LIPSIN: Line Speed Publish/Subscribe Inter-Networking

Petri Jokela[1], András Zahemszky[1], Christian Esteve Rothenberg[2],
Somaya Arianfar[1], and Pekka Nikander[1]
[1] Ericsson Research, NomadicLab, Finland
{petri.jokela, andras.zahemszky, somaya.arianfar, pekka.nikander}@ericsson.com
[2] University of Campinas (UNICAMP), Brazil
chesteve@dca.fee.unicamp.br

ABSTRACT

A large fraction of today's Internet applications are internally publish/subscribe in nature; the current architecture makes it cumbersome and inept to support them. In essence, supporting efficient publish/subscribe requires data-oriented naming, efficient multicast, and in-network caching. Deployment of native IP-based multicast has failed, and overlay-based multicast systems are inherently inefficient. We surmise that scalable and efficient publish/subscribe will require substantial architectural changes, such as moving from endpoint-oriented systems to information-centric architectures.

In this paper, we propose a novel multicast forwarding fabric, suitable for large-scale topic-based publish/subscribe. Due to very simple forwarding decisions and small forwarding tables, the fabric may be more energy efficient than the currently used ones. To understand the limitations and potential, we provide efficiency and scalability analysis via simulations and early measurements from our two implementations. We show that the system scales up to metropolitan WAN sizes, and we discuss how to interconnect separate networks.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.6 [**Computer-Communication Networks**]: Internetworking

General Terms

Design

Keywords

Bloom filters, publish/subscribe, multicast, forwarding

1. INTRODUCTION

Many networking applications are internally publish/ subscribe in nature [8]; the actual acts of information creation and consumption are decoupled in time and/or space, and often there are multiple simultaneous receivers. For example, RSS feeds, instant messaging, presence services, many typical web site designs, and most middleware systems are either based on a publish/subscribe-like information paradigm or internally implement a publish/subscribe system.

In general, publish/subscribe [15] is a data dissemination method which provides asynchrony between data producers and consumers. Key ingredients include handling data itself as a first class citizen at the naming level, efficient caching to loosen the coupling between producers and consumers in the time dimension, and multicast to efficiently disseminate new data, including both user-published data and system-internal metadata. In addition to pure pub/sub applications, peer-to-peer storage systems and some data-center applications may also benefit from these ingredients [34, 42].

In topic based pub/sub networks, the number of topics is large while each topic may have only a few receivers [24]. IP multicast [13] and application level multicast have scalability and efficiency limitations under such conditions. Similarly, while multicast is a natural choice for data centers, it has the drawback of requiring routers to maintain additional state and performing costly address translations [42]. Hence, the main challenge in efficient pub/sub network design is how to build a multicast infrastructure that can scale to the general Internet and tolerate its failure modes while achieving both low latency and efficient use of resources.

In this paper, we propose a novel multicast forwarding fabric. The mechanism is based on identifying links instead of nodes and using Bloom filters [6] to encode source-route-style forwarding information into the packet header, enabling forwarding without dependency on end-to-end addressing. This provides native support for data-oriented naming and in-network caching. The forwarding decisions are simple and the forwarding tables are small, potentially allowing faster, smaller, and more energy-efficient switches than exists today. The proposed model aims towards balancing the state between the packet headers and the network nodes, allowing both stateless and stateful operations.

The presented method takes advantage of "inverting" the Bloom filter thinking [9]. Instead of maintaining Bloom filters at the network nodes and checking if incoming packets are included in the sets defined by the filters, we put the Bloom filters themselves in the packets and allow the nodes on the path to determine which outgoing links the packet should be forwarded to.

In addition to the design, we briefly describe the two implementations we have built and evaluate the scalability and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'09, August 17–21, 2009, Barcelona, Spain.
Copyright 2009 ACM 978-1-60558-594-9/09/08 ...$10.00.

efficiency of the proposed method with simulations. Further, we give an indication of the potentially achievable speed from our early measurements on our NetFPGA-based implementation.

The rest of this paper is organized as follows. First, in Section 2, we discuss the overall problem and outline the proposed solution. In Section 3, we go into details of the design. Next, in Section 4, we provide scalability evaluation of our forwarding fabric in networks up to metropolitan scales. Section 5 discusses how to inter-connect multiple networks, scaling towards Internet-wide systems, and Section 6 briefly describes our two implementations. Section 7 contrasts our work with related work, and Section 8 concludes the paper.

2. BACKGROUND AND BASIC DESIGN

Our main focus in this paper is on a multicast forwarding fabric for pub/sub-based networking. First, we briefly describe the overall pub/sub architecture our work is based on, and then present our forwarding solution, in the context of that architecture. The presented solution, providing forwarding without end-to-end addressing, is a first step towards an environment preventing DDoS attacks, as the data delivery is based on explicit subscriptions. Finally, at the end of the section, we briefly describe how our proposed forwarding fabric could be used within the present IP architecture.

2.1 A pub/sub-based network architecture

In general, pub/sub provides decoupling in *time*, *space*, and *synchronization* [15]. While publish/subscribe, as such, is well known, it is most often implemented as an overlay. Our work is based on a different approach where the pub/sub view is taken to an extreme, making the whole system based on it. In the work we rely on, inter-networking is based on topic-based publish/subscribe rather than the present send/receive paradigm [32, 39, 41].

The overall pub/sub architecture can be described through a recursive approach, depicted in Figure 1. The same architecture is applied in a recursive manner on the top of itself, each higher layer utilising the rendezvous, topology, and forwarding functions offered by the lower layers; the idea is similar to that of the RNA architecture [20] and the one described by John Day [12]. At the bottom of the architecture lies the forwarding fabric, denoted as "forwarding and more", the main focus of this paper.

The structure can be divided into a data and control plane. At the *control plane*, the topology 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 system lies the rendezvous system, which has the responsibility of handling the matching between the publishers and subscribers. The rendezvous does not need to differ substantially from other topic-based pub/sub systems; cf. [15, 23, 36]. Whenever it identifies a publication that has both a publisher (or an up-to-date cache) and one or more active subscribers, it requests the topology system to construct a logical forwarding tree from the present location(s) of the data to the subscribers and to provide the publisher (or the caches) with suitable forwarding information for the data delivery. While being aware of the scalability requirements for rendezvous and topology systems, we do not describe them in details, but refer to our ongoing work in these areas [41, 45].

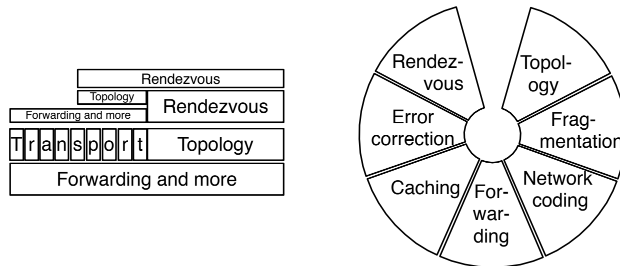

Figure 1: Rendezvous, Topology, Forwarding

The *data plane* takes care of forwarding functionality as well as traditional transport functions, such as error detection and traffic scheduling. In addition to that, a number of new network functions are envisioned (referred to as *more*), such as opportunistic caching [14, 40] and lateral error correction [3].

