LIPSIN: Line Speed Publish/Subscribe Inter-Networking

TR09-0001

Document Properties:

Title of Contract	Publish-Subscribe Internet Routing Paradigm
Acronym	PSIRP
Contract Number	FP7-INFSO-IST 216173
Start date of the project	1.1.2008
Duration	30 months, until 30.6.2010
Document Title:	LIPSIN: Line speed Publish/Subscribe Inter-Networking
Date of preparation	09.01.2009
Author(s)	Petri Jokela (LMF), Andras Zahemszky (LMF), Somaya Arianfar (LMF), Pekka Nikander (LMF) and Christian Esteve (LMF / Unicamp)
Responsible of the deliverable	Petri Jokela Phone: +358 9 299 2413 Fax: +358 9 299 3535 Email: petri.jokela@ericsson.com
Target Dissemination Level:	PU
Status of the Document:	Final
Version	1.00
Document location	http://www.psirp.org/
Project web site	http://www.psirp.org/

Table of Contents

LIPSIN: Line speed Publish/Subscribe Inter-Networking

Petri Jokela, Andras Zahemszky, Somaya Arianfar, Pekka Nikander
Ericsson Research, NomadicLab
02420 Jorvas, Finland
firstname.secondname@ericsson.com

Christian Esteve Rothenberg
University of Campinas (UNICAMP)
School of Electrical and Computer Engineering
P.O. 6101, Sao Paulo, Brazil
chesteve@dca.fee.unicamp.br

January 11, 2009

Summary

In an attempt to solve many of the problems in the current Internet, a number of so-called clean slate approaches have been proposed. Common to many of them is to focus on data, rather than nodes, and to use some sort of publish/subscribe, instead of send/receive, as the main API abstraction. In this paper, we explore further the space where the pub/sub paradigm replaces send / receive completely, throughout the stack. In particular, we introduce a new efficient forwarding architecture that naturally supports multicast and may be relatively easily extended to support networking functions beyond forwarding, including network coding and caching. The novelty of our approach consists of compacting source route link identifiers in small Bloom-filter-based packet headers. The suggested method is simple and can be readily implemented in hardware for fast forwarding. Although we focus on a pub/sub-based architecture in this paper, our forwarding method is not limited to pub/sub. It can be used in any network setting with known link identifiers and sufficient topology information allowing larger than hop-by-hop forwarding decisions.

Chapter 1

Introduction

The present host-oriented way of inter-networking [19] is based on routing packets using the destination host IP address. The network was designed to serve the packet sender; it delivers the packet to the receiver the best it can. However, this scheme does not consider the possibility that the receiver may not be willing to receive that particular piece of data; the assumption was that the sender will honour the recipient's consent. Hence, the original design of the Internet has led to an imbalance of powers, in which the sender has too much control over the network, compared to the receiver. This is a root cause for a number of well-known problems, such as unwanted traffic.

A number of mechanisms have been developed and turned out to be useful to fight against problems caused by the present design. For example, firewalls are used to protect hosts by aiming to block all unwanted traffic, and NAT boxes provide partial protection by simply fracturing connectivity [38, 39]. However, from a larger perspective, such mechanisms only lead to an armament competition between those wanting protection and those wanting connectivity, for legitimate or illegitimate reasons, thereby further reducing the potential utility of the Internet.

In contrast to this background, information-centric inter-networking provides a new model for communication [19]. In an information-centric model, the focus is on data, not nodes. The underlying nodes and hosts become more-or-less impartial, possibly to an extent where they no longer have persistent, IP-address-like names (cf. with [4]). We are not interested in the name of the host that has converted the original information into the form and delivered it to us, as long as the data is timely and correct.

In this paper, we focus on how to implement topic-based publish/subscribe [15], or pub/sub for short, directly on the top of raw hardware, in a scalable manner. To our knowledge, the proposed scheme is the first one that has the potential of supporting pub/sub in Internet-like scales, allowing one to realistically consider pub/sub, instead of IP-like approaches, as an inter-networking paradigm. Furthermore, our scheme is not limited to publish/subscribe networks; instead, it can be used in any network that has a rich-enough topology layer so that

our combination of source routing and stateful routing can be implemented; see Section 3. Hence, it is plausible that the method could be used, for example, to implement efficient IP multicast.

When designing a routing and forwarding system, one has to consider the balance between the information stored in the forwarding nodes and the present in the packet headers. In the basic IP design, the packet header contains the destination address and all the forwarding nodes know their best next hop towards all destinations. The only known way to make that design to scale is to aggregate the address space so that state is needed only for each aggregate [14].

At the opposite end from the present Internet design lies strict source routing [36], with its well-known problems related to packet sizes and security [5]. In strict source routing, the packet's path is described, hop by hop, in the packet header. A single forwarding node does not have to know anything else than its neighbors; it just picks the next hop node from the packet header and delivers the packet.

Our approach is based on an assumption that there are no stable end-to-end *addresses* for the network nodes, for three reasons. First, such addresses might foil most of our envisioned benefits in fighting unwanted traffic. Second, in an information-centric architecture long-lived node addresses should not be needed. Third, any such addresses used as identifiers are detrimental to the ability of supporting mobility and multi-homing.

Such node-address-less design generates new kind of problems, especially for routing and forwarding. Our Bloom-filter-based mechanism solves the forwarding problem without end-to-end addressing, using a link-identifier-based approach that combines elements from source routing and stateful routing, in a flexible way. In particular, we show that our approach leads to fast hardware-implementable forwarding decisions at the forwarding nodes, reduces the possibilities for malicious nodes for sending unwanted traffic, and at the same time could be scaled to Internet-wide scales.

Hence, the main contribution of this paper is showing that it is feasible to build a scalable and secure multicast forwarding layer, allowing one to explore the space of source-addressable inter-networking instead of the present destination-addressable internetworking schemes. In particular, our results indicate that it should be possible to build flat content-based name spaces in a scalable manner, contrary to previous results [8].

The rest of this paper is organised as follows. First, in Section 2, we motivate our work and give an outline of our proposed methods. In Section 3, we go into details of the design. Next, in Section 4, we provide extensive evaluation and analysis, in the form of simulation results, to support the validity of the scalability claims and accuracy of the results reported. In Section 5, we briefly describe our early FreeBSD-based end-node implementation, provide some initial performance measurements from our NetFPGA-based forwarding-node implementation, and discuss our plans for continuing with the implementation. Section 6 contrasts our work with related work, and Sec. 7 concludes the paper.

Chapter 2

Towards scalable pub/sub

2.1 Background

In the current frenzy of the so called clean slate network designs, different sorts of publish/subscribe have become one of the more promising approaches [11, 24, 32, 37]. While the term "publish/subscribe" covers a wide variety of networking approaches [15], the so-called topic-based publish/subscribe seems to offer the best scalability. In our envisioned world, internet-networking is based on topic-based publish/subscribe rather than the present send / receive paradigm [12, 32, 37]. In topic-based publish/subscribe, the basic unit of publication and subscription is a topic, identified by a unique identifier.

From an architectural point of view, in a topic-based pub/sub a topic can be thought as an identifier of a channel. Whenever there are events related to the topic, information is delivered over the channel from the event source to the subscribers. For example, if the topic refers to a document, whenever a publisher publishes a new version of the document, the new version (or metadata describing it) is sent.

An important feature of pub/sub inter-networking is that would allow us to protect the hosts against most types of unwanted traffic. Basically, pub/sub requires a subscription by the subscriber before any data moves. The architecture becomes more balanced in power than the current Internet, allowing both the sender and the receiver to exert a roughly equal amount of control over the network.

The pub/sub abstraction allows the sender and the receivers to be separated by time. For example, a subscriber may indicate its interest a long time before a publisher has anything to publish, while another subscribe may come along only later, once the publisher has already published some data, and get the latest version of the data from a cache.

Overall, our aim is an information-centric inter-networking system that scales to tens of billions of nodes and some 10^{15} active pub/sub topics, at the same time reducing the relative amount of unwanted traffic by at least an order of

magnitude from its present level.

2.2 A straw-man design

Now, while almost all of the so-called *clean slate* designs are somehow based on the assumption of an underlying network that forwards packets more-or-less similar to the current Internet, we have envisioned a "post IP" world where the inter-networking layer is based on topic-based pub/sub [12, 32, 37]. Hence, instead of routing and forwarding packets based on a destination address, in our architecture packets are ultimately routed based on a topic identifier. Furthermore, we want to build a network where the receiver has a great deal of control of what to receive, without the network being able to exert its power in the form of content-based price discrimination [41].

Using these assumptions, it becomes natural to leave any solutions based global end-to-end addressing behind, and move to the unmapped territory. A straightforward way of implementing information-centric data delivery, where the pieces of data have an identity of their own, is to set up a time-dependent delivery-path or tree from the publisher to the subscribing nodes, based on the topic identity. In such a solution, the routers along the path or tree are explicitly configured to forward data based on the particular topic.

It is obvious that such a simple solution does not scale. First, the number of potential topics and transmission events in the network is huge. The size of the forwarding table in a single router would grow extremely large, as each new subscription generates a separate piece of state at all routers residing on the transmission path between the existing tree and the new subscriber. Second, updating the forwarding tables due to subscribers' churn would generate a large amount of control traffic and take too long to process.

The straw-man design can be contrasted with IP multicast. In IP multicast, the data source defines a group ID for the delivered data. The group ID is announced to the potential receivers, who need to set up a connection towards the source node. In practise, a receiver "joins" the multicast group, and the network forms a multicast tree based on the "join" messages. As it is well known, the approach has both scalability and deployability problems, as it, too, involves adding state to all of the intermediate routers.

Our approach provides a flexible design where the network control functions can vary the amount of routing information placed in the packet header vs. installed at the network nodes.

