Università della Svizzera italiana

Faculty of Informatics

USI Technical Report Series in Informatics

Scalable Routing for Tag-Based Information-Centric Networking

Michele Papalini[1], Koorosh Khazaei[1], Antonio Carzaniga[1], Alexander L. Wolf[2]

[1] Faculty of Informatics, Università della Svizzera italiana, Switzerland
[2] Department of Computing, Imperial College London, United Kingdom

Abstract

Routing in information-centric networking remains an open problem. The main issue is scalability. Traditional IP routing can be used with name prefixes, but it is believed that the number of prefixes will grow too large. A related problem is the use of per-packet in-network state (to cut loops and return data to consumers). We develop a routing scheme that solves these problems. The service model of our information-centric network supports information pull and push using tag sets as information descriptors. Within this service model, we propose a routing scheme that supports forwarding along multiple loop-free paths, aggregates addresses for scalability, does not require per-packet network state, and leads to near-optimal paths on average. We evaluate the scalability of our routing scheme, both in terms of memory and computational complexity, on the full Internet AS-level topology and on the internal networks of representative ASes using realistic distributions of content and users extrapolated from traces of popular applications. For example, a population of 500 million users requires a routing information base of 3.8GB with an almost flat growth and, in this case, a routing update (one content descriptor) can be processed in less than 5ms on commodity hardware. We conclude that information-centric networking is feasible, even with (or perhaps thanks to) addresses consisting of expressive content descriptors.

Report Info

Published
February 2014

Number
USI-INF-TR-2014-01

Institution
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

1 Introduction

Information-centric networking (ICN) is founded on the notion that a primary purpose of modern communication networks is to provide access to *information* and, therefore, should support addressing schemes based on information rather than on host location. This notion has been developed in two historically separate streams, one examining a "pull" service [10, 12, 15] and the other a "push" service [3]. Variations and combinations of these service models have also been studied quite extensively in recent years, including within high-profile projects (e.g., CCN/NDN, DONA, SAIL). However, a fundamental problem remains open: there is yet no demonstrably scalable scheme that supports true content-based *routing*, that is, content-based *packet* switching with multiple sources/destinations (multihoming), as opposed to a per-flow/object lookup followed by a traditional host-based data transfer. In fact, largely because of this gap, the validity and utility of the whole ICN approach has been called into question [9]. In this paper we present a routing scheme that combines support for both push and pull ICN, and demonstrate its scalability through an extensive set of experiments derived from Internet-scale data sets.

The primary approach to routing and forwarding in ICN (as typified by CCN/NDN [12], but also in the

earlier work on TRIAD [10]) is to adapt IP routing to use name prefixes instead of IP prefixes. While this approach has the great advantage of reusing much of the network infrastructure, it also has fundamental limitations. First, being based on IP unicast routing, it can not reliably support multiple sources or destinations for the same information. A router may list multiple next-hops for the same prefix, but the routing scheme provides no indication of how to forward consistently across routers so as to follow one *path* to a destination (or multiple paths to multiple destinations). Moreover, multiple next-hops may lead to loops. In fact, the main approach is not to avoid but merely to detect loops, tracing each packet throughout the network with per-packet state, thereby increasing the overall cost of forwarding. For analogous reasons, IP routing/forwarding can not directly support "push" communication, especially with content-based addresses. And, here again most crucially, the already vast and growing content space is believed to pose a fundamental scalability limitation to traditional routing.

IP *multicast* routing supports push communication along multiple loop-free paths. IP multicast *partitions the information space* into groups and is most effective when many consumers are interested in all the information produced by relatively few producers. However, the complex and highly dynamic flows one can envision for an information-centric network are not amenable to an effective partitioning scheme. Such flows would require a fine-grained partitioning, which would allow applications to accurately select information of interest, but that would also use too many groups, each with only a few if any stable members. This problem, which is known as the *channelization problem* [13], severely limits the applicability of IP multicast as a substrate for ICN routing. Futhermore, this problem is inherent in the service model of IP multicast, and is therefore independent of any aggregation of multicast addresses [23].