The data and control plane functions will work in concert, utilizing each other in a *component wheel* [41], similar to the way Haggle managers are organized [33] into an unlayered architecture, providing asynchronous way of communicating between different functional entities in a node.

In this paper, we focus on the forwarding layer, including the required information needed to be passed to it. The rendezvous and topology systems have responsibility for higher-layer operations, such as scalable handling of publish/subscribe requests (multicast tree *join/leave* in IP); they do not affect the forwarding performance directly.

2.2 Recursive bootstrapping

To achieve initial connectivity in the pub/sub network, the rendezvous and topology systems need to be bootstrapped [30]. Bootstrapping is done bottom-up, assuming 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 also used to bootstrap the rendezvous system, allowing the dedicated rendezvous nodes to advertise themselves [32, 41].

2.3 Forwarding on Bloomed link identifiers

In our approach, we do not use end-to-end addresses in the network, and instead of naming nodes, we identify all links with a name. To forward packets through the network, we use a hybrid, Bloom-filter-based approach, where the topology system both constructs *forwarding identifiers* by encoding the link identifiers into them in a source routing manner (see Figure 2), and on demand installs new state at the forwarding nodes. In this section, we present the basic ideas in a somewhat simplified form, ignoring a number of details such as loop prevention, error recovery, etc., which are described in Section 3.

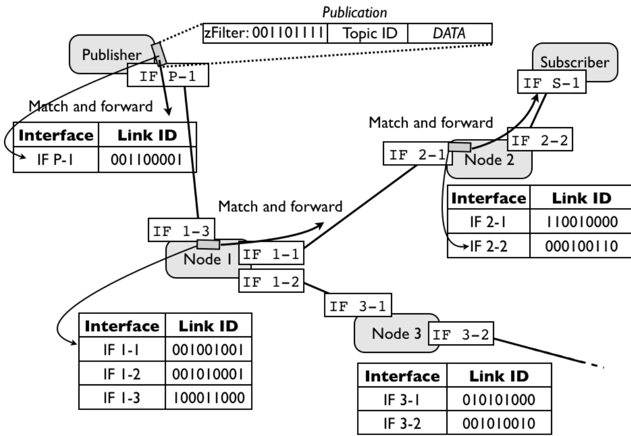

Figure 2: Example of Link IDs assigned for links, as well as a publication with a zFilter, built for forwarding the packet from the Publisher to the Subscriber.

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 multi-point (e.g. wireless) link, we consider each pair of nodes being connected with a separate link. With this setup, we do not need any common agreement between the nodes on the Link IDs – each Link ID may be locally assigned, as long as the probability of duplicates is low enough.

Basically, a Link ID is an m-bit long name with just k bits set to one. In Section 4 we will discuss the proper values for m and k, and what are the consequences 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., with $m = 248$, $k = 5$, # of Link IDs $\approx m!/(m-k)! \approx 9 * 10^{11}$).

The topology system creates a graph of the network using Link IDs and connectivity information. When it gets a request to determine a forwarding tree for a certain publication, it first creates a conceptual delivery tree using the network graph and the locations of the publisher and subscribers. Once it has such an internal representation of the tree, it knows which links the packets need to pass, and it can determine when to use Link IDs and when to create state [45]. The topology layer is also responsible for reacting to changes in the delivery tree, caused by changes in the subscriber set.

In the default case, we use a source-routing-based approach which makes forwarding independent from routing. Basically, we encode all Link IDs of the tree into a Bloom filter, and place it into the packet header. Once all link IDs have been added to the filter, a mapping from the data topic identifier to the BF is handed to the node acting as the data source and can be used for data delivery along the tree. The representation of the trees in packet headers is source specific and different sources are very likely to use different BFs for reaching the same subscriber sets. To distinguish the BFs in the actual packet headers from other BFs, we refer to the in-packet Bloom filters as zFilters[1].

Each forwarding node acts on packets roughly as follows. For each link, the outgoing Link ID is ANDed with the zFilter in the packet. 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, causing extra traffic. This sets a practical limit for the number of link names that can be included into a single zFilter.

Our approach to the Bloom filter capacity limit is twofold: Firstly, we use recursive layering [12] to divide the network into suitably-sized components; see Section 5. Secondly, the topology system may dynamically add *virtual links* to the system. A virtual link is, roughly speaking, a unidirectional delivery *tree* that consists of a number of links. It has its own Link ID, similar to the real links. 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 on a match.

2.4 Forwarding in TCP/IP-based networks

While unicast IP packets are forwarded based on address prefixes, the situation is more complicated for multicast. In source specific multicast (SSM) [19], interested receivers join the multicast group (topic) and the network creates specific multicast state based on the join messages. The state is typically reflected in the underlying forwarding fabric, for example, as Ethernet-level multicast groups or multicast forwarding state in MPLS fabrics.

From the IP point of view, LIPSIN can be considered as another underlying forwarding fabric, similar to Ethernet or MPLS. When an IP packet enters a LIPSIN fabric, the edge router prepends a header containing a suitable zFilter, see also Sect. 5.1; similarly, the header is removed at the egress edge. For unicast traffic, the forwarding entry simply contains a pre-computed zFilter, designed to forward the packet through the domain to the appropriate egress edge.

For SSM, the ingress router of the source needs to keep track of the joins received on multicast group through the edge routers, just like any IP multicast router would need to. Hence, it knows the egress edges a multicast packet needs to reach. Based on that information, it can construct a suitable zFilter from the combination of physical or virtual links to deliver the packets, leading to more flexibility and typically less state than in current forwarding fabrics.

3. DESIGN DETAILS AND EXTENSIONS

In this section, we present the details of our link-identity-based forwarding approach. We start by giving a formal description of the heart of the forwarding design, the forwarding decision. Then, we focus on enhancements of the basic design: Link ID Tags generation and selection of candidate Bloom filters. Next, we discuss additional features that make the scheme practical: virtual links, fast recovery after failures, and loop prevention. In the end, we consider control messages and return paths.

3.1 Basic forwarding method

The core of our forwarding method, the forwarding decision, is based on a binary AND and comparison operations,

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

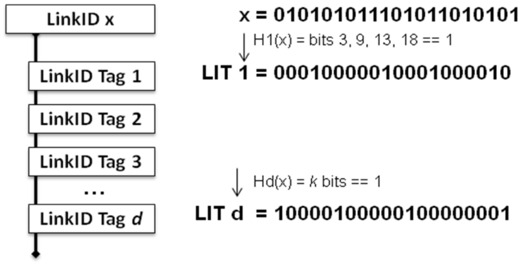

Figure 3: An example relation of one Link ID to the d LITs, using k hashes on the Link ID.

