Fig. 8, as the number of links increases, the cardinality of R increases, and the false positive rate increases. However, the optihash consistently offers a lower false positive rate than the Bloom filter with k_{min} and, more importantly, a slower growth as the length of the route increases.

To evaluate the influence of the node degree, now we consider unicast

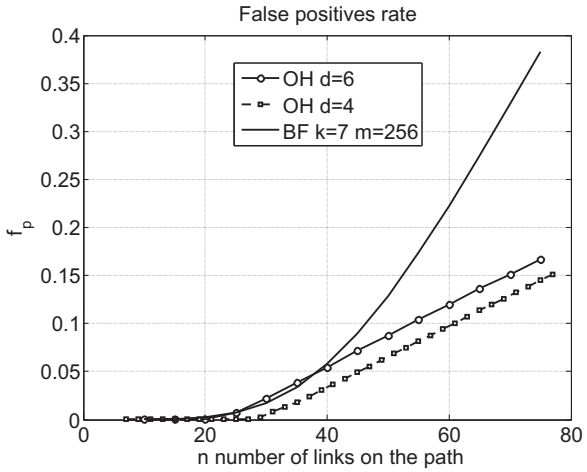

Figure 9: The false positive rate for the Bloom filter and for the optihash for unicast routes of different length along nodes with different node degree equal to 5 and 7. The false positive probability has been computed for the Bloom filter using $m = 256$ and $k = 7$.

routes of different number of links through nodes with the same node degree equal to 7. We plot the false positive rate of the optihash together with the graph for the degree 5, considered above. Fig. 9 shows that as the node degree increases, the false positive rate increases, since the cardinality of $h(D)$ increases. On average, this means that it is more difficult to find a pair (α, β) which assigns hashes with a low number of collisions. The minimal number of false positive occurrences increases. However, the increase is moderate, and the gradient of the graph is smaller than for the Bloom filter as the number of links in the path increases.

5.2. False positive rate for unicast routes of fixed length through nodes with variable degree

To evaluate the influence of the node degree, we consider a fixed length route of 50 links and we consider different node degrees. In this case we

keep the number of links on-route constant, namely the cardinality of the set E. In this case, since $R = h(E)$, it does not necessarily follow that $|R|$ is constant[11], however the fluctuation of $|R|$ is small and uniformly distributed so that we can assume that the variation in the false positive rate are due to the node degree only. As the node degree increases, the cardinality of $h(D)$ increases and therefore the probability to obtain a smaller minimal number of false positive occurrences decreases. A higher node degree offers a higher average false positive rate. Fig. 10 shows that as the node degree increase, the false positive rate increases. However, for a route of 50 links the false positive rate reaches the value for the Bloom filter with $k = 7$ for very high node degree.

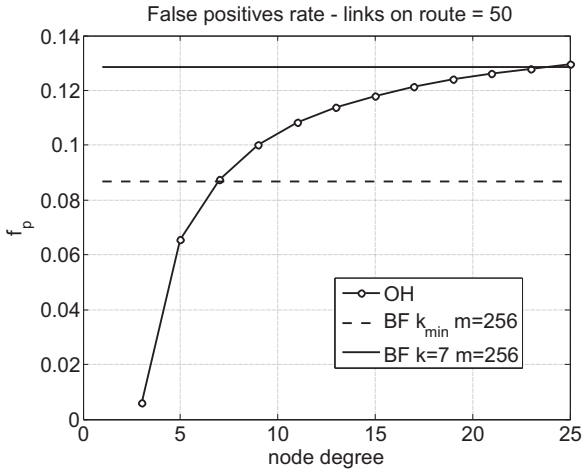

Figure 10: The false positive rate for the optihash in case of different node degrees for routes of 50 links for unicast. The false positive probability for the Bloom filter with $m = 256$, $k = 7$ and $k = k_{min}$ is plotted.

[11]It is constant if the cardinality of R if considered as a multiset.

5.3. False positive rate for multicast routes with a fixed number of destinations through nodes of fixed degree

Now we consider multicast routes through nodes with constant node degree (we consider node degree = 5) of different sizes with a fixed number of destinations (5 destinations). We compare the false positive rate of multicast with 5 destinations and unicast for routes composed of the same number of links. These graphs are plotted in comparison with the false positive probability of the Bloom filter with $m = 256$, $k = 7$ and $k = k_{min}$; note that the false positive rate of the Bloom filter depends only on the number of links but does not depend on whether one considers the multicast or the unicast model. From Fig. 11 it is clear that multicast routes allow the optihash to achieve lower minima of the false positive rate than unicast routes, because the multiset $h(D)$ contains fewer elements than in the case of unicast routes, for the same number of on-route links.