In the following, we first briefly describe our overall pub/sub based inter-networking architecture, and then present a forwarding solution that resembles strict source routing but uses a *fixed-sized*, compact header, using Bloom filters, suitable for fast hardware implementation. Furthermore, our solution *hides the network structure* so that it is difficult, using only the forwarding information from the packet headers, to figure out the path that any given packet will take.

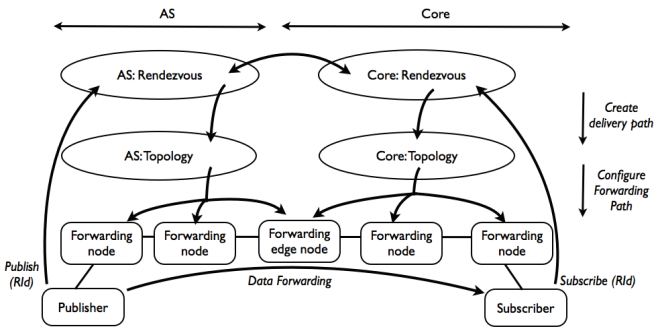


Figure 2.1: Rendezvous, Topology, and Forwarding in pub/sub architecture

2.3 The Network Architecture

In our topic-based pub/sub architecture, the topics do appear at various layers in the architecture, at the lowest layers representing single packets, higher up channels, and still higher up documents. The applications can request publish or subscribe access to these entities, typically asking existing entities in their virtual memory to be published in the network, or network-based entities to be subscribed and mapped to their virtual memory space.

While the overall architecture of the system is still work in progress (cf. [12]), the approach can be described through a recursive approach [32], depicted in Figure 2.1. At the bottom of the architecture lies the forwarding layer, the main focus of this paper. Its main responsibility is to forward packets from their sources to their sinks in an efficient, best-effort manner. The structure above the forwarding layer can be divided into a data and control plane. At the *control plane*, the topology management system creates a distributed awareness of the structure of the network, similar to what today's routing protocols do. On the top of the topology management system lies the rendezvous system, which is responsible for matching the interests of the publishers and subscribers.

At the *data plane*, at the top of the forwarding layer, in addition to the traditional transport functions, such as error detection and traffic scheduling, we envision a number of new network functions, such as opportunistic caching and lateral error correction. The transport functions will work in concert, utilising each other in a *component wheel* [12], similar to the way Haggle managers are organised [33].

As in most pub/sub schemes, communication is based on multicast. In our design, the basic communication scheme is functionally similar to IP-based source specific multicast (SSM), with the IP multicast groups having been replaced by the topic identifiers[1]. Hence, in our architecture multicast is used as the basic transmission method, and unicast is a special case of multicasting

[1]For multi-sender single-receiver situations, some sort of concast will be supported; see Section 6.3.1 of [12].

with only one receiver.

Each publication is identified with a Rendezvous Identifier (RId). For the purposes of this paper, it is sufficient to consider each RId to be unique, acting as a handle to the publication[2]. When a publisher wants to publish a new piece of information, it acquires a new RId from the system and instructs the system where in its virtual memory space the publication data can be found. Correspondingly, when a subscriber wants to get access to a piece of information, it acquires the RId through an application specific means, e.g. from a well-known directory, and asks the system to arrange the data to appear at its virtual memory. It is the responsibility of the node-local part of the network to actually announce each new publication to the rendezvous system, and correspondingly, to request the rendezvous system to (eventually) arrange the data delivery to take place.

Once the rendezvous system has identified a publication that has both a publisher (or an up-to-date cache) and one or more subscribers, the network requests the topology system to build a forwarding tree from the present location of the data to the subscribers. While sounding trivial, the actual implementation of the functionality in a scalable manner turns out to be quite complex.

In this paper, we concentrate mostly on channel-granularity publications and only on the forwarding layer, including the way the topology system controls the forwarding layer. We do not describe any particular solution for the rendezvous mechanism or for the topology collection phase of the topology system, and we do not discuss the transport functions at the data plane. Furthermore, we mostly focus on intra-domain forwarding, i.e. forwarding inside a single autonomous system (AS). As our architecture is recursive, the method can be applied also on inter-AS routing.

2.4 Recursive bootstrapping

As mentioned above, we are creating a new network architecture on the assumption that there is no end-to-end addressing scheme deployed. The only initial way of communication is link-local broadcasting, used to bootstrap the topology mapping and rendezvous systems [30]. As a part of the bootstrapping process, we expect the topology system to create local static forwarding paths for the purposes of the rendezvous system, thereby allowing publication announcements and subscription requests to be forwarded between any nodes and the rendezvous system.

This approach is applied in a recursive manner. That is, while at the local links the rendezvous system runs directly on the top of the local broadcasting function, without any topology management at all, at the higher layers the rendezvous system relies on the static forwarding paths, created during the bootstrap and provided by the lower layers.

[2]More precisely, each publication is identified with a $< ScopeID, RendezvousID >$ pair. The scopes both help the rendezous system to scale and to organise the publications.

We bootstrap the system bottom-up, assuming recursively that the layer below offers (static) connectivity between any node and the rendezvous system. At the lowest layer, this assumption is trivially true since any two nodes connected by a shared link (wireline or wireless) can, by default, send packets that the other node(s) can receive.

During the bootstrap process, the topology management functions on each node learn their local connectivity, by probing or relying on the underlying layer to provide the information. Then, in a manner similar to the current routing protocols, they exchange information about their perceived local connectivity, creating a map of the network graph structure. The same messages are simultaneously used to bootstrap the rendezvous system.

2.5 Forwarding on Bloomed link identifiers

In order to forward packets through the network, we use a hybrid approach where the topology system both constructs Bloom-filters-based forwarding identifiers, used in a source-routing manner, and on demand installs new state at the forwarding nodes. For both of these functions, we use an approch where the links, not the nodes, have names.

For each point-to-point link, we assign two identifiers, called Link IDs, one in each direction. For example, a link between the nodes A and B has two identifiers, $\overrightarrow{AB}$ and $\overleftarrow{AB}$. In the case of a multipoint link, such as a wireless link, we consider each pair of nodes as a separate link. With this setup, we don't need any common agreement between the nodes on the link identities – each link identity may be locally assigned, as long as the probability of duplicates is low enough[3].

Basically, a Link ID is an m-bit Bloom filter whith just k bits set to one. In Sec. 4 we will discuss the proper values for m and k, and what are the concequences if we change the values; however, for now it is sufficient to note that typically $k \ll m$ and m is relatively large, making the Link IDs statistically unique (e.g., m=256, k=5, Link IDs $\approx m!/(m-k)!$). As briefly explained above, the topology management system creates a graph of the network using the Link IDs and connectivity information (creating the "topology map" or "routing table"). Using the network graph, the topology system can determine a forwarding tree for any publication, considering the locations of the publisher and subscribers.

Now, when the topology management system gets a request to determine a forwarding tree for a certain publication, it first creates a conceptual delivery tree for the publication using the network graph. Once it has such an internal representation of the tree, it knows which links the packets need to pass during delivery, and it can determine when to use Bloom filters and when to create state.

[3] As we will see later, under certain settings it is beneficial to coordinate the link identifiers in a centralised manner, reducing the amount of forwarding loops and false positives.

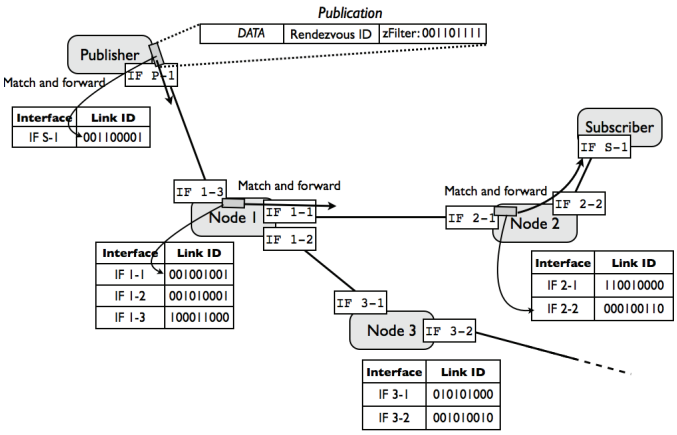


Figure 2.2: Example of Link IDs assigned for links, as well as a publication with a zFilter, built for forwarding the packet from the Publisher to the Subscriber.

In the default case, we use the source-routing-based approach. Basically, we encode the all Link IDs of the tree into Bloom filter, placed in the packet header. By having Link IDs taking the form of a BF-compatible bit vector structure (m-bit long with k one bits), the topology module can create a usable in-packet representation of the tree by simply ORing the Link IDs. Once all link IDs have been added to the filter, a mapping from the rendezvous identifier (RId) to the BF is given to the node acting as the data source, which can now create packets that will be delivered along the tree. To distinguish the BFs in the actual packet headers from other BFs, the in-packet Bloom filters (together with an addional byte) are referred as zFilters[4].

Each forwarding node acts on incoming packets roughly as follows. For each link in its forwarding table, the outgoing Link ID is ANDed with the zFilter. If the result matches with the Link ID, it is assumed that the Link ID has been added to the zFilter and that the packet needs to be forwarded along that link.

With Bloom filters, matching may result with some false positives. In such a case, the packet is forwarded along a link that was not added to the zFilter. However, we do not consider this a problem as long as the rate of false positives remains sufficiently low; see Section 4.1. Indeed, it is even possible to take advantage of some packets being transmitted unnecessarily: the packets can be opportunistically cached and used to fulfill possible future subscriptions.