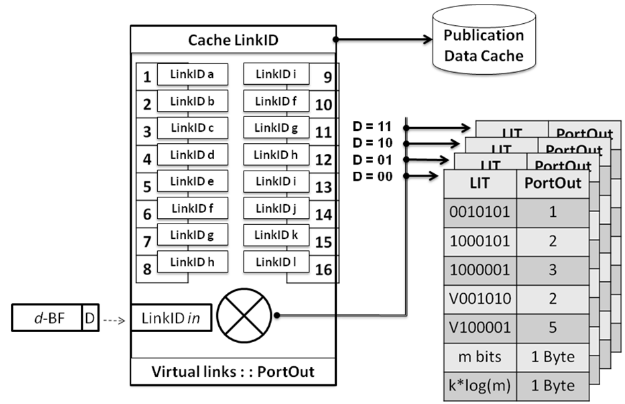

Figure 4: Outgoing interfaces are equipped with d forwarding tables, indexed by the value in the incoming packet.

both of which are very simple to implement in hardware. The base decision (Alg. 1), i.e. whether to forward on a given outbound link or not, can be easily parallelised, as there are no memory or other shared resource bottlenecks. From now on, we build an enhanced system on the top of this simple forwarding operation.

Algorithm 1: Forwarding method of LIPSIN

```
Input: Link IDs of the outgoing links; zFilter in the
       packet header
foreach Link ID of outgoing interface do
    if zFilter & Link ID == Link ID then
        Forward packet on the link
    end
end
```

3.2 Link IDs and LITs

Due to the nature of Bloom filters, a query may return a false positive, leading to a wrong forwarding decision. To reduce the number of false positives, we now introduce *Link ID Tags* (LITs), as an addition to the plain Link IDs. The idea is that instead of each link being identified with a single Link ID, every unidirectional link is associated with a set of d distinct LITs (Fig. 3). This allows us to construct different candidate zFilters and to select the best-performing one from the candidates, e.g., in terms of the false positive rate, compliance with network policies, or multi path selection.

The forwarding information is stored in the form of d forwarding tables, each containing the LIT entries of the active Link IDs, as depicted in Fig. 4. The only modification of the base forwarding method is that the node needs to be able to determine on which forwarding table it should perform the matching operations; for this, we include the index in the packet header.

Construction: When determining the actual forwarding tree based on the network graph, and the locations of the publisher and subscribers, we can apply various policy restrictions (e.g. link-avoidance) and keep traffic engineering in mind (e.g. balancing traffic load or avoiding temporarily congested parts of the network). As a result, we get a set of unidirectional links to be included into the zFilter. The final step is ORing together the corresponding LITs of the included links, yielding a candidate BF. As each link has d different identities, we get d candidate BFs that are "equivalent" representations of the delivery tree. That is, a packet using any of the candidates will follow, at minimum, all the network links inserted into the BF.

Selection: Recall that a false positive will result in an excess delivery; i.e., a packet will be forwarded over a link that is not part of the delivery tree. To achieve better performance in terms of lower false positive probability, we first consider two relatively simple strategies:

(i) Lowest false positive after hashing (fpa): The selected BF should be the one with the lowest false probability estimate after hashing: $min\{{\rho_0}^{k_0}, \ldots, {\rho_d}^{k_d}\}$, where ρ is the fill factor, i.e. the ratio of 1's to 0's.

(ii) 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 forwarding nodes along the delivery tree.

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

To further enhance *fpr*, false positives at different places can be weighted; i.e., we can consider some false positives less harmful than others. For example, we can avoid forwarding towards non-peered domains, resource constrained regions, or into potential loops. We call such selection criteria as *link avoidance*, since they are based in penalizing those candidate BFs that yield false positives when tested against certain links. For example, the following kinds of criteria could be considered:

(i) Routing policies: A T_{set} of links to be avoided due to routing policies.

(ii) Congestion mitigation: A *static* T_{set} of links avoided due to traffic engineering (e.g., low capacity links) and a *dynamic* T_{set} of congested links.

(iii) Security policies: A T_{set} of links avoided due to security concerns.

As a consequence, having multiple candidate representations for a given delivery tree is a way to minimise the number of false forwardings in the network, as well as restricting these events to places where their effects are smallest.

3.3 Stateful functionality

So far, we have considered stateless operations, where each forwarding node maintains only a static forwarding table

storing the LITs. We now carefully introduce state to the network in the form of virtual links and fast failure recovery. While increasing hardware and signaling cost, the state reduces the overall cost due to increased traffic efficiency when facing large multicast groups or link failures.

3.3.1 Virtual links

In the case of dense trees, especially when a number of trees share multiple consecutive links, it becomes efficient to identify sets of individual links with a separate Link ID and associated LITs. We call such sets of links as *virtual links.* The abstraction introduces the notion of tunnels (or link aggregation) into our architecture – a notion more general than traditional one-to-one or one-to-many tunnels, being able to represent any link sets, including partial one-to-many trees, forests of partial trees, many-to-one concast trees, etc.

A virtual link may be generated by the topology layer whenever it sees the need for such a tree. The creation process consists of selecting the individual links over which the virtual link is created, assigning it a new Link ID, and computing the LITs. To finalize the creation process, the topology layer needs to communicate the Link ID, together with the LITs, to the nodes residing on the virtual link.

Note that virtual link maintenance does not need to happen in line speed; there are always alternative ways of sending the same data. For example, if a virtual link is needed to support a very large multicast tree, the sender can still send multiple packets instead of one, each covering only a part of the tree.

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 for 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. Given the typical Zipf-distribution of the number of multicast receivers [24], the sizes of the forwarding tables will still remain small compared to the current situation with IP routers. Unfortunately, falsely matching to a virtual link will mean falsely forwarding packets through the entire connected part of the denoted subgraph; however, this can be mitigated by careful naming of the virtual links (e.g. more 1-bits than in the case of physical links) and explicitly avoiding these false positives during BF-selection.

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 around single link and node failures.

Our first approach is to replace a failed link with a functionally equivalent virtual link. We call this as *VLId-based recovery.* The idea is to have a separate virtual backup path pre-configured for each physical link ID, to be dynamically used in case of failure. This virtual backup path has the same Link ID and LITs as the physical link it replaces, but is initially inactive to avoid false forwarding.

The main advantage of this solution is that there is no need to change the packets. Basically, it is enough that the node detecting a failure sends an activation message over the replacement path, activating it 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, and as a result, the unmodified packets flow over the replacement path.

Another approach is to have a pre-computed zFilter encoding the replacement path. In this method, when a node detects a failure, it simply needs to add the appropriate LIT(s) representing the backup path 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 by increasing the fill factor of the zFilter.