5.4. Fixed length multicast routes with different numbers of destinations through nodes of fixed degree

In order to be able to appreciate the influence of multicast on the optimization, we consider multicast routes of 36 links connecting nodes with degree equal to 5 with different number of destinations. Fig. 12 shows the false positive rate for the optihash against the false positive probability of the Bloom filter. Again, as described in Sec. 5.2, keeping the number of elements on-route constant, it does not necessarily follow that the cardinality of the set R is constant, however the fluctuation of $|R|$ is small.

Having a fixed number of elements on-route, as the number of destinations increases, $|h(D)|$ decreases, and, therefore, the probability to obtain a smaller

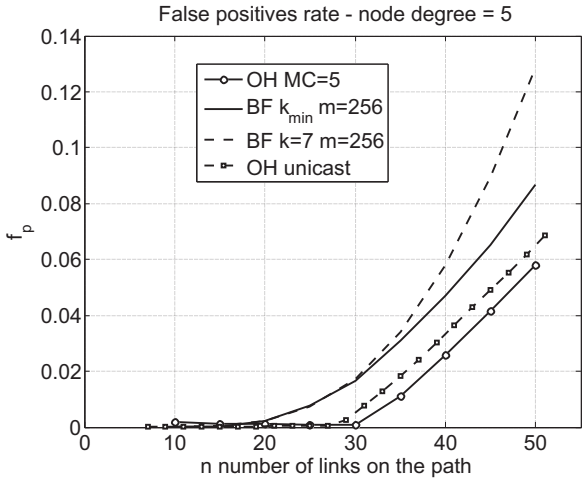

Figure 11: The false positive rate for the optihash in case of unicast and multicast routes with 5 destinations. The node degree of the network is 5. The false positive probability for the Bloom filter with $m = 256$, $k = 7$ and $k = k_{min}$ is plotted.

minimal false positive rate decreases. A higher number of destinations offers a lower average false positive rate for the optihash, in contrast with the Bloom filter, where the false positive rate remains constant. Indeed, the false positive probability for the Bloom filter does not depend on the number of destinations and in case of $m = 256$ and $n = 36$, $k_{min} \approx 5$ it is $f_p = 0.0331$. For a number of destinations higher than 13, the false positive rate of the optihash is very low ($\simeq 10^{-4}$).

We have to point out that the false positive rate of the optihash depends on the topology of the multicast tree. In this particular case we have considered a set of connected nodes and starting from the unicast case (see Fig. 5), we have moved links from the set $h(D)$ to the set R to generate multicast. Generally, the false positive rate of the optihash depends on the cardinality

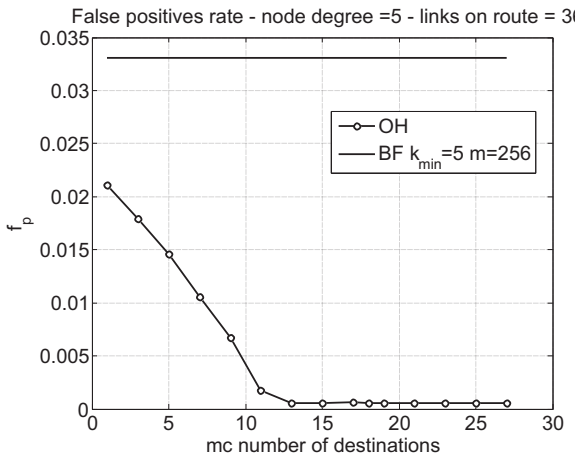

Figure 12: The false positive rate for the optihash in case of different destinations multicast routes of 36 links. The node degree of the network is 5. The false positive probability for the Bloom filter $f_p = 0.0331$ with $m = 256$, $n = 36$, $k = k_{min} = 5$ is plotted.

of the set R and the multiset $h(D)$.

6. Conclusions

In this paper we have proposed a new variant of the Bloom filter, the optihash which aims to control the false positive rate at a cost of slightly more processing. It offers an optimization mechanism on the false positive occurrences for the specific data set to be encoded in the optihash. The optimization space for the choice of the element descriptors is large but the scheme is flexible enough to change/reduce it. Moreover, it offers a robust (with a statistic meaning) control over the false positive occurrences. We have analyzed through simulations the performances of the optihash scheme for the in-packet forwarding function in the PSIRP architecture. The numerical model of the false positive probability of the optihash has been compared

with the false positive probability of a Bloom filter of comparable size. We have assumed an artificial graph composed of a set of nodes with equal node degree. This has been assumed in order to be able to study the behavior of the optihash depending on the network parameters. If our aim is to reduce false positive occurrences, the optihash offers a considerably lower false positive rate than the Bloom filter especially in the case of multicast. It also offers more flexibility at a cost of slightly more processing.