We propose a different approach to routing, one based on trees in which edges are annotated with content descriptors. The routing scheme we propose easily supports both publish/subscribe ("push") and request/reply ("pull") ICN. While the routing scheme is almost completely independent of the addressing scheme, and in particular could work with content names, we develop it with addresses consisting of content *descriptors* and in particular *tag sets*. Tag sets are strictly more expressive than name prefixes and yet admit to an intuitive and effective aggregation that is fundamentally superior to the aggregation of name prefixes. The scheme also supports loop-free paths to multiple destinations, meaning that both communication primitives can be dynamically assigned an arbitrary fan out, from anycast (forward to any one of many destinations) to m-anycast (any m destinations) or multicast (all destinations). Last, but not least, the scheme does not use any per-packet state within the network, unlike previous designs.

A single tree may not use the most direct paths and may be more vulnerable to congestion and network partitioning. We therefore propose to use *multiple trees*, so as to reduce path lenghts on average, reduce congestion, and improve reliability. We develop a hierarchical multi-tree routing scheme that allows for the creation of sets of trees with specific properties at different levels (e.g., shortest-paths trees within an AS along with policy-specific inter-AS trees).

In principle, however, multiple trees also require larger routing tables, which leads us back to the fundamental question of scalability. We address the issue of scalability through the aggressive aggregation of content descriptors. Beyond the natural aggregation of tag sets, we develop a routing table based on PATRICIA tries that aggregate content descriptors *across all trees*. We also develop the necessary algorithms to maintain such routing tables incrementally, which is essential in the presence of dynamic user-defined addresses.

We evaluate the memory complexity of the routing scheme and its implementation at the global network scale. We emulate the scheme over the full AS-level topology of the current Internet and within a number of representative ASes. In order to test the scheme under realistic current and potential future application demands, we extrapolate from traces of some characteristic content-driven applications. These extrapolations give us various workloads of content descriptors that correspond to several hundred million users. We then use such workloads to assess the concrete memory requirements of the scheme on routers at the local and global, inter-AS level.

Our analysis shows that content descriptors indeed aggregate very effectively, and therefore that the routing information base remains contained in size even with a growing population of users and, therefore, with more and more content descriptors. For example, for a number of representative applications, a population of 500 million users using a total of nearly 10 billion content descriptors would require a routing information base of 3.8GB with an almost flat growth enabled by effective aggregation. We also show that this same aggregated routing information can be updated dynamically at a reasonably high frequency (over 200 updates per second) even on inexpensive, commodity PC hardware.

2 Network Architecture

We begin by describing the service model, addressing scheme, and overall architecture of our information-centric network. The service model is not novel *per se*, but is a significant superset of other related models [2, 12, 15]. We review this model here for clarity and completeness.

We propose a network characterized by two types of communication services: a request/reply ("pull") service and a publish/subscribe ("push") service. Both services in essence transmit information of interest from producers to consumers, and both use content descriptors (detailed below) to identify what information is offered by which producer and what information is of interest to which consumer.

The request/reply service consists of three primitive network functions:

Offer: A producer registers one or more descriptors that identify the data that the producer is willing and able to provide.

Request: A consumer requests a data packet by issuing a request packet carrying a descriptor that specifies the requested data packet. The network then delivers the request packet to one or more producers that are willing and able to satisfy the consumer's request.

Reply: A producer (or a caching router) responds to a request packet by returning a data packet (i.e., the content) carrying a descriptor that identifies the data.

The descriptors registered by producers express a standing commitment of those producers to satisfy some specific consumer interests, and are therefore used by the network to route request packets. On the other hand, requests express a single, one-time interest that is satisfied and thereby canceled by a single reply packet.

The publish/subscribe service consists of two primitive functions:

Subscription: A consumer registers one or more descriptors that specify the data that the consumer wishes to receive.

Notification: A producer publishes a data packet carrying a descriptor that identifies the data.

Here the descriptors registered by consumers represent continuing interests in specific published data and, therefore, are used to route published data packets towards interested consumers.

Notice that the two service models are almost symmetric in the way they use descriptors to attract data. To avoid ambiguities in the use of descriptors, we refer to their roles in the five primitives, respectively, as producer *offer*, consumer *request*, producer *data reply*, consumer *subscription*, and producer *notification*; we avoid the term "interest" to avoid confusion between immediate interests (requests) and continuing interests (subscriptions).