Both of the mechanisms are capable of re-routing the traffic with zero convergence time and without service disruption. Besides protecting against single link failures, they are also able to recover from single node failures, if the operator has configured multiple backup paths or a backup tree towards all the neighbours of the failed node. These two types of failures cover around 85% of all unplanned outages [27]. In the complex cases where the proposed mechanisms are not able to perform local rerouting, new zFilters need to be computed.

3.3.3 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, as the constructed delivery tree may cause a loop, we can still use the *fpr* method to select only loopless candidate BFs. However, this does not guarantee loop freeness as the 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 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 compares them to the zFilter. 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 over a different link than the cached one. Our early evaluation is based on this approach and suggests that a small caching memory does not penalize the performance.

As a third alternative, at the inter-AS level we can divide the links into up, transit, and down ones, and utilise the valley-free traffic model. As a final method, it remains always possible to use TTL similar to what IP uses today.

3.3.4 Explicitly blocking false positives

Most false positives cause a packet to be sent to a node that will drop it. In some cases, the traffic generated as a result of a false positive should be fully truncated; e.g., in the case of low capacity or congested links, heavy non-cacheable traffic flows, or inter-domain link policies it may be necessary to locally disable forwarding of some traffic. Hence, we need a means to explicitly block the falsely forwarded traffic flows at an upstream point.

Therefore, any node can signal upstream a request to block a specific zFilter over that physical link. This can be implemented as a "negative" virtual Link ID, where a match blocks forwarding over the link instead of enabling it.

3.4 Control messages, slow path, and services

To inject packets into the slow path of forwarding nodes, each node can be equipped with a local, unique Link ID denoting the node-internal passway from the switching fabric to the control processor. That allows targeted control messages to be passed to one or a few specific nodes, if desired. Additionally, there may be virtual Link IDs attached to these node-local passways, making it possible to multicast control messages to a number of forwarding nodes without needing to explicitly name each of them. If the messages need to be modified, or even stopped on a node, the simultaneous forwarding should be blocked. This can be done by using zFilters constructed for node-to-node communication, or using a virtual Link ID especially configured to pass messages to the slow path and make the forwarding decision after the message has been processed.

Generalising, we make the observation that the egress points of a virtual link can be basically anything: nodes, processor cards within nodes, or even specific services. This would allow our approach to be extended to upper layers.

Another usage of control messages is collecting a symmetric reverse path from a subscriber to the publisher for the purpose of e.g. providing feedback. The publisher can initiate a control message triggering reverse path collection. Getting the message, each intermediate node bitwise ORs the appropriate reverse LIT with the path already collected and forwards it towards the subscriber. When the message finally reaches the subscriber, it will have a valid zFilter towards the publisher. The zFilter was created without interacting with the topology system.

4. EVALUATION

We now study some of the design trade-offs in detail. First, we introduce a few performance indicators, and then explore scalability limits and system performance. We use packet-level ns-3 simulations over realistic AS topologies, gaining insights on the forwarding efficiency of the proposed solution. Finally, we consider security aspects.

4.1 Performance indicators

A fundamental metric is the *false positive rate* of the in-packet Bloom filter. Link ID Tags are already in the form of m-bit vectors, with k bits set to one, as they are added to a candidate BF_i. An accurate estimate of the basic false positive rate can be given once the *fill factor* ρ of the BF is known. The *false positive after hashing* fpa is the expected false positive estimate *after* BF construction:

$$fpa = \rho^k \quad (1)$$

The *fpa-optimized* BF selection was introduced in Sec. 3.2 and is based on finding the set of LITs with the smallest predicted fpa. The *observed false positive probability* is the actual *false positive rate* (fpr) when a set of membership queries are made on the BF:

$$fpr = \frac{\#\text{Observed false positives}}{\#\text{Tested elements}} \quad (2)$$

Note that the fpr is an experimental quantity and not a theoretical estimate. The minimum observed fpr of the d candidate BFs provides a reference lower bound for a specific BF design.

These two metrics form the basic BF-selection criteria. While *fpa-optimized* selection is cheaper in computational terms, the *fpr-optimized* selection will give better results as the actual topology is more precisely considered in this process. However, the *fpr* describes the overall network performance only indirectly. In order to capture better the actual bandwidth consumption due to false positives, we introduce *forwarding efficiency* as a metric to quantify the bandwidth overhead caused by sending packets through unnecessary links:

$$fwe = \frac{\#\text{Links on shortest path tree}}{\#\text{Links during delivery}} \quad (3)$$

In other words, forwarding efficiency is 100% if the packets strictly follow the shortest path tree for reaching the subscribers. Consequently, this metric is representative and useful in the scenarios where larger subscriber sets are reached with multiple smaller delivery trees, or in virtual link scenarios, where false positives may be costly by causing deliveries over multiple hops.

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

Table 1: Graph characterization of a subset of router-level AS topologies used in the experiments.

4.2 Packet level simulations

First, we used the intra-domain AS topologies from Rocketfuel [1] to simulate the protocol behaviour. Though not completely accurate, they are a common (best) practice to experiment with new forwarding schemes in real world scenarios[2]. A second useful data set is SNDlib [28], from where we selected the largest network (*TA2*). The most important properties of these networks are shown in Table 1.

Using ns-3, we implemented a zFilter-based forwarding layer and a simple topology module, which computes zFilters based on publisher and subscriber locations and the actual network map; the selected tree is always defined by the shortest paths between the publisher and each of the subscribers. We set m, the size of the BF to 248 bits; a fair comparison to the IPv6 source and destination fields $(2 \cdot 128)$. We briefly considered $m = 120$ and $m = 504$, but abandoned the former due to poor performance and the latter due to relatively small overall gains compared to the per-packet cost. A more flexible design, allowing m to vary per packet, is left for further study. We investigated the effect of different numbers of forwarding tables (d), the number of subscribers (n), and the different LIT-sets for the nodes (constant $k = 5$, variable $k \in [3, 3, 4, 4, 5, 5, 6, 6]$), as well as different BF-selection strategies.

Stateless forwarding: We present the essence of our simulation results on Tables 2 and 3. Table 2 contains results using the *fpa* selection criteria with the variable distribution

[2] Recent studies [35] have pointed out some limitations in Rocketfuel data, suggesting that the number of actual physical routing elements may be less than inferred by their measurement technique. However, this particular inaccuracy in the present data places more stress on our mechanism than the suggested corrected scheme would place.

Users	AS	Links (#)		Efic. (%)		fpr (%)	
		mean	95_{th}	mean	95_{th}	mean	95_{th}
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

Table 2: ns-3 results for d=8, variable k-distr.