Fixed-sized Bloom filters have a finite capacity; when the number of links exceeds this capacity, the rate of false positives would raise to an unbearable level. While we are able to increase the capacity somewhat through using some clever coding choices, our fundamental approach in solving the capacity problem is twofold: Firstly, we use recursive layering [10] to divide the network into

[4]The name is not due to `zfilter.com` nor the e-mail filter, but due to one of the authors reading Franquin's Zorglub for the Nth time during the early days of the presented work.

suitably-sized components; see Section 3.4. Secondly, and perhaps more importantly, the topology system may dynamically add *virtual links* to the system[5].

A virtual link is, roughly speaking, an unidirecitional delivery tree that consists of a number of links. It has its own Link ID, similar to the real links. Furthermore, the functionality in the forwarding nodes is identical: the Link ID is compared with the zFilter in the incoming packets, and the packet is forwarded with a match. The difference is that a virtual Link ID is associated with several outgoing links at several routers, causing matches over the whole path.

In this section, we have outlined our architecture and especially the forwarding system. As the astute reader has realised, the basic solution has some shortages, e.g., how to deal with broken links or the (rare) loops that the system inherently imposes due to the false positives. In the next sections, we describe a number of enhancements to this basic forwarding solution, dealing with these shortcomings.

[5]The virtual links may also be used to implement the aformentioned static forwarding paths, needed by the rendezvous system.

Chapter 3

Design

In the previous section, we described our basic Bloom-filter-based forwarding mechanism using Link IDs. In this section, we first consider the overall design of the system. The rendezvous and topology functions are described without going into too much details, on a level that is needed for understanding the underlying forwarding layer operations. The forwarding layer itself is described in detail, including *Link ID Tags* generation, creation and selection of the forwarding Bloom filters, loop prevention, virtual links, fast recovery, and stateful operations.

3.1 Rendezvous and Topology

The main purpose of the *rendezvous* system is to keep track of offered publications and to find publications matching with subscriptions. The *topology* system has two main functions: to autonomously collect and maintain a representation of the network graph (i.e. the "routing table"), and to create zFilters and virtual links in response to requests from the rendezvous system. That is, the topology system determines the most suitable delivery trees that are then implemented at the forwarding layer in some combination of zFilters and virtual links.

An important aspect of the topology system is to optimize the trees, both in the terms of latency as well as on, e.g., the amount of traffic and network usage. For the purposes of this paper, we assume that for each request the topology module initially selects a (semi)optimal delivery tree, to be implemented by the forwarding layer.

3.2 Link IDs and LITs

We now introduce the concept of *Link ID Tags* (LITs) as an addition to the plain Bloom-filter compatible Link IDs. That is, instead of each link being identified with a single Link ID, every unidirectional link is associated with a set of d LITs, uniformly computed by applying some mapping function (see Fig. 3.1).

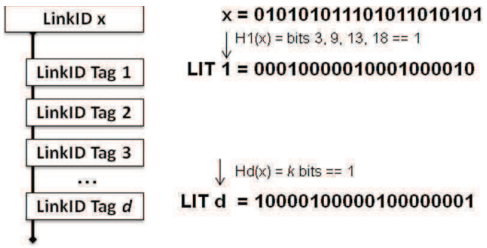


Figure 3.1: Relation of one link identifier (link ID) to d Link ID Tags (LIT). Example of $d - LIT$ generation by using k hashes the Link ID.

That allows us to construct zFilters that can be optimized in terms of the false positive rate and/or compliance with network policies. By introducing the LITs, the actual Link IDs no longer need to be BF compatible but are free to take any form (e.g., our baseline implementation uses 256-bit random bit vectors).

Hence, for each link, there are d LITs, where d is a system parameter that can vary depending on the network. As we explore later, a practical value of d is in the range of multiples of 2 between 2 to 32. The key idea is that we have a set of LITs that each equivalently represents a given link. That allows us to take advantage of the random distribution of the LITs, selecting them in a way that leads to better performing BFs.

In the default configuration, each node assigns a random Link ID and computes the corresponding d LITs for each of its outgoing links that it knows of. This information is locally maintained in the topology system with d forwarding tables containing the LIT entries of the active Link IDs.

Upon packet arrival, the zFilter and the corresponding LIT entries in the forwarding table are ANDed in parallel. The packet is then forwarded on all matching outgoing links (other than the link where the packet arrived from). Fig. 3.2 provides a conceptional view of a forwarding element. The operation is so fast that a long packet packet can be on its way out before the tail of the packet has been received.

To achieve the maximal entropy in the bit distribution of each LIT, it would be required to have k different hash functions per LIT[1]. However, we use the *double hashing* technique to generate the d LITs efficiently, using only two random independent hash functions, without any increase of the asymptotic false positive probability. That is, we rely on the result of Kirsch and Mitzenmacher [23] on linear combination of hash functions, concluding that in practice *double hashing* matches the performance of random hashing. Two independent hash functions $h_1(x)$ and $h_2(x)$ are used simulate i hash functions of the form:

$$g_i(x) = h_1(x) + i * h_2(x) + i^2 mod\ m \tag{3.1}$$

Moreover, simple hash functions have been shown [23] to perform well enough

[1]The number k can be different for each LIT, which allows us to adapt to the different fill factors of the BF, thereby yielding lower false positive rates.

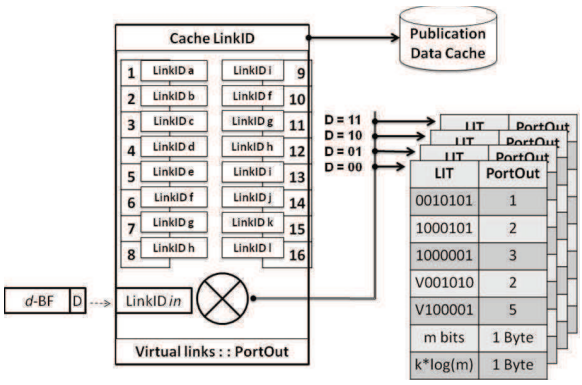


Figure 3.2: Outgoing interfaces are identified by a Link IDs and d forwarding tables indexed by Link ID Tags are maintained.

when the entropy of the elements to be hashed is sufficiently large, which is the case of our pseudo-randomly generated link identifiers.

Furthermore, using the double hashing method, any LIT generation process can be compactly represented by the set of indices i that were used to compute the k positions of the LIT set to 1 (see forwarding table discussion in Sec. 4.4). Therefore, as $h_1(x)$ and $h_2(x)$ are system wide parameters, only sharing $d * k$ integers i is required to derive the LITs for any Link ID.

3.2.1 zFilter construction using LITs

The Link ID Tags enable a more flexible BF construction scheme by taking advantage of the power of choice [26] during the probabilistic process of generating the zFilters. For each of the d LITs of every link to be inserted in the BF, a candidate BF is generated by ORing one LIT representation at a time of each link[2] (see formal example algorithm 1). This results in having d candidate BFs that are "equivalent" representations of a certain delivery tree, meaning that the packet using any of the candidates will follow, at minimum, all the network links inserted into the BF.

[2]In the case we allow different links having a different number of LITs, we simply reuse the LITs in a modular fashion. For example, if a link as $d = 4$ and we are constructing the 7th candidate, we use the 3rd LIT.

Algorithm 1: Construction of d candidate BFs and selection based on expected false positives.

Input: Link IDs of the delivery tree
Output: zFilter to be used as forwarding identifier

foreach *Link ID in Delivery Tree* **do**
  foreach *Link ID Tag* $LIT_i / \ i \in [1:d]$ **do**
    Candidate $BF_i \leftarrow BF_i$ OR LIT_i;
  end
end
// Procedure to apply the filter selection criteria (e.g., BF with the best fill factor)
Return zFilter $\leftarrow$ SelectBest(BF_i);

Due to the probabilistic nature of the LITs, each candidate BF constructed by ORing the LITs will look different regarding the number and position of the bits set to 1. The fill factor ρ (or the ratio of 1's to 0's) in the candidate BFs will be key when counting for false positives. It can be shown that the false positive rate of a BF is a function of the probability that a bit cell is empty (see analysis in Sec. 4.1).

In addition to the actual BF, the packet header must now include the LIT set index d of the selected BF. At packet forwarding time, the nodes receiving the d-enhanced BF can pick the dth[3] forwarding table entry that includes the correct LIT, i.e. the same one that was used during the construction time. Our approach thereby enables finer control over the effects of false positives while keeping the appealing properties of the Bloom filters.

3.2.2 Selecting the optimal filter

When optimizing forwarding, for better performance in terms of lower false positive probability we first consider two different, relatively simple strategies:

Lowest false positive after hashing (fpa): When every LIT contributes with k bit positions set to one, selecting the best candidate BF implies simply choosing the candidate BF with the lowest amount of 1's. If different number k of bits were used for the LIT representations, the selected BF should be the one with the lowest false probability estimate after hashing: $min\{{\rho_0}^{k_0}, \ldots, {\rho_d}^{k_d}\}$. See details in Sec. 4.1.

Lowest observed false positive rate (fpr): Given a test set T_{set} of link IDs, the candidate BF can be chosen after counting for false positives against T_{set}. The objective is to minimize the observed false positives when querying against a known set of Link IDs active in the routing elements along the delivery path.

The *fpa* strategy is simple and aims at lower false positives rates for any set of link IDs under membership test. On the other hand, the *fpr* yields the

[3] dkth entry in the case there are only k LITs associated with the link

best performance of false positives for a specific test set at the expense of higher complexity.

Going further, the BF selection criteria can also take other network policies into consideration beyond false positives. For instance, we can check for false positives of specific link IDs that may be troublesome, such as forwarding packets towards non-peered domains, resource constrained regions, or potential loops. Any such candidate BF would then be disregarded, even though it might offer better false positive estimates in absolute terms. We call such selection criteria as *link avoidance*, since it is based in penalizing those candidate BFs that yield false positives when tested against sets of LITs to be avoided. Such link test sets (T_{set}), for a specific criterion, can be weighted against each other when choosing the final BF. For example, the following kinds of criteria could be considered:

Routing policies: T_{set} of links to be preferably avoided due to inter/intra-domain routing policies.