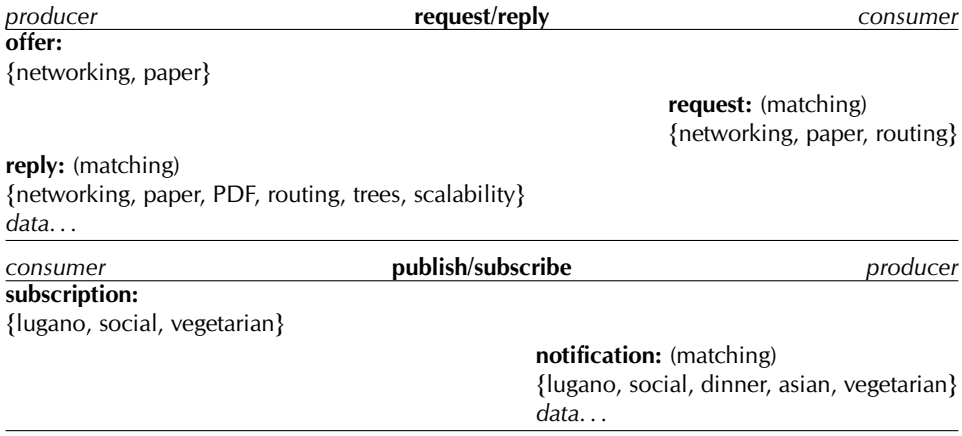

Figure 1: Tag-Based Content Descriptors

Thus descriptors play a central role analogous to IP prefixes. The semantics of descriptors define the semantics of the network service, and in particular they define how data replies match requests, how offers

match requests (and, therefore, how offers describe the data available from a producer), and how notifications match subscriptions. As discussed so far, descriptors are abstract and generic. Indeed, much of what we propose is conceptually independent of their specific form and semantics. However, in order to develop a concrete service and a corresponding concrete routing scheme, we must define descriptors. For this purpose we adopt "tags" (see Figure 1).

A descriptor consists of a set of string tags, with the matching relations corresponding to the intuitive subset relations between sets of tags. Specifically, a descriptor D in a data reply would match a descriptor R in a request when the reply contains all the tags of the request, that is, when $D \supseteq R$. Consistently, a descriptor R in a request would match a descriptor O in an offer when the request contains all the tags of the offer, and thus $R \supseteq O$. Also consistently, a descriptor N in a notification would match a descriptor S in a subscription when the notification contains all the tags of the subscription, and thus $N \supseteq S$.

Notice that tag sets are strictly more expressive than name prefixes. A name prefix can be represented as a single tag set. For example, */org/gnu/software/* can be written as tag set {*1:org, 2:gnu, 3:software*}, and would match descriptor {*1:org, 2:gnu, 3:software, 4:emacs*}. Conversely, the semantics of a tag set would require an exponential number of prefixes (all permutations).

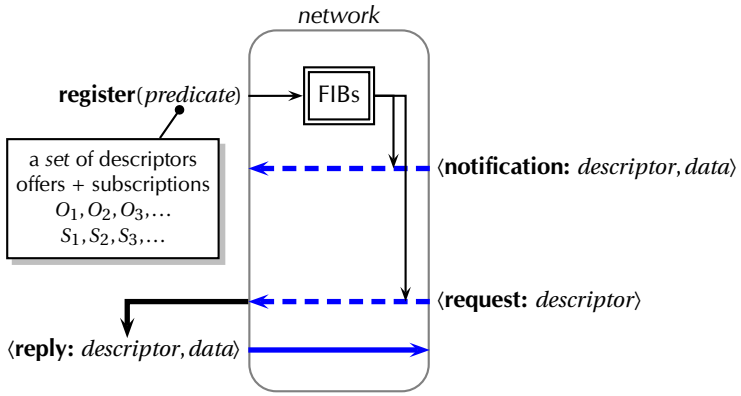

Figure 2: High-Level Network Architecture

The commonalities of the request/reply service and the publish/subscribe service suggest a network architecture in which both services share a unified FIB, as illustrated in Figure 2. Notice that for the request/reply service the FIB directs *requests* toward hosts that are willing and able to satisfy them (with corresponding data replies), while for the publish/subscribe service the FIB directs *notifications* toward hosts that are willing to receive them. So, both forms of communication allow hosts to declare which messages they intend to receive—requests and notifications, respectively—and their difference is simply the *source* of the routing information—producers in the request/reply service, and consumers in the publish/subscribe service.