Users	AS	links	fpr_{fpa} (%)		fpr_{fpr} (%)		Stdrd
		mean	k_c	k_d	k_c	k_d	$k=5$
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

Table 3: Mean fpr values for different configurations.

of k. The performance appears adequate in all of the topologies, up to 23 subscribers ($\approx$ 32 links); forwarding efficiency is still above 90% in the majority of the test cases. The result is much better than multiple unicast, where the same links would be used multiple times by the same publication. For example, in AS3257 the unicast forwarding efficiency is only 43% for 23 subscribers.

Table 3 sheds light on the difference between *fpa* and *fpr* algorithms. There is an interesting relation between the distribution of k and the optimization strategies: in our region of interest, $k_c = 5$ performs better than the variable k distribution (k_d). As expected, *fpr-optimization* successfully reduces the false positive rate, and outperforms the non-optimised ($d = 1$) approach by 2–3 times in the scenarios with 16 users. The gain of using *fpa* instead of the non-optimised algorithm is clear, although not as significant as with *fpr*. These improvements can be also observed in the sample results of AS6161, see Fig. 5.

Of course, as the link IDs are inserted into the zFilters, delivery trees are only present in the packet headers, and therefore completely independent from each other. Hence, the number of simultaneous active trees does not affect the forwarding performance.

Stateful forwarding: In networks with scale-free properties, a large part of the traffic flows between high-degree hubs. We experimented with the effects of installing virtual

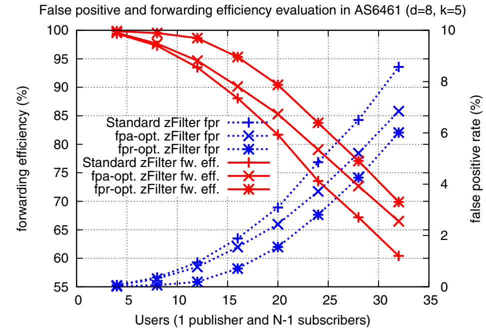

Figure 5: ns-3 simulation results for AS 6461.

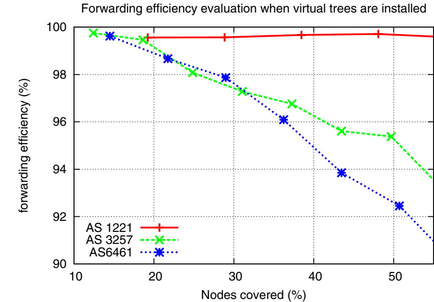

Figure 6: Stateful dense multicast efficiency

links covering different parts of the network. We built virtual links from the publisher towards the core and between the hubs, but that enhanced the performance only slightly, as virtual links substituted only a couple of physical links.

Significant performance enhancements can be reached if we install virtual links rooted at (high-degree) core nodes and covering a set of subscribers, avoiding thereby the presence of many LITs in the zFilter. The results on Fig. 6 show that dense multicast can be supported with more than 92%-95% forwarding efficiency even if we need to cover more than 50% of the total nodes in the network (cf. Table 2).

Forwarding table sizes: Assuming that each forwarding node maintains d distinct forwarding tables, with an entry consisting of a LIT and the associated output port, we can estimate the amount of memory we need for the forwarding tables:

$$FT_{mem} = d \cdot \#Links \cdot [size(LIT) + size(P_{out})] \quad (4)$$

Considering $d = 8$, 128 links (physical & virtual), 248-bit LITs and 8 bits for the outport, the total memory required would be $256Kbit$, which easily fits on-chip.

Although this memory size is already small , we can design a more efficient forwarding table by using a *sparse representation* to store just the positions of the bits set to 1. Thereby, the size of each LIT entry is reduced to $k \cdot log_2(LIT)$ and the total forwarding table requires only $\approx 48Kbit$ of memory, at the expense of the decoding logic.

4.3 Discussion

To support larger trees than we can comfortably address with a single zFilter, two choices can be considered. First, we can create virtual links to maintain the fill factor and to keep the overdeliveries under control. This comes at the

price of control traffic and the increase of state in forwarding nodes. Second, we can send multiple packets: instead of building one large multicast tree we can build several smaller ones, thereby keeping zFilters' fill factor reasonable. The packets will follow the desired route with acceptable false delivery rates, but exact copies will pass through certain links where the delivery trees overlap. Depending on the scenario specifics, this can result in more bandwidth waste than in the case of a single larger tree.

So far, we have calculated the performance of zFilters for specific sized subscriber sets. A further step is to estimate the overall performance of the network, where the traffic matrix is consisting of a large variety of different subscriber sets. Here we rely on current systems centered around disseminating information objects. First, according to RSS workload data collected at Cornell, the number of subscribers for different topics follows a Zipf distribution [24]. Second, YouTube video popularity also shows a power-law distribution in a campus network [17]. Third, IPTV channel popularity [11] was measured to have the same characteristics even with a faster drop in the case of unpopular channels than the Zipf-distribution would suggest. Fourth, in typical data centers there is a need for a large number of multicast groups, albeit all contain only a small amount of receivers [5].

Based on these observations, assuming a long tail in the popularity of topics, with $m = 248$ our results confirm that our fabric needs no forwarding state for the large majority of topics and requires virtual links or multiple sending only for the few most popular topics. This is a clear advantage compared to IP multicast solutions, where even the small groups need forwarding states in the routers. Furthermore, as we can freely combine the stateful and stateless methods, we can readily accommodate a number of changes in the popular topics before needing to signal a state change in the network, avoiding unnecessary communication overhead.

4.4 Security

The probabilistic nature of Bloom filters directly provides the basis for most of our security features. Furthermore, as zFilters are location specific, it is 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 particular set of nodes.

In a simple *zFilter contamination attack*, the attacker tries to get a single packet to be broadcasted to all possible links by using a BF containing a large amount of 1's (or even only 1's). A simple countermeasure for such attack, also observed in [44], is to limit the fill factor, e.g., to 50–70%. We have implemented this in hardware, without causing any additional delay. As a result, a randomly generated zFilter will match outgoing links only at the false positive rate resulting from the maximal allowed fill factor.

In a more advanced attack, 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 by attempting to lure lots of subscribers from different parts of the network. The attacker learns a number of valid zFilters originating at it and, using AND for the received LITs, guesses the LITs of the next few links. This attack, however, requires a lot of work, and there are a few direct countermeasures. First, the number of parallel LIT's close to the publisher can be increased and the uplink Link IDs can be changed more often. Second, by varying the selection of the Bloom filter (Sec. 3.2), though not optimal, we may increase 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. Our on-going work is focusing on hash chains and pseudo-random sequences in this area, meaning that with a shared secret between the individual forwarding nodes and the topology system the control overhead of communicating the changes could be kept at a minimum. The caveat would be that the zFilters being used in the network need to be re-calculated once in a while.

Overall, no forwarding state is created if there aren't a fairly large 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.