Congestion mitigation: *Static* T_{set} of links to be avoided due to traffic engineering (e.g., low capacity links) and *dynamic* T_{set} containing links suffering from congestion.

Subscribers density: T_{set} of links that lead to a dense area of subscribers where false positives are expected to cause more cascaded false positives and undesired traffic.

Security policies: T_{set} containing links to be avoided due to security concerns.

With the d-choice LIT BF construction scheme, we have a powerful method to construct a compact forwarding identifier that can reduce the effects of packets by reducing and confining false positives to certain network areas. Increasing the power of choices (larger d values) leads to more possible combinations of hash functions, and hence better chance of having optimal fill factors of the BFs. However, a larger d implies larger forwarding tables that may not pay off the gains in false positive rates. Section 4.3 provides more experimental insights on these trade-offs.

3.2.3 Loop prevention

In some cases false positives can result in loops; for instance, consider the case where a zFilter encodes a forwarding path $A \rightarrow B \rightarrow C$, but, due to a false positive, the zFilter also matches with a separate link $C \rightarrow A$, which is used to forward packets from C to A. Without loop prevention, this will cause an endless loop of $A \rightarrow B \rightarrow C \rightarrow A$. Obviously, we can construct a set of links that may cause a loop and use the *pfr* method above to select only loopless candidate BFs. However, this may not guarantee loop free trees; for example, the network graph presentation may not have all links in it, e.g. due to slow reaction to network changes.

As an alternative solution, we start with each node knowing the neighboring nodes' *outgoing* Link ID and LITs towards the node itself. We call these as the *incoming* Link ID and LITs. Now, for each incoming packet, the node checks the

incoming LITs of its interfaces, except the one from where the packet arrives, and matches them to the zFilter in the packet. A match means that there is a possibility for a loop, and the node caches the packet's zFilter and the incoming Link ID for a short period of time. In case of a loop, the packet will return to this node over a different link than the cached one and the packet is simply dropped.

3.3 Stateful functionality

3.3.1 Virtual Links

So far, a number of LITs have been inserted into a zFilter, each LIT representing an individual physical link. However, in case of dense multicast trees, heavy traffic, and long-lived connections, it becomes more efficient to identify sets of individual links with a separate Link ID and a set of associated LITs. We call such sets of links for *virtual links.* A single virtual link can define an end-to-end path partly or completely. They are always unidirectional, but they may also encode multicast trees instead of simple paths.

A virtual link may be generated by the topology layer whenever it sees the need for such a link. The creation process consists of selecting the individual links over which the virtual link is created, assigning the link a new Link ID, and computing the LITs. Once the virtual link has been generated, the topology layer needs to communicate the Link ID, together with the LITs (or their hash indices) to the nodes residing on the virtual link.

Once the virtual link creation process is finished, we can use a LIT of this virtual link in any zFilter instead of including all the individual LITs into it. This reduces the probability of false positives when matching the zFilter on the path. On the other hand, adding forwarding table entries into nodes increases the sizes of the forwarding tables, but compared to the current situation with IP routers, the sizes of the forwarding tables still remain relatively small, unless a huge amount of virtual links needs to be added.

3.3.2 Fast recovery

Whenever a link or a node fails, all delivery trees flowing through the failed component break. In this section, we consider two approaches for fast re-routing; in particular, we examine the simple case where there is only one failed node(link) on any given path.

Our first approach is to replace a failed link with a functionally equivalent, preconfigured virtual link. We call this the *VLId-based recovery* approach. The idea is to have a separate virtual backup path built for each physical link ID, to be dynamically used in case of failure of a node determining that it no longer can send packets out through one of its interfaces. This virtual backup path has the same Link ID as the physical link it replaces, but is not active by default to avoid false forwarding.

The main advantage of this approach is that there is no need to change the zFilter in the packets; basically, it is enough that the node detecting a failure sends an activation message over the replacement route, activating the backup route for both the failed physical link and any virtual links flowing over the physical link, and then starts to forward the packets normally. When receiving the activation message, the nodes along the backup path reconfigure their forwarding tables, starting to forward packets along the replacement path. As a result, the packets flow, unmodified, over the replacement path.

Another approach is to have a separate Link ID (and LITs) for the replacement route. In this method, when a node detects a failure, it adds the appropriate LIT representing the replacement route into the zFilter in the packet. This method does not add any additional signaling or state to the forwarding nodes, but it increases the probability of false positives.

3.4 Edge or border filtering

Let us reconsider Fig. 2.1 on page 3. In the figure, the AS on the left and the core on the right each form separate subnetworks, using zFilters internally. As a result of the rendezvous systems asking the topology system to create a forwarding path, each of the topology modules compute a suitable zFilter, to be used internally. We present two alternative operations at the edge nodes, depending if we use label switching or recursive stacking for the zFilters.

In the case of *label switching*, as the AS topology module realises that the packets must continue to the core, it instructs the edge forwarding node to record a mapping from the zFilter to the RId. Correspondingly, the core topology module instructs the edge node to form a mapping from the RId to its zFilter. Note, that a single zFilter may be used to carry packets from multiple publications, each of having different RId and potentially a different set of subscribers. Thus, each of the zFilters may be mapped to multiple RIds on both directions.

Using the mappings, the edge node can now inspect incoming packets, filter out false positives, replace the incoming zFilter in the packet with an outgoing one, and pass the packet to the other network for delivery. In the case of connecting multiple domains, the edge network may forward multiple copies the packet, each having a different zFilter.

The main drawback is the huge amount of state required. Basically, there needs to be a table entry for each active RId, which becomes impractical towards the core. This method is mainly suitable for use at the edges of the network, where the order of active RIds is likely to be millions.

In the case of *recursive stacking*, the packet carries two zFilters, one for the inter-domain layer and the other one for intra-domain use.

The inter-domain zFilter contains only two links: one from the node closest to the publisher to the border node, and another from the border node to the node closest to the subscriber. These links are *virtual* in the sense that they cannot be used for forwarding through the networks but for mapping purposes.

The edge node removes the intra-domain zFilter from the packet and determines from the inter-domain zFilter the possible other edge nodes as well as candidate internal subscriber virtual links. In our example , no edge nodes exist in the zFilter and there would be only one entry for the local subscriber. The edge node conceptually contacts the local topology layer, which determines the proper intra-domain zFilter to be used for delivering the packet to the subscriber. In practice, the local zFilter can usually be pre-determined and simply looked up from a cache, or formed from a few cache entries by ORing together the corresponding zFilters.

Even in this case the border router needs quite a lot of state, as it needs to have a separate virtual link for each separate *subscriber set*. However, for typical unicast and sparse multicast cases it suffices to have a separate entry for each local edge node, which will can be scaled up to a few millions without compromising the performance.

Chapter 4

Evaluation

The main objective of this section is to explore the scalability limits of the zFilter forwarding approach. First, we study the false positive probability of the proposed Bloom filters for different design parameters and optimizations. Second, we estimate the amount of link IDs required to establish delivery trees in typical intra-domain AS topologies for different subscribers' density. Establishing a maximum false positive probability to achieve reasonable performance levels (e.g., 5% false positives), we will be able to draw some conclusions on the scalability of the zFilters without further extensions. Third, packet-level ns-3 simulations demonstrate the feasibility of LIPSIN and complement the previous analysis with actual false positives measurements and insights on the forwarding efficiency. Finally, we conclude the evaluation section with considerations on forwarding table sizes, security, and some final remarks.

4.1 False positive probability

As already described in Sec. 3.2, Link ID Tags are already in the form of m-bit bit vectors with k bits set to one and are added to a candidate BF_i through a simple OR operation of the LIT_i. Assuming that the hash functions to compute the LITs are uniform, it is easy to show [26, 17] that the *a priori false positive estimate* fpb is the expected false positive probability for the given set of parameters (m, n, k) *before* adding the set of elements:

$$fpb = \left[1 - \left(1 - \frac{1}{m}\right)^{k \cdot n}\right]^k \tag{4.1}$$

Setting the partial derivative of fpb with respect to k to zero, gives the number k that minimizes the false positive probability. This is attained when $k = log_2 \cdot \frac{m}{n}$, and is rounded to an integer that determines the optimal number of ones in the LIT representation.

Note that the definition of fpb does not involve knowing exactly how many bits are finally set to one in the BF. Therefore, a more accurate estimate can be

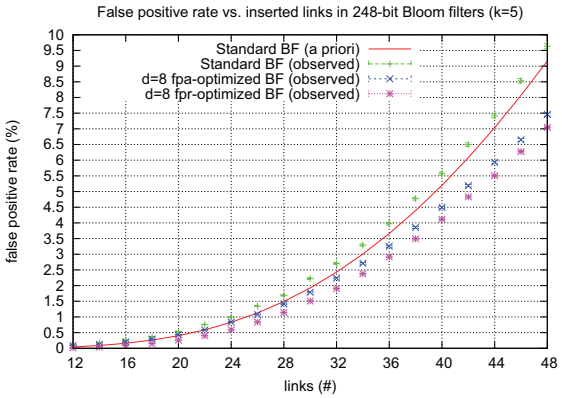


Figure 4.1: Having more choices when constructing BFs leads to improved BFs: $\approx 1\%$ better fpr in our region of interest (30-40 links or below 5% fpr).

given once the *fill factor* ρ of the filter is known; that is the factor of bits that are set to one after all the keys are hashed. We can define the *posterior false positive estimate* fpa as the expected false positive estimate *after* hashing the elements:

$$fpa = \rho^k \tag{4.2}$$

In practice, the prior and the posterior estimates are usually very close due to their concentration around the mean. In our small size bit vector scenario, every bit counts and fewer false positives will be key to the performance of our system. The *fpa optimized* BF selection introduced in Sec. 3.2.2, is based on finding the set of LITs with the smallest predicted fpa.