In terms of data traffic, the difference between the two primitives is a bit more involved. Both requests and notifications are forwarded along paths toward matching descriptors. However, a request is ideally an anycast packet, while notifications are multicast. Also, a request is expected to generate a corresponding reply, while a notification is a one-way message. Furthermore, the caching semantics are different. A request that can be satisfied by cached content will not be forwarded downstream toward the original producer node, while a notification must be forwarded all the way to interested consumers (although notifications might also be cached for reliability purposes).

In summary, the network layer is configured through the registration of descriptors that define a descriptor-matching *predicate* analogous to a host address. All predicates feed into the forwarding information base, which is then used to transmit one-way notifications ("push") and requests that expect a reply ("pull").

3 Routing Scheme

We propose a routing scheme based on multiple trees. At the core of the scheme is simple routing on a tree cover. We enhance this basic scheme to use multiple trees within the same routing domain and over a hierarchy of domains (intra- and inter-AS). We start by describing the scheme on a single tree, elaborate on how descriptors are represented, and develop the scheme for multiple trees. We then discuss forwarding

over multiple trees, the structure of routing tables, and how to efficiently represent and maintain the routing information base.

3.1 ICN Routing on One Tree

Consider a network spanned by a tree T. For now consider a router-level network. T is identified within each notification and request packet so that each router v can determine the set adj_v^T of its neighbors that are also adjacent to v in T. This can be done by adding an identifier for T in the packet and storing the adjacency set adj_v^T at each router v.

The forwarding information base (FIB) of router v associates each neighbor w in adj_v^T with the union $P_{T,w}$ of the predicates registered by all the hosts reachable through neighbor w on T (including w). An example is shown in Figure 3.

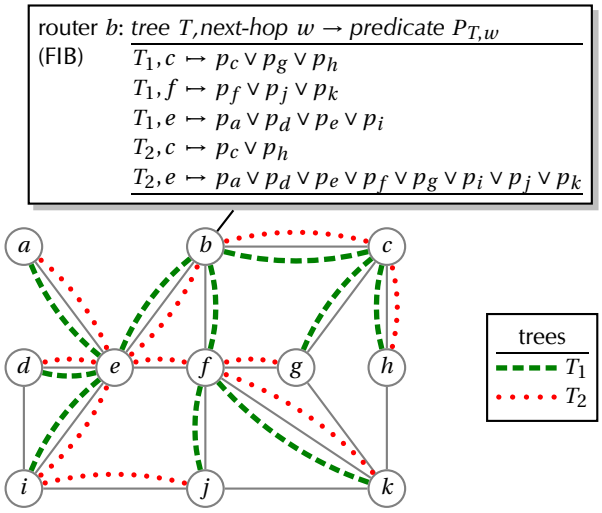

Figure 3: Multi-Tree Routing Scheme

With a FIB representing $P_{T,w}$ for each adjacent router w in adj_v^T, forwarding proceeds as follows: Router v forwards a notification with descriptor N received from neighbor u to all neighbors $w \neq u$ in adj_v^T whose associated predicate $P_{T,w}$ matches N. (A predicate P matches a descriptor X if any one of the descriptors in P matches X.) Similarly, router v forwards a request with descriptor R received from neighbor u to one neighbor $w \neq u$ in adj_v^T whose associated predicate $P_{T,w}$ matches R.

3.1.1 Fan-Out Limit: From Anycast to Multicast

A tree is an ideal structure to reach multiple destinations. Thus the basic scheme can be readily used with a given fan-out. Notifications and requests may specify a fan-out limit k. A limit of $k = 1$, which is the default for requests, corresponds to an *anycast* delivery, meaning that the request (or the notification) is forwarded to one of the neighbors with associated matching predicates. And since the forwarding is within a tree, this router-local choice is guaranteed to induce a single forwarding path to a single destination. A limit $k = \infty$, which is the default for notifications, corresponds to a full *multicast* in which the notification (or the request) is forwarded to all matching next-hops, and therefore to all hosts with matching predicates. And again since the forwarding domain is a tree, the combination of all forwarding paths will still be loop-free. A finite limit grater than one ($1 < k < \infty$) requires only minimal additional local processing: the router selects k matching neighbors and then partitions the fan-out limit k over the selected neighbors.