Finally, consider a situation where an attacker has successfully launched a *DDoS attack*. Initially, the victim can quench the packet stream by requesting the closest upstream node to filter traffic according to the operation defined in Section 3.3.4. After that, the LITs on the forwarding nodes can be changed to extinguish the attack. However, the latter is a slower operation, requiring updates to the topology layer and recalculation of zFilters for affected active subscriptions. Additional future work will consider how legitimate traffic can exploit the multi-path capabilities of the zFilters.

5. FEASIBILITY

We now turn our attention to the overall feasibility of our approach, focusing on the inter-networking aspects. In particular, we consider how our forwarding fabric can be extended to cover inter-domain forwarding. We discuss the efficiency and scalability aspects for the pure pub/sub case. For the IP-based multicast case, described in Section 2.4, we need to use currently existing mechanisms, limiting the breadth of the issues. We also discuss how the proposal is (slightly) better than IP in supporting data-oriented naming and in-network caching.

5.1 Full connectivity abstraction

As mentioned in Sect. 2, the overall architecture we rely on is based on a recursive approach, where each layer provides a full connectivity abstraction. Hence, to implement inter-domain forwarding, we need to attach two forwarding headers into a packet, an intra-domain and an inter-domain one, and replace the intra-domain header at each domain boundary. For IP multicast, the IP header with the IP multicast address takes the place of the inter-domain header.

To provide the full mesh abstraction, a domain provides an *inter-domain Link ID (IdLId)* for each of its neighboring domains. Furthermore, the domain provides a distinct Link ID to be added to packets that have local receivers. Hence, in the inter-domain zFilter of an incoming packet, there is the incoming *IdLId* for the link from the previous to this domain, the outgoing *IdLId*s for the links from this domain to any next domains, and if there are any local receivers, the *IdLId* denoting their existence.

When we receive a packet from outside, we first may verify that the packet is forwarded appropriately, e.g., that the inter-domain zFilter contains the incoming *IdLId*. Af-

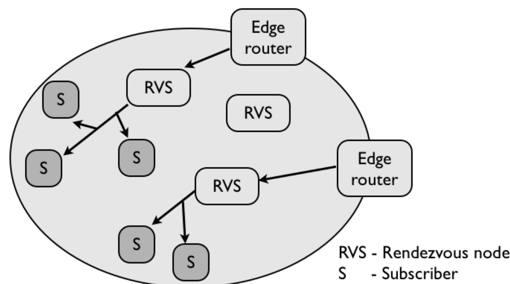

Figure 7: Inter-domain forwarding with distributed RVSs

ter that, we match the zFilter against all outgoing *IdLIds*, simultaneously looking up the corresponding intra-domain zFilters. The intra-domain zFilters can usually be simply merged. If the inter-domain zFilter indicates local recipients, more processing is needed.

We assume that the data topic identifier, carried inside the packet, or other suitable identifier such as an IP multicast address, is used to index the set of local recipients. For IP multicast addresses, it is reasonable to expect the edge nodes to maintain the required state. For the pub/sub case, where the number of active topics may be huge, the subscriber information may be divided between a set of intra-domain rendezvous nodes (see Figure 7), providing load distribution.

Eventually, a rendezvous node looks up the intra-domain zFilter by using the topic identifier. As it takes time to pass the packet to the right rendezvous node, and as the lookup may take some time, the rendezvous nodes can construct cache-like forwarding maps and distribute them to the edge nodes.

5.2 Resource consumption

We now estimate the amount of resources needed to maintain topic-based forwarding tables, needed for the recursive layering in the pub/sub case. To estimate the storage requirements, we consider the number of indexable web pages in the current Internet as a reasonable upper limit for the number of topics subscribed within a domain. In 2005 there was around 10^{10} indexable web pages [18]; today's number is larger and we assume it to be around 10^{11}. Considering that each topic name would take 40 bytes and each forwarding header takes $32-34$ bytes, in the order $\approx 10\ TB$ of storage would be needed.

Following the argumentation by Koponen et al [23] and assuming similar dynamics, it is plausible that even a single large multi-processor machine could handle the load. However, a multi-level lookup caching system is needed to reduce per-packet lookup delay to a reasonable level. For example, each edge node could cache a few million most active topics, each rendezvous node could keep in their DRAM a few billion less active topics, and the information about rest could be stored on a fast disk array. If only a small fraction of subscriptions would be active at any given point of time, the suggested multi-level caching may make it possible to handle the typical lookup load with just one or a few large server PCs.

We note that the approach may be problematic for applications where the inter-packet delay is long but latency requirements are strict. If needed, the problem can be solved by introducing explicit signaling that would allow certain topics to be always kept in the cache, even when not actively used[3].

An interesting open problem is to consider potential space saving techniques, such as determining commonalities between inter-domain zFilters, perhaps allowing them to be used as indices. If the topics sharing a single inter-domain zFilter can be distinguished with only a few bits, it may be possible to develop clever data structures for compressing the topic-based forwarding tables.

5.3 Policy compliance and traffic engineering

For the IP case, we expect no real changes to traffic engineering or policies, as the forwarding fabric would be invisible outside of the domain. In the recursive pub/sub case, we have to make sure that the inter-domain zFilters are policy-compliant. As a starting point, each edge node can verify that all traffic is either received from a paying customer or passed to a paying customer. However, due to multicast, there are difficult cases not covered by the typical IP-based policy compliance rules, such as traffic arriving from one upstream provider and destined both to a paying customer and another upstream provider. In general, we will eventually need a careful study of the issues identified by Faratin et al [16]. As observed in [30], it is an open problem how the kind of source routing we propose may change the overall market place and policies.

Considering traffic engineering, sender-based control would be easy. At this point, however, open questions include how the transit operators may affect the paths or how the receivers can express their preferences. We surmise that those aspects have to be implemented elsewhere in the architecture, as our forwarding layer can redirect traffic only by redirecting links.

5.4 Naming and caching

As mentioned earlier, both data-oriented naming and in-network caching are needed for efficient pub/sub. Our stack structure and independence of end-node addresses in zFilter forwarding, make both of these functions simpler compared to IP networks. Our architecture treats data as first class citizens. The focus is on efficient data delivery instead of connecting different hosts for resource sharing. The default choice of multicast brings natural separation of rendezvous (addressing/naming) and routing. The resulting identifier/locator split gives better support for data-oriented naming than the current IP-based architecture, cf. e.g. [2]. Once routing is based on location-independent identifiers, any kind of native naming and addressing on the infrastructure turns out to be a straightforward task.

The zFilter forwarding eases in-network caching by supporting the required decoupling between publishers and subscribers. Publishers can publish data in the network, independent of the availability of subscribers. Packet caching and further delivery from the caches is relatively simple, as node based addressing is not needed. Caching can also be used for other purposes, e.g., enhancing reliability. Combining data-oriented naming and caching, we can turn the traditional packet queues and the sibling recipient memories into opportunistic indexable caches, allowing, for example, any node to ask for recent copies of any missed or garbled

[3]Obviously, such a service would either need strict access controls or an explicit fee structure.