Finally, the *observed false positive probability* is the actual *false positive rate* (fpr) that is observed when a set of queries are made on the BF:

$$fpr = \frac{\text{Observed false positives}}{\text{Tested elements}} \tag{4.3}$$

Note that the fpr is an experimental quantity computed via simulation or actual measurements and not a theoretical estimate. The minimum observed fpr of the d candidate BFs provides a reference lower bound on the actual false positive rate that we can achieve for a specific BF design (e.g., $m = 248bits, k_{opt}$).

Recall that a *fpr optimized* selection of the LIT-based zFilters can be done when the topology module bases the selection of the zFilter on the lowest observed fpr after checking for false positives against a set of LITs expected in the network path. Alternatively the sending node is provided with all candidate zFilters and is able to use the one that shows the actual best fpr based on some type of network feedback.

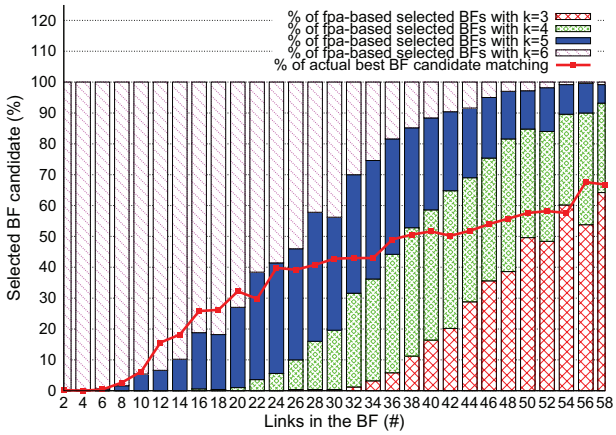
4.1.1 System parameters and experiment setup

We carry a set of simulations to get practical knowledge on the actual performance and the design space of a the BF-based data structure for compact representation of Link IDs. In order to have a small overhead in the packet headers we set the size of the zFilter to 256 bits(248-bit BF + 8 reserved bits), which is also a fair comparison value to the source and destination to pair of IPv6 address fields ($2 \cdot 128bits$).

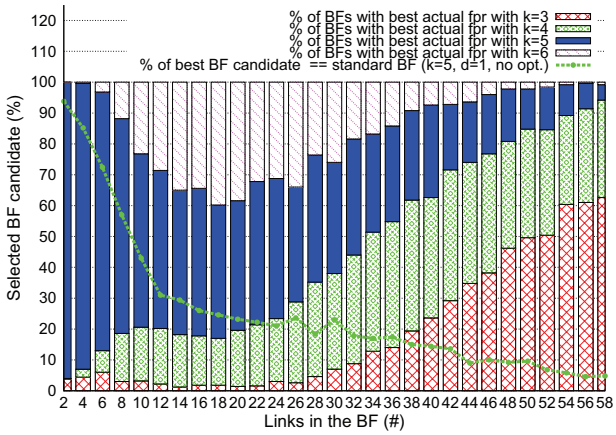
Our goal is to evaluate how many links can be inserted in the zFilter for a certain upper bound false positive probability $P_{max} = 5\%$. Recall that false positives are traduced in data packets delivered over neighbouring link IDs not included in the zFilter. However, the probability that a false positive propagates over further links is multiplicative at each hop by P_{max}, given the large space of link IDs ($m!/(m-k)!$). Recall that in our data-centric inter-networking scenario with massive *opportunistic caching*, such false positives are even less harmful; if there is space available the data may be cached at these false locations thus making the network usage more efficient.

We simulated many system parameters ($k \in [4, 5, 6]$,$d \in [2, 4, 8, 16, 32]$,) and for pairwise combinations we run 500 tests counting for false positives by querying for the presence of 1000 link IDs of a randomly generated disjoint T_{set}. Figure 4.1 shows the false positive ratio results for $d = 8$ degrees of choice, $k = 5$ for the standard BF and a $k_{dist} = \{3, 3, 4, 4, 5, 5, 6, 6\}$ to build the 8 candidate BFs. The points in the graph are the mean values of the experiments and the error bars denote the standard error deviation. As expected, the best performing BF (*fpr optimization*) is the one we can select when T_{set} is known. The BF selected after observing its fill factor (*fpa optimization*) shows also better performance than a standard BF with no choice (d=1). After inserting 32 elements, a *fpa-optimized* BF reduces on average the *fpr* in around 0.5% (from 2.75% to 2.25%) and the best performing candidate brings the fpr another 0.5% down to under 1.8%. The performance gains increase with number of elements inserted in the filter. The best BF candidate supports up to 43 elements before reaching the 5% false positive rate. In comparison, the standard BF already shows this fpr after having inserted around 38 links. These extra 5 links means that more subscribers can be reached without loosing forwarding efficiency.

Figure 4.2 illustrates the performance gains of having 8 choices when constructing BFs. The percentage boxes illustrate the distribution of k among the experiments when the BF is selected following the lower fpa criteria (see eq. 4.2) and when the actual best performing BFs are considered. The percentage curve highlights how often the used BF performed the best among the 8 candidates in the standard BF scenario (figure 4.2(b)) and when the candidate BF was selected after computing the lowest fpa (figure 4.2(a)). One can also appreciate the adaptive selection characteristic of the k_{opt} depending on the *fill factor*. With ≈ 32 inserted elements, almost every second time the best performing BF was selected just looking at the *fill factor*.



(a) *fpa*-based Bloom filter selection



(b) *fpr*-based Bloom filter selection

Figure 4.2: Performance gains of d-LIT constructed BFs and the distribution of k_{opt} .

Table 4.1: Graph characterization of a subset of router-level AS topologies used in the experiments.

AS	1221	1755	3257	3967	6461	*TA2*	*Cost*
Nodes (#)	104	87	161	79	138	65	37
Links (#)	151	161	328	147	372	108	57
Diameter	8	11	10	10	8	8	8
Radius	4	6	5	6	4	5	5
Max. degr.	18	11	29	12	20	10	5
Avg. degr.	2	4	3	3	5	3	3

4.2 Data set

Choosing realistic AS topologies to validate new forwarding algorithms is a challenging task, mainly due to the inaccuracies of the available data sets. One set of data we used are 6 intra-domain AS topologies from Rocketfuel [2]. Though not completely accurate, they are a common (best) practice to approximate new forwarding schemes to real world scenarios. A second useful data set of "realistic" router-level topologies is SNDlib [28], where contributors worldwide have published their network maps to the research community. Recent studies [34] have pointed out the limitations of the Rocketfuel data, suggesting that the number of actual physical routing elements is less than the inferred one by their measurement technique. We observed that Rocketfuel graphs are more complex than the largest topologies in SNDlib. Four our purposes this fact (potential inaccuracy) is actually good to stress our forwarding mechanisms.

We are also aware that switching elements are not present in topology maps and could have an impact on the number of link IDs required as the number of network element hops increase. However, we believe that routers should be regarded as the significant networking units on which link IDs are defined, letting layer 2 elements hidden from the topology function.

4.2.1 User density and number of links

We carried a series of experiment to estimate the number of links required to establish delivery trees for different user densities. The experimental results for the number of links required averaged over 6 different topologies are given in Fig. 4.3. In the center figure, we show the case when one additional link ID is required for the hop between the user and the first forwarding node. The figure on the right presents the case where the leaf nodes represent rendezvous points of the subscribers and therefore maintainance of forwarding state for their attached users can be assumed. Finally, the left figure shows the experiment snapshot from the largest router-level topology (AS 1239 - Sprint).

In the previous false positive probability analysis, we observed that 40 link IDs is the maximum capacity of the zFilters to maintain the bandwidth waste

under control. For 40 link IDs, we can conclude that up to 10 users can be reached over typical AS intra-domain topologies, or up to 20 users when some state is maintained in the forwarding node where the users are attached.

4.3 Packet-level Simulation

Another important part of our evaluation studies was to implement our line speed forwarding scheme in ns-3 [18]. First, it serves as a quick proof-of-concept implementation of the LIPSIN idea. Second, it investigates its behaviour on different topologies and gives a detailed picture about the effect of the different parameters.

We implemented a zFilter-based forwarding layer over an existing Layer2 implementation[1]. Over the forwarding layer, we implemented a simplified pub/sub-based internetworking-layer to send publication data to a given rendezvous identifier. A separate topology module has all the information about the network topology (user locations, Link IDs and LITs) and can thereby construct the zFilter by defining a tree using the shortest paths between the publisher and each subscriber. We also implemented a rough loop prevention method that solves the problem of infinite loops by caching packet headers in the forwarding elements that may potentially cause loops if a forwarded packet is received again over another input interface.

In our experiments we investigated the different settings of the number of forwarding tables (d), as well as the number of users interested in a publication (n). Furthermore, we investigated the effect of different LIT-sets for the nodes(constant k, variable k distribution), and used different BF selection algorithms (fpr/fpa optimization). We carried out our tests on the well-referenced topologies (data set from Sec. 4.2) and performed 500 runs for each parameter setting.

We can define the *forwarding efficiency* as the ratio of *inserted links* (programmed links in the zFilter) to *forwarded links* (actual links the packet traversed through). This metric gives a notion of the bandwidth overhead due to false-positive-based forwarding decisions.