Acknowledgment

This work has been carried out through the support of the EPSRC and British Telecom (BT) through an EPSRC CASE award studentship and through an EU FP7 project PURSUIT under grant FP7-INFSO-ICT 257217. The authors would like to thank Dirk Trossen for the fruitful discussions.

References

[1] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13, no.7 (1970) 422–426.

[2] A. Broder, M. Mitzenmacher, Network applications of Bloom filters: A survey, Internet Mathematics 1, no.4 (2004) 485–509.

[3] S. Tarkoma, C. Rothenberg, E. Lagerspetz, Theory and practice of Bloom filters for distributed systems, IEEE Communications Surveys and Tutorials 14, no.1 (2012) 131–155.

[4] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, P. Nikander, LIPSIN: Line speed publish/subscribe inter-networking, ACM SIGCOMM '09, Barcelona, Spain.

[5] C. Rothenberg, C. Macapuna, F. Magalhaes, F. Verdi, A. Wiesmaier, In-packet Bloom filters: Design and networking applications, Elsevier Computer Networks 55, no.6 (2010) 1364–1378.

[6] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, R. Braynard, Networking named content, Communications of the ACM 55, no.1 (2012) 117–124.

[7] D. Trossen, G. Parisis, Designing and realizing an information-centric Internet, IEEE Communications Magazine, Special Issue on Information-centric Networks 50, no.7 (2012) 60–67.

[8] A. Zahemszky, P. Jokela, M. Sarela, S. Ruponen, J. Kempf, P. Nikander, MPSS: Multiprotocol stateless switching, 13th IEEE Global Internet Symposium 2010, San Diego CA USA.

[9] M. Särelä, C. E. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, J. Ott, Forwarding anomalies in Bloom filter based multicast, in: 30th IEEE International Conference on Computer Communications (IEEE INFOCOM 2011), Shanghai, China.

[10] L. Mackett, G. Lohamn, R* optimiser validation and performance evaluation for distributed queries, in: M. Kaufmann (Ed.), Proceedings of the Twelfth International Conference on Very Large Databases 1986, San Francisco, CA, pp. 149–159.

[11] J. Mullin, Optimal semi-joins for distributed data systems, IEEE Transactions on Software Engineering 16, no.5 (1990) 558–560.

[12] D. Lagutin, K. Visala, S. Tarkoma, Publish/subscribe for Internet: PSIRP perspective, in: Future Internet Assembly 2010, pp. 75–84.

[13] M. Mitzenmacher, Compressed Bloom filters, IEEE/ACM Transaction on Networking 10, no.5 (2002) 604–612.

[14] J. Mullin, A second look at Bloom filters, Communications of the ACM 26, no.8 (1983) 570–571.

[15] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, Y. Tang, On the false positive rate of Bloom filters, Inf. Process. Lett. 108, no.4 (2008) 210–213.

[16] K. Christensen, A. Roginsky, M. Jimeno, A new analysis of the false positive rate of a Bloom filter, Information Processing Letters 110 (2010) 944–949.

[17] D. Meyer, L. Zhang, K. Fall, Report from the IAB workshop on routing and addressing, RFC 4984 (Informational), 2007.

[18] M. AL-Naday, R. A. Jr, K. Guild, M. Reed, Design proposal of a photonic multicast Bloom filter node, accepted in Photonic Network Communications (2012).

[19] C. Rothenberg, Compact forwarding: A probabilistic approach to packet forwarding in content-oriented networks, Ph.D. thesis, University of Campinas - Brasil, 2010.

[20] S. Lumetta, M. Mitzenmacher, Using the power of two choices to improve Bloom filters, Internet Mathematics 4, no.1 (2007) 17–33.

[21] A. Pagh, R. Pagh, S. Srinivas Rao, An optimal Bloom filter replacement, in: SIAM (Ed.), Proceeding of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms 2005, pp. 823–829.

[22] N. Smart, Cryptography: An introduction, McGraw-Hill, 2003.

[23] W. Blizard, Multiset theory, Notre Dame Journal of Formal Logic 30, no.1 (1989) 36–66.