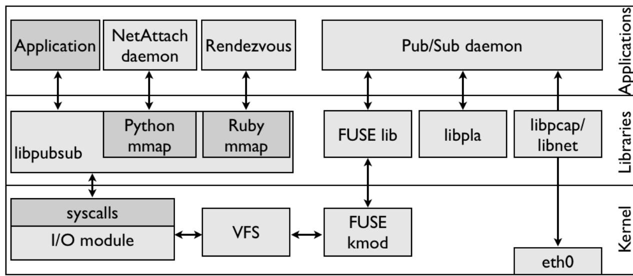

Figure 8: FreeBSD prototype structure

packets; cf. [3, 14]. The fine-grain path control allows us to easily determine those nodes that may have copies of recent packets in their memory. Multicast, in turn, allows local control queries to be sent efficiently. The falling prices of memory compared to bandwidth indicates the economical feasibility of our model.

6. IMPLEMENTATION

There are currently two partial prototypes of the system. The FreeBSD-based end-node prototype consists of some 10000 lines of C code, implementing both the pub/sub subsystem and the forwarding fabric. Our NetFPGA-based forwarding node prototype has currently some 390 lines of Verilog, implementing the main ideas from this paper. In this section, we briefly describe the implementation details and present early measurements of the NetFPGA forwarding module.

6.1 End node

The structure of the end-node prototype is depicted in Fig. 8. The I/O module implements a few new 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)[38]. Hence, each publication is backed up by a virtual file, located in a separate virtual file system running under FUSE.

Currently, forwarding and other network traffic is handled in separate threads running within the Pub/Sub daemon, simply sending and receiving raw Ethernet frames with `libnet` and `libpcap`. Ethernet frames are always broadcasted, basically using each Ethernet cable as a point-to-point link, disregarding any Ethernet bridging or switching.

6.2 Forwarding node

We have implemented an early prototype of a forwarding node using Stanford NetFPGA [25]. Starting from the Stanford reference switch implementation, we removed most of the for-us-unnecessary code in the reference pipeline and replaced it with a simple zFilter switch. At this point, we have implemented the basic LIT and virtual link ideas, and tested it with 4 real and 4 virtual LITs per interface. With this configuration, the total usage of NetFPGA resources for the logic is 4.891 4-input LUTs out of 47.232, and 1.861 Slice Flip/Flops (FF) out of 47.232. No BRAMs are reserved. For the whole system, the corresponding numbers are 20.273 LUTs, 15.347 FFs, and 106 BRAMs.

# of NetFPGAs	Average latency	Std. Dev.	Latency/ NetFPGA
0	$16\mu s$	$1\mu s$	N/A
1	$19\mu s$	$2\mu s$	$3\mu s$
2	$21\mu s$	$2\mu s$	$3\mu s$
3	$24\mu s$	$2\mu s$	$3\mu s$

Table 4: Simple latency measurement results

To get some understanding of the potential speed, we have made some early measurements. The first set of measurements, shown in Table 4, focused on the latency of the forwarding node with a very low load. In each case, the latency of 10000 packets was measured, varying the number of NetFPGAs on the path from zero (direct wire) to three. Packets were sent at a rate of 25 packets/second; both sending and receiving was implemented directly in the FreeBSD kernel.

The delay caused by the Bloom filter matching code is $56ns$ (7 clock cycles), which is insignificant compared to the measured $3\mu s$ delay of the whole NetFPGA processing. With background traffic, the average latency per NetFPGA increased to $5\mu s$.

To get an idea of the achievable throughput, we compared our implementation with the Stanford reference router. This was quantified by comparing ICMP echo requests processing times through a plain wire, our implementation, and the reference IP router with five entries in the forwarding table. To compensate the quite high deviation, caused by sending and receiving ICMP packets and involving user level processing, we averaged over 100 000 samples. The results are shown in Table 5.

While we did not directly measure the bandwidth (due to lack of test equipment to reliably fill the pipes), there are no reasons why the implementation would not operate at full bandwidth. The code is straightforward and should be able to keep the pipeline full under all conditions.

7. RELATED WORK

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

Network level multicast: Our basic communication scheme is functionally similar to IP-based source specific multicast (SSM) [19], with IP multicast groups replaced by topic identifiers. The main difference is that we support stateless multicast for sparse subscriber groups, with unicast being a special case of multicast; IP multicast typically creates lot of state in the network if one needs to support a large set of small multicast groups.

In "Revisiting IP multicast" [31], Ratnasamy et al propose source border routers to include an 800-bit Bloom-filter-based shim header (`TREE_BF`) in packets. TREE_BFs represent AS-level paths of the form $AS_a : AS_b$ in the dissemination tree of multicast packets. Moreover, a second type of Bloom filters is used to aggregate active intra-domain multicast groups piggybacked in BGP updates. The presented method uses standard IP-based forwarding mechanisms enriched with the built-in TREE_BF to take the inter-domain forwarding decisions. However, our multicast fabric uses the in-packet Bloom filter directly for the forwarding decisions, removing the need for IP-addresses and proposing Link IDs as a generic indirection primitive.

Path	Avg. latency	Std. Dev.
Plain wire	$94\mu s$	$28\mu s$
IP router	$102\mu s$	$44\mu s$
LIPSIN	$96\mu s$	$28\mu s$

Table 5: Ping through various implementations

In Xcast [7], source nodes encode the list of multicast channel destinations into the Xcast header. Each router along the way parses the header, partitions the destinations based on each destination's next hop, and forwards a packet appropriately until there is only one destination left where the Xcast packet is unicasted. Our fixed size Bloom filter approach shares the simplicity and stateless operations of Xcast while it avoids costly header re-writings and the destination IP packet header overhead.

Data-center applications and multicast: In [4] Bhargava et al discuss the performance achieved with kernel-level multicast for distributed databases. Due to the problems of IP multicast, such approaches are not commonly used in data-center applications. Recently, there has been some efforts on mapping traditional IP multicast to new models to ease wider use of IP multicast in these applications [42]. Our approach provides some new ground for considering multicast and explicit routing (e.g., middlebox serialization) in data-center environments.

Explicit routing: The simplest form of source routing [37] is based on concatenating the forwarding nodes' network identifiers on the path between senders and receivers. Our approach addresses the main caveats of source routing, including the overhead of having to carry all the routing information in the packet. Moreover, our approach does not reveal node or link identifiers, not even to the sending nodes, nor the sequence or exact amount of hops involved.

GMPLS [26] is being marketed as a solution to provide fast forwarding. By separating control and forwarding planes, it introduces more flexibility and promises performance gains, with the hardware-based fast label switching. However, it does not directly scale for massive multicast due to the limited label space and no capability for label aggregation.