We present the essence of our simulation results on Tables 4.2 and 4.3. Table 4.2 contains results using the *fpa* selection criteria with the above mentioned distribution of k around 5. Besides the mean values, we also counted the 95% percentiles to observe the sensibility of the results. We found that we have adequate performance in all of the topologies, if we consider one publisher and 23 subscribers ($\approx$ 32 links). When increasing the number of subscribers the results tend to be more topology-dependent. AS3257 presents the poorest performance, which can be expected due to its graph characteristics (many nodes, high diameter and radius).

Table 4.3 sheds light on the fpa/fpr-based BF selection algorithms. From the results we discovered an interesting relation between the distribution of k

[1]This was purely an early implementation choice. Ethernet or other Layer 2 MAC is not required.

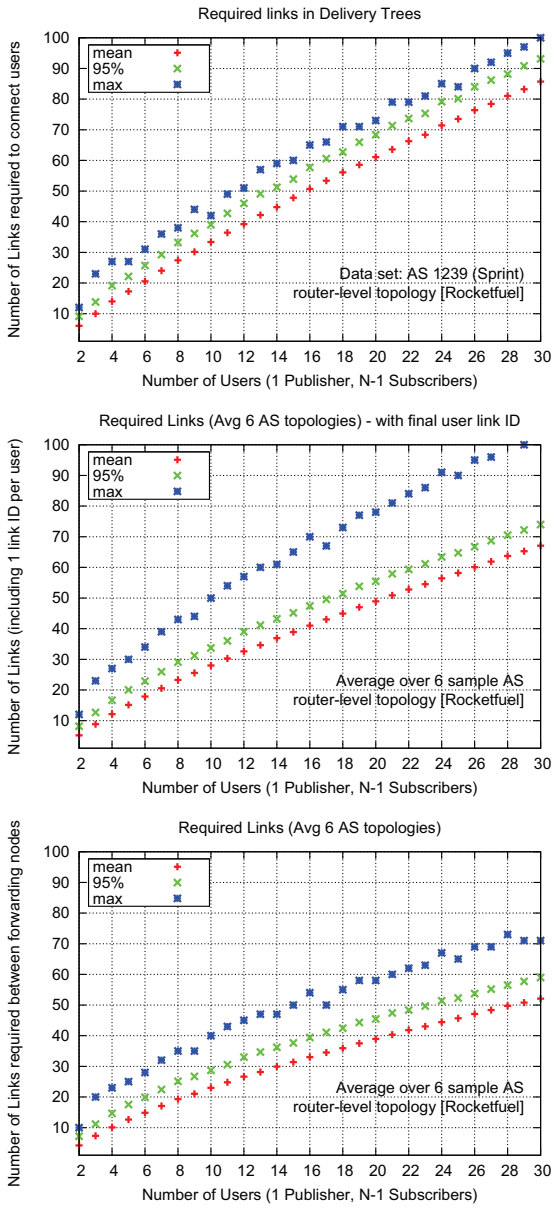


Figure 4.3: Simulation results on the number of links required to establish delivery paths for different user densities over Rocketfuel topologies.

Table 4.2: ns-3 simulation results (d=8, multi-k=5).

Users	AS	Links (#)		Efic. (%)		fpr (%)	
		mean	95%	mean	5%	mean	95%
4	*TA2*	8.6	12.7	99.92	100	0.02	0
	1221	9.7	13.6	98.08	88.89	0.37	2.13
	3257	9.6	13.5	99.83	100	0.02	0
8	*TA2*	15.6	20.0	99.6	94.12	0.2	1.59
	1221	16.8	21.3	97.78	90.89	0.54	2.02
	3257	17.9	22.9	98.95	91.3	0.28	1.25
16	*TA2*	25.7	30.9	97.92	91.67	0.83	2.67
	1221	27.4	31.0	95.51	88.22	1.28	3.17
	3257	31.3	36.7	92.37	79.58	1.76	3.86
24	*TA2*	34.1	38.8	95.2	87.18	1.95	4.63
	1221	36.1	41.0	92.06	83.33	2.65	5.19
	3257	42.2	48.1	82.27	67.69	4.17	6.96
32	*TA2*	41.4	46.0	92.04	84.31	3.46	6.46
	1221	44.0	48.3	88.22	78.95	4.32	7.45
	3257	52.2	57.9	71.47	59.34	7.3	10.41

and the optimization strategies. We observed that in our region of interest the $k_{const} = 5$ performs better than the variable k distribution. As expected, *fpr-optimization* successfully reduces the false positive ratio, and outperforms the standard approach 2 or 3 times in the scenarios with 16 users. The gain of using *fpa-optimization* instead of the standard algorithm is clear, although not as significant as in the previous case. These improvements can be also observed in the sample results of AS6161 on Fig. 4.4.

4.4 Forwarding table sizes

Each forwarding node maintains d forwarding tables with one entry for every active Link ID and one associated outport value. In the basic system design, the memory requirements for the LIT-based forwarding tables are:

$$FTmem = d \cdot \#Links \cdot [size(LIT) + size(PortOut)] \tag{4.4}$$

Considering $d = 8$, 128 link IDs (physical & virtual), 248-bit LITs and 8 bits for the outport, the total memory required for the forwarding table would be $\approx 256Kbit$.

The d-LIT enhancement comes at a price of factor d in fast memory sizes. Although this memory size is small due to the source-based routing approach, there is room to design a more efficient forwarding table. Since LIT entries will only contain k bits set to one, a *sparse representation* would only require to store the k positions of the bits to be ANDed in the test membership on

Table 4.3: ns-3 simulation results comparing mean fpr values for different configurations.

Users	AS	links	fpa-opt. (fpr %)		fpr-opt. (fpr %)		Stdrd.
		mean	k_{const}	k_{distr}	k_{const}	k_{distr}	$k = 5$
8	*TA2*	15.6	0.12	0.2	0	0	0.18
	1221	16.83	0.44	0.54	0.26	0.26	0.55
	3967	17.72	0.28	0.33	0.03	0.03	0.48
	6461	17.18	0.32	0.39	0.06	0.07	0.36
16	*TA2*	25.7	0.54	0.83	0.01	0.03	0.8
	1221	27.37	1.17	1.28	0.36	0.45	1.57
	3967	29.04	1.13	1.29	0.24	0.34	1.48
	6461	29.31	1.55	1.57	0.71	0.83	1.89
24	*TA2*	34.1	1.65	1.95	0.38	0.58	2.03
	1221	36.14	2.48	2.65	1.21	1.33	3.55
	3967	37.65	2.55	2.78	1.31	1.48	3.22
	6461	39.60	3.72	3.79	2.81	2.86	4.86

packet arrival. Thereby, the size of each LIT entry is nor to only $k \cdot log_2(LIT)$, the total forwarding table size is thereby reduced to $48Kbit$. Furthermore, if we used a *segmented hash* approach where each hash to generate the LIT is limited to a region of $\frac{m}{k}$ bits, each LIT entry would require just $k \cdot log_2(\frac{LIT}{k})$ bits, which in our example is traduced to a total forwarding table size of $36Kbit$. Our first evaluations on a FPGA implementation suggest that the forwarding performance is not penalized when adopting the sparse representation of the LIT entries.

4.5 Security

The probabilistic nature of Bloom filters inherently provides the basis for most of our security features. Furthermore, due to their construction, zFilters are location specific, making it unlikely that any given zFilter could induce any usable traffic if used outside of its intended links. Without knowledge of the actual network graph, including the active Link IDs and LITs, it is unpractical trying to guess a zFilter that would reach any set of host(s).

Consider a simple *zFilter contamination attack*, where a sending node can try to use a BF with a large amount of 1's (or even containing only 1's) that would result in the packet being broadcasted at every forwarding hop due to matches over every link. A simple countermeasure, also observed in [40], is to limit the amount of 1's in a BF, e.g., to 50–70% of the bits. This can be easily implemented in hardware without causing any additional delay. As a result, on the average any randomly generated zFilter will match outgoing links only at the false positive rate, which could be tuned in our system, e.g. to the level of 5%.

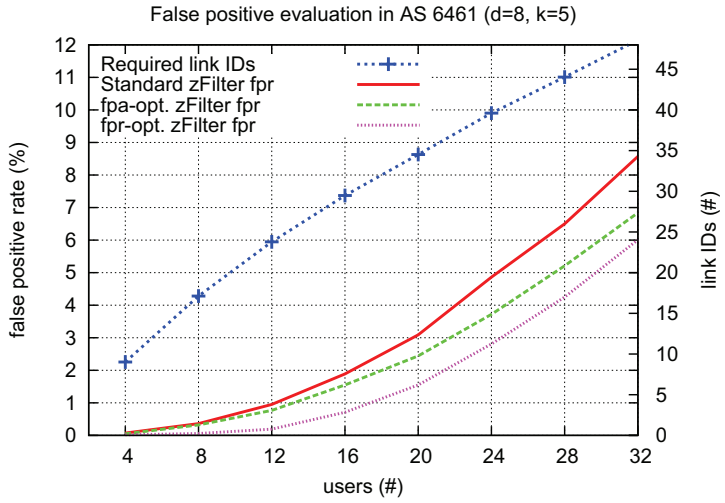


Figure 4.4: ns-3 simulation results for AS 6461.

In a more advanced attack, basically combining a *LIT learning attack* and a *zFilter re-use attack*, an attacker may first attempt to figure out the LITs of the links nearby it. For example, an attacker may attempt to lure lots of subscribers from different parts of the network, thereby allowing it to learn a number of valid zFilters originating at it and learning probable local LIT or combination of LITs for the next few links by using AND for the received LITs.

The presented attack requires quite a lot of work and there are few direct countermeasures. First, using more parallel LITs on links close to the data sources makes collection more expensive. Second, uplink Link IDs can be changed relatively often, forcing the attacker to restart. Third, varying the selection of the Bloom filter (as described in Section 3.2.2), and using those that have more ones, increases the probability that the attacker gets a too full zFilter.

More generally, we can avoid many of the known, and probably a number of still unknown attacks, by slowly changing the Link IDs over time. Thus, there would not exist fixed zFilters that can be used for data between two points at any time. The caveat would be that the connection between the hosts has to be re-calculated once in a while. For actual data delivery, we expect higher level virtual trees to be used deeper in the network which can be easily made short-lived and thereby become unusable after the timer expires.

Finally, consider the situation where an attacker has succesfully launched a *DDoS attack*. That is, there are lots of packets that do not belong to any active subscription arriving at a victim. Initially, the victim can easily quench the packet stream by changing its downlink LITs. After that, any upstream node close to the victim can AND together the zFilters from the unwanted packets, thereby determining if there are any LIT or combination LITs indicating a com-

mon data path. If there are any such links, the traffic can be further quenched by requesting LIT changes also on those links. The above mentioned periodic Link ID changes can then be used to extinguish the attack.

Looking at our system from a higher perspective, forwarding on a hash-based data structure, such as proposed in our system, complies with several of the steps recommended by Handley and Greenhalgh [16]. By separating the roles of publishers and subscribers, basically any addressing scheme can be used in the domain where the communicating entities reside. The use of periodically changing link IDs avoids requiring global client identifiers.

In other words, the pub/sub paradigm inherently builds a powerful defense architecture empowering the receivers. In our design, no forwarding state is created if there aren't a sufficient number of subscribers that have explicitly indicated their interest in data delivery. We thereby avoid the typical problems of multicast routers maintaining state of unnecessary multicast groups, e.g., an attacker joining many low-rate multicast groups.

Our architecture has the potential to drastically reduce the amount of packet-level or DDoS-like unwanted traffic by making it more costly for the sender. Additionally, we are also considering other security mechanisms, such as using cryptographic rendezvous identifiers or using hash chains or trees to verify the integrity of packet sequences. However, the details fall beyond the scope of this paper.

4.6 Final remarks

The results of our evaluation work suggest that we can make 40 Link IDs easily available for stateless unicast traffic, covering sparse multicast, too, in small world topologies. However, dense multicast still requires explicit state.

The small memory requirements of the forwarding tables and the line speed matching of the outport LITs indicate that we could easily have a few hundreds of Link IDs defined per physical interface. We believe that with such an amount of Link IDs we can establish even more stable and larger (multi-hop) network paths via virtual links, reaching thereby larger groups of users, perhaps up to a few tens, without compromising the fast forwarding efficiency. The exact trade-offs of leveraging the (physical) Link ID based forwarding with a certain amount of virtual links has been left for future work.

Another direction towards scalability is to use a second BF, when higher capacity zFilters are required. LITs are inserted into either one of the BF. The forwarding nodes need to check for the presence of LinkIDs in both BFs. This "double query" comes at the cost of increased zFilter construction time and false positives rate.

With the diameter[2] of the Internet appearing to be established around 4 [3] and the assumption that the current Internet hass small world topology [21], on

[2] `http://thyme.apnic.net/current/data-summary` accessed in Nov. 2008: BGP routing report *average AS path length visible in the Internet* = 3.6

average just 4 domain level hops are involved and less than 10 routers in typical end-to-end connections.

We believe that our design, with the briefly sketched edge and border operations involving zFilter swapping and stacking, should easily scale to an Internet-wide system. However, exact estimates for the memory requirements at the edge routers is left for future work.

Chapter 5

Implementation and future work

We have implemented two partial prototypes of the system, a FreeBSD-based end-node prototype and a Stanford NetFPGA based forwarding node prototype. The structure of the end-node prototype is depicted in Fig. 5.1. Each publication is represented as a virtual memory area, depicted with a "P". The PSIRP I/O implements the needed system calls for creating new publications (reserving memory areas), publishing, and subscribing. When allocating memory for a publication, the pager is set to be a vnode pager, and the backing file to be in the Filesystem in Userspace (FUSE)[1]. Hence, each publication is backed up by a virtual file, located in a separate virtual file system running under FUSE.

In the current implementation, the actual contents of the file are kept only in the memory. The physical memory pages are mapped onto the virtual memory of the `psirpd` daemon, the publisher (if any), and the subscribers (if any). The mapping is copy-on-write, causing the kernel to allocate new pages if the publisher or any subscriber modify pages that contain published data. Whenever the publisher wants to publish a new version, the system creates a new snapshot, basically marking any changed pages again as copy-on-write. In this way, there is never need to copy data during the publication time and the system can easily keep track of changed pages, eventually leading to more efficient communications.

On a parallel track, we have implemented an early prototype of the forwarding node using Stanford NetFPGA [25]. Starting from the reference switch implementation, we removed most of the unnecessary code in the reference pipeline and replaced it with a simple zFilter switch. At this point, we have a hand coded forwarding table, with eight parallel entries and support for only one LIT ($d = 1$). The present implementation takes less than 100 lines of Verilog code.

In order to estimate the packet handling delay, we implemented a simple measurement tool on FreeBSD, using a PC with two separate gigabit Ethernet NICs. The tool sends a timestamped packet out through one NIC and waits

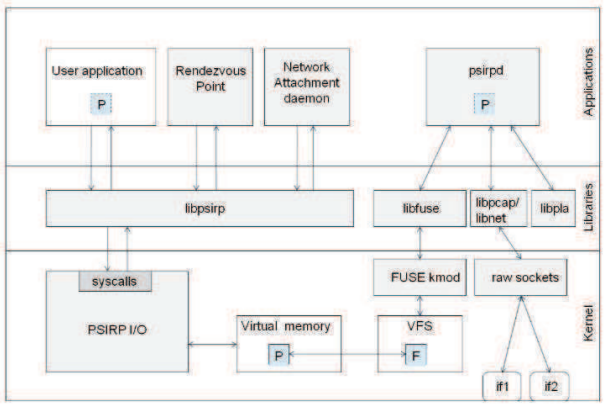


Figure 5.1: Overall view of the FreeBSD Prototype

for the packet to be returned through the other one. We measured the average packet delay over 10000 packets, once with the NetFPGA on the path and another time using a loopback cable. The results are given in Table 5.1 and indicate that the switching delay on the NetFPGA is negligible. We are 95% confident that adding the NetFPGA implementation increases the latency only by 1.01-3.95 μs.

As the next steps, we will enhance the implementation to support dynamic and larger forwarding tables, virtual links, and parallel LITs. Once done, we will measure the performance in a three NetFPGA network, and compare that with the NetFPGA reference Ethernet switch implementation.

We will integrate our end-node implementation with the rendezvous system prototype, build by another group within our project. The resulting system, while still lacking a proper topology system, will be functionally complete and allow us to test pub/sub based applications in small network settings. We have started to implement a proper topology management module, and we will also enhance the system otherwise.

On a wider scope, we will continue with the evaluation when using vir-

Table 5.1: Preliminary NetFPGA implementation performance results. Processing time in μs

	NetFPGA	loopback
Avg.	91.54	89.06
Samples	9969	9969
SD	37.46	37.27
SE	0.38	0.37
95% CI	(90.8, 92.27)	(88.33, 89.79)

tual Link ID to explore the signaling and state trade-offs of this enhancements towards global scalability. Simulation work is going in the field of state-full operations between network domains to implement the edge or border filtering approaches. Moreover, we have not exhausted all dimensions of the zFilter design parameters.

Chapter 6

Related Work

Related work falls into various categories, which we briefly discuss in the following paragraphs.

Pub/sub and multicast: To improve the network layer performance in topic-based pub/sub systems, one approach is mapping pub/sub topics to IP multicast groups, so data can be directly sent to subscribers with a single message on the wire. However, this method, though network efficient, does not help to solve the scalability issues of IP multicast. In case of many concurrent active receiver groups, the routers are forced to maintain huge forwarding states due to the lack of aggregation.

The authors of [31] revisit the case of IP multicast and propose Bloom filters to aggregate active multicast groups inside a domain, piggybacking this information in BGP updates. Their IP multicast design also includes a Bloom-filter-based shim header in packets to represent AS-level paths of multicast packets. Our work handles links as more general destination information than IP prefixes and explores more dimensions of the pub/sub forwarding/routing space [42].

Fast forwarding: GMPLS[27] is being extensively used to provide fast forwarding. By separating control and forwarding plane, it introduces more flexibility and important performance gains with the pure hardware fast label switching. However, it does not suit topic-based pub/sub because of the limited label space and no label aggregation capabilities. Some approaches have tried to address this problem by the help of tunneling and group aggregation. Our approach also benefits from the separation of these different planes and is extremely hardware-friendly promising a line speed forwarding layer.

Source routing: The simplest form of source routing [36], is concatenating the forwarding nodes' network identifiers on the path between senders and receivers. Our zFilters address one of the main caveats of source routing, namely the overhead of having to carry all the routing information in the packet. Moreover, our approach does not reveal these identifiers to the sending nodes, nor the sequence or amount of hops involved.

In the PoMo architecture [29], the authors suggest a routing/forwarding solution that trades overdeliveries for reduced state and reduced dependence of

node network locators. In [9], they propose a networking approach with link identities having a pivotal role.

In line with our LIPSIN forwarding scheme, the BANANAS framework [22] for explicit and multipath routing in the Internet is based on the observation that a path can be encoded as a short hash (PathID) of a sequence of globally known identifiers. While the focus of BANANAS relies on host-centric multipath communications, our design is centered around non-global, opaque Link IDs and its compact representation for efficient information-oriented multicast communication. Many of the schemes developed in [22] for route computation and deployability over existing connectionless routing protocols (e.g., OSPF and BGP extensions) may be borrowed to be used for LIPSIN over legacy networks.