In PoMo [29], Poutievski, Calvert, and Griffioen suggest an approach that trades overdeliveries for reduced state and reduced dependence of node network locators. In [10], the same authors propose an architectural approach with link identities having a pivotal role.

The BANANAS framework [22] is based on encoding each path as a short hash (PathID) of a sequence of globally known identifiers. The focus of BANANAS is on host-centric multipath communications, while ours is centered around non-global, opaque Link IDs and their compact representation. Some of the schemes developed in [22] for route computation and deployability over existing connectionless routing protocols (e.g., OSPF and BGP extensions) may be used to support LIPSIN over legacy networks.

Routing and forwarding with Bloom filters: Multiple flavours of Bloom filters [9] have been proposed to assist the forwarding operations of diverse systems (e.g., P2P, WSN, pub/sub). In the field of content-based pub/sub [21], Bloom filters are employed to represent a conjunction of subscriptions' predicates (`SBSTree`) used at content-based event forwarding time. In comparison, our pub/sub primitives are topic-based and the Bloom filters are built into packets to carry link IDs and not summarized subscriptions stored in network elements. Other forms of in-packet Bloom filters include the loop detection mechanism in Icarus [43], the credentials-based data path authentication in [44], and the aforementioned AS-level path representation for IP multicast [31].

8. CONCLUSIONS

Building on the idea of placing a Bloom filter into data packets, we have proposed a new forwarding fabric for multicast traffic. With reasonably small headers, comparable to those of IPv6, we can handle the large majority of Zipf-distributed multicast groups, up to some 20 subscribers, in realistic metropolitan-sized topologies, without adding any state in the network and with negligible forwarding overhead. For the remainder of traffic, the approach provides the ability to balance between stateless multiple sending and stateful approaches. With the stateful approach, we can handle dense multicast groups with very good forwarding efficiency. The forwarding decisions are simple, energy efficient, parallelised in hardware, and have appealing security properties. All these attributes make our work, in its current form, a potential choicer for data-center applications.

While a lot of work remains, the results indicate that it may be feasible to support Internet-wide massive multicast in a scalable manner. Technically, the main remaining obstacles are related to determining the right local delivery tree for traffic arriving from outside of a domain. Our current proposal scales only linearly. The problems related to the deployment and business aspects are likely to be even harder, but fall beyond the scope of this paper.

From a larger point of view, support for massive multicast is but one component needed for Internet-wide publish/subscribe. The other two components, data-oriented naming and in-network caching, we touched only indirectly. However, we hope that our work allows others to build upon it, allowing experimentation with network architectures that are fundamentally different from the currently deployed ones.

9. ACKNOWLEDGEMENTS

This research was supported by the EU's PSIRP project (FP7-INFSO-IST 216173). The authors thank the SIGCOMM reviewers and our shepherd Jon Crowcroft for the comments that helped to improve the paper. We also thank NomadicLab's implementation team for their efforts.

10. REFERENCES

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

[2] B. Ahlgren, L. Eggert, A. Feldmann, A. Gurtov, and T. R. Henderson. Naming and addressing for next-generation internetworks. Technical report, Dagstuhl, 2007.

[3] M. Balakrishnan, K. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral Error Correction for Time-Critical Multicast. In *NSDI' 07*, 2007.

[4] B. Bhargava, E. Mafla, and J. Riedl. Communication in the Raid distributed database system. *Comput. Netw. ISDN Syst.*, 1991.

[5] K. Birman, M. Balakrishnan, D. Dolev, T. Marian, K. Ostrowski, and A. Phanishayee. Scalable Multicast Platforms for a New Generation of Robust Distributed Applications. In *COMSWARE' 07*, 2007.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 1970.

[7] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms. Explicit multicast (Xcast) concepts and options. IETF RFC 5058, 2007.

[8] R. Briscoe. The implications of pervasive computing on network design. *BT Technology Journal*, 22(3):170–190, 2004.

[9] A. Z. Broder and M. Mitzenmacher. Survey: Network applications of Bloom filters: A survey. *Internet Mathematics*, 2004.

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

[11] M. Cha, P. Rodriguez, S. Moon, and J. Crowcroft. On next-generation telco-managed P2P TV architectures. In *IPTPS '08*, 2008.

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

[13] S. E. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. on Comp. Syst.*, 1990.

[14] F. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. Andersen. Ditto - A System for Opportunistic Caching in Multi-hop Wireless Mesh Networks. In *ACM Mobicom*, 2008.

[15] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 2003.

[16] P. Faratin, D. Clark, P. Gilmore, S. Bauer, A. Berger, and W. Lehr. Complexity of Internet interconnections: Technology, incentives and implications for policy. In *TPRC' 07*, 2007.

[17] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube Traffic Characterization: A View From the Edge. In *ACM SIGCOMM IMC'07.*, 2007.

[18] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *WWW '05*, 2005.

[19] H. Holbrook and B. Cain. Source-specific multicast for IP. RFC 4607. 2006.

[20] J.D.Touch and V.K.Pingali. The RNA metaprotocol. In *ICCCN '08*, 2008.

[21] Z. Jerzak and C. Fetzer. Bloom filter based routing for content-based publish/subscribe. In *DEBS '08*, 2008.

[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.*, 2003.

[23] 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*, 2007.

[24] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In *IMC'05*, 2005.

[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*, 2007.

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

[27] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot. Characterization of failures in an IP backbone. In *INFOCOM 2004*, 2004.

[28] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design Library. In *INOC' 07*, 2007.

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

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

[31] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. In *SIGCOMM'06*, 2006.

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

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

[34] A. Sharma, A. Bestavros, and I. Matta. dPAM: a distributed prefetching protocol for scalable asynchronous multicast in P2P systems. In *INFOCOM' 05*, 2005.

[35] R. Sherwood, A. Bender, and N. Spring. Discarte: a disjunctive Internet cartographer. *SIGCOMM Comput. Commun. Rev.*, 2008.

[36] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM'02*, 2002.

[37] C. A. Sunshine. Source routing in computer networks. *SIGCOMM Comput. Commun. Rev.*, 1977.

[38] M. Szeredi. Filesystem in Userspace. *Located at http://fuse. sourceforge. net.*

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

[40] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, and T. Bressoud. Opportunistic use of content addressable storage for distributed file systems. In *USENIX' 03*, 2003.

[41] D. Trossen (edit.). Architecture definition, component descriptions, and requirements. Deliverable D2.3, PSIRP project, 2009.

[42] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock. Dr. multicast: Rx for datacenter communication scalability. In *HotNets-VII*, 2008.

[43] A. Whitaker and D. Wetherall. Forwarding without loops in Icarus. In *Proc. of OPENARCH*, 2002.

[44] T. Wolf. A credential-based data path architecture for assurable global networking. In *IEEE MILCOM*, 2007.

[45] 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.