Networking applications of Bloom filters: Bloom filters [6] have attracted considerable attention from a networking application perspective. Today BFs are one of the most popular data structures, spanning their application domain [7] from distributed environments (e.g., distributed caching) to hardware implementations, becoming a daily aid for network processing applications (e.g., IP forwarding, security, etc.).

In Icarus [35], a method to detect possible loops in the routing by using in-packet BF is presented. The security model proposed in [40] shares also the principle of carrying a small BF in packets to provide data path authentication.

In resource location applications [7], BFs have been used to bias random walks in P2P networks. In content-based pub/sub systems [20], summarized subscriptions are created using BFs and used for event routing purposes and general.

We can expect more and more applications Bloom filters and its derivatives in future internetworking proposals relying in hash-based flat identifiers [13].

Improved Bloom filters: Prior work include the Power of Two Choices filter [26] and the Partitioned Hashing [17], which combine the power of choices at hashing time to improve the performance of BFs. Besides sharing the same motivation, our fast forwarding application posses some extra challenges and opportunities for optimization.

False positives are reduced in [17] by a careful choice of the group of hash functions that are well-matched to the input elements. However, this scheme is not practical in a distributed highly dynamic environments. The main idea of [26], is also to choose one of c sets of hash functions so that the number of ones in the filter is reduced. In our system we have the information of which group of hash functions was used (d value) transmitted to the test-membership issuer (forwarding nodes), avoiding thereby the caveat of checking multiple sets. On the other hand, we need to stick to one set of hash functions for all elements in the BF, whereas in [26] the optimal group of hash functions can be chosen on an element base. However, in our small BF scenario we are able to select an optimal BF after evaluating all d candidates. We recognize that there is still room for optimization and other probabilistic compact representations and design parameters could be used as long as simplicity is not sacrificed.

Chapter 7

Conclusions

In this paper, we presented the design, analysis, and partial implementations for a recursive multicast-based forwarding layer, suitable for implementing very large topic-based publish/subscribe networks. In a two-layer model, similar to the present Internet inter-domain and intra-domain routing model, we are using a slightly larger packet header (64 bytes) than the current Internet (IPv4's 20 bytes and IPv6's 40 bytes). While the intra-domain forwarding is straightforward with our design, in inter-domain case we need to maintain some additional information about neighbour edge nodes at each of the domain's edge nodes.

Our analysis and simulations indicate that we can easily support sparse multicast trees up to approximately 20 receivers per domain. For denser multicast trees, one can either install a separate forwarding table entry for each group, or implement partial trees that can be easily shared between groups. By installing state in the network as and where required depending on the usage dynamics, we expect that applying the recursive operations it should be possible to scale our design into Internet-wide scales.

As our solution is based on partial source routing, it is potentially vulnerable to the well-known problems known from IP source routing. While the actual discussions are lengthy, we can summarise the security discussion by noting that the Bloom filters are opaque and location dependent, making it hard to construct targeted packets. We deal with the looping and packet amplification problems by using preventive actions as well as detection in the network. Link and node failures are handled using preconfigured hot spares.

While we only briefly defined the requirements for the control plane, in the companion paper [42] we discuss the problems of multicast tree construction and maintenance in more detail, and and show how the problem can be divided into a number of subproblems, each separately solvable.

We have implemented two early prototypes of our system, one running on FreeBSD (end node) and another based on the Stanford NetFPGA (forwarding node). In the FreeBSD implementation, we have implemented an publish/subscribe API using the virtual memory, resulting in zero-copy operations. With the present early NetFPGA-based implementation, forwarding

single packets takes only about 2.5 microseconds compared to a plain loopback cable, suggesting a very fast packet forwarding plane.

Acknowledgments

We want to thank Sasu Tarkoma and Mikko Sarela for their valuable comments on the various versions of the paper, Jukka Ylitalo and Jimmy Kjallman for implementing the measurement tool and working together with Kristian Slavov and Teemu Rinta-Aho on the FreeBSD implementation, and Jari Keinanen for the contribution to the NetFPGA implementation.

Bibliography

[1] Fuse: File system in userspace. http://fuse.sourceforge.net/.

[2] Rocketfuel isp topology data. http://www.cs.washington.edu/research/networking/rocketfuel/maps/weights-dist.tar.gz.

[3] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (aip). In *ACM SIGCOMM*, 2008.

[4] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the internet. In *SIGCOMM '04: 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 343–352, New York, NY, USA, 2004. ACM.

[5] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM CCR*, 19(2):32–48, 1989.

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

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

[8] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica. Rofl: routing on flat labels. *SIGCOMM Comput. Commun. Rev.*, 36(4):363–374, 2006.

[9] K. L. Calvert, J. Griffioen, and L. Poutievski. Separating Routing and Forwarding: A Clean-Slate Network Layer Design. In *In proceedings of the Broadnets 2007 Conference*, September 2007.

[10] J. Day. *Patterns in Network Architecture: A Return to Fundamentals*. Prentice Hall, 2008.

[11] M. Demmer, K. Fall, T. Koponen, and S. Shenker. Towards a modern communications api. In *Proceedings of HotNets-VI*, November 2007.

[12] D. (edit.). Conceptual architecture of PSIRP including subcomponent descriptions. Deliverable d2.2, PSIRP project, Aug. 2008.

[13] C. Esteve, F. Verdi, and M. Magalhaes. Towards a new generation of information-oriented internetworking architectures. In *First Workshop on Re-Architecting the Internet*, Madrid, Spain, 12 2008.

[14] D. Estrin, Y. Rekhter, and S. Hotz. Scalable inter-domain routing architecture. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 40–52, New York, NY, USA, 1992. ACM.

[15] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[16] M. Handley and A. Greenhalgh. Steps towards a dos-resistant internet architecture. In *FDNA '04: ACM SIGCOMM workshop on Future directions in network architecture*, pages 49–56. ACM, 2004.

[17] F. Hao, M. Kodialam, and T. V. Lakshman. Building high accuracy bloom filters using partitioned hashing. In *SIGMETRICS '07*, pages 277–288. ACM, 2007.

[18] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. B. Kopena. Network simulations with the ns-3 simulator. SIGCOMM'08 Demos, August 2008. Code available: http://www.nsnam.org/releases/ns-3.1.tar.bz2.

[19] V. Jacobson, M. Mosko, D. Smetters, and J. J. Garcia-Luna-Aceves. Content-centric networking: Whitepaper describing future assurable global networks. Response to DARPA RFI SN07-12, 2007.

[20] Z. Jerzak and C. Fetzer. Bloom filter based routing for content-based publish/subscribe. In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, pages 71–81, New York, NY, USA, 2008. ACM.

[21] S. Jin and A. Bestavros. Small-world characteristics of internet topologies and implications on multicast scaling. *Comput. Netw.*, 50(5):648–666, 2006.

[22] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi. Bananas: an evolutionary framework for explicit and multipath routing in the internet. *SIGCOMM Comput. Commun. Rev.*, 33(4):277–288, 2003.

[23] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: building a better bloom filter. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 456–467, London, UK, 2006. Springer-Verlag.

[24] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM '07*, pages 181–192, New York, NY, USA, 2007. ACM.

[25] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 160–161, Washington, DC, USA, 2007. IEEE Computer Society.

[26] S. Lumetta and M. Mitzenmacher. Using the power of two choices to improve bloom filters. Accepted to Internet Mathematics. Preprint version available at http://www.eecs.harvard.edu/m̃ichaelm.

[27] E. Mannie. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. RFC 3945, Oct. 2004.

[28] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007. http://sndlib.zib.de.

[29] L. B. Poutievski, K. L. Calvert, and J. N. Griffioen. Routing and forwarding with flexible addressing. *Journal Of Communication and Networks*, 9:383–393, December 2007.

[30] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma. Incentive-compatible caching and peering in data-oriented networks. In *ReArch'08 - Re-Architecting the Internet*, Dec. 2008. Submitted.

[31] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting ip multicast. In *Proceedings of ACM SIGCOMM'06*, Pisa, Italy, Sept. 2006.

[32] M. Särelä, T. Rinta-aho, and S. Tarkoma. Rtfm: Publish/subscribe internetworking architecture. ICT Mobile Summit, Stockholm., June 2008.

[33] J. Scott, J. Crowcroft, P. Hui, and C. Diot. Haggle: a networking architecture designed around mobile users. In *3rd Annual Conference on Wireless On-demand Network Systems and Services*, pages 78–86. IFIP, January 2006.

[34] R. Sherwood, A. Bender, and N. Spring. Discarte: a disjunctive internet cartographer. *SIGCOMM Comput. Commun. Rev.*, 38(4):303–314, 2008.

[35] A. C. Snoeren. Hash-based ip traceback. In *SIGCOMM '01*, pages 3–14, New York, NY, USA, 2001. ACM.

[36] C. A. Sunshine. Source routing in computer networks. *SIGCOMM Comput. Commun. Rev.*, 7(1):29–33, 1977.

[37] S. Tarkoma, D. Trossen, and M. Särelä. Black boxed rendezvous based networking. In *ACM MobiArch '08*, 2008.

[38] J. Touch. Those pesky nats [network address translators]. *Internet Computing, IEEE*, 6(4):96–, Jul/Aug 2002.

[39] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *OSDI'04*, pages 15–15, Berkeley, CA, USA, 2004. USENIX Association.

[40] T. Wolf. A credential-based data path architecture for assurable global networking. In *Proc. of IEEE MILCOM*, Orlando, FL, October 2007.

[41] X. Yang, G. Tsudik, and X. Liu. A technical approach to net neutrality. In *Fifth Workshop on Hot Topics in Networks (HotNets-V)*, Nov. 2006.

[42] A. Zahemszky, A. Csaszar, P. Nikander, and C. Esteve. Exploring the pubsub routing/forwarding space. In *International Workshop on the Network of the Future 2009*, 2009. To be submitted.