3.1.2 Replies and Label-Based Unicast

The mainstream approach to route replies back to the requesting host is to leave a trail of pointers along the forwarding path of each request, so that the reply can follow the same path backwards, thereby also removing the trail of pointers [12]. The trail of pointers is also essential in this approach to cut loops while forwarding requests.

Instead, the tree structure admits to an efficient forwarding of requests and replies without any kind of per-packet state. The loop-free structure of a tree avoids loops by construction, so the only remaining problem is to route replies back to the requesting node. This can be done using a routing scheme for trees developed by Thorup and Zwick [24]. Each node in the tree can be assigned a short label such that, given the label of the requesting host plus its own label, a router can very efficiently forward a reply back to the requesting host along the same path of the request (the only path, on T). These labels, which we refer to as TZ-labels, are $(1+o(1))\log_2 n$-bit long for a network of n nodes, where the vanishing constant $o(1)$ depends on a particular encoding of the labels. Concretely, we use a fast and practical version of the scheme that needs only at most $3.4\log_2 n$ bits, which for the network of our experiments amounts to 54 bits, although in practice we observed that 46-bits TZ-labels are sufficient for every tree we used to cover the Internet at the AS-level. We use TZ-labels to forward data replies as well as other unicast packets as discussed below in Section 3.3.1.

A tree can be labeled with a two-step distributed algorithm. In the first step, a converge-cast algorithm calculates the size of descendants of each node on the tree, while the second step consists of a DFS numbering of nodes on the tree. The first step can also be combined with the algorithm that constructs the tree.

3.2 Descriptors and Aggregation

Conceptually, at the application level a descriptor is a set of tags. Concretely, we represent descriptors as Bloom filters, and we develop our routing scheme around this representation. So, packets as well as routing messages carry Bloom filters. Matching two descriptors amounts to checking the inclusion relation (bitwise) between two Bloom filters, while matching a descriptor against a predicate (i.e., a set of descriptors) amounts to finding one or more Bloom filters in the predicate that are subsets (bitwise) of the input Bloom filter.

In order to choose good Bloom filter parameters, which must be global properties of the routing scheme, we conservatively estimate that tag sets would most likely have no more than 15 tags. We therefore use Bloom filters with $k = 7$ hash functions and $m = 192$ bits, which ensures that a subset test $S_1 \subseteq S_2$ would be accurate up to a false-positive probability of $\left(1 - e^{-k|S_2|/m}\right)^{k|S_1 \setminus S_2|}$. For example, for a descriptor of $|S_2| = 10$ tags, a test $S_1 \subseteq S_2$ with another descriptor S_1 that differs by $|S_1 \setminus S_2| = 3$ tags would evaluate to true (a false positive, since $|S_1 \setminus S_2| > 0$) with probability 10^{-11}.

Content descriptors aggregate in a way that is analogous to IP prefixes. Given the semantics of tag-based descriptors, a descriptor X subsumes all other descriptors Y that contain X. For example, any descriptor matching $\{networking, conference\}$ would also match its subset $\{networking\}$, so a router might combine the two by storing only the more general tag set $\{networking\}$.

This subsumption between tag sets applies directly to their representations as Bloom filters, as exemplified by the tables of Figure 4. The tables represent the per-tree maps that associate predicates with tree edges for router b in Figure 3, given a set of sample content descriptors. (For simplicity all predicates except p_j and p_k consist of a single descriptor, and descriptors are reduced to 8-bit Bloom filters.) The long table represents the full predicates, while the short table represents the aggregated predicates.

3.3 Using Multiple Trees

Routing on a tree has two disadvantages. First, paths might be "stretched", meaning the distance between two nodes on the tree might be longer than on the full graph. Second, traffic would flow only on the tree, thereby reducing the overall network throughput.

It is well known that these sorts of problems can be alleviated by using multiple trees, each with their own forwarding state. In the case of ICN, a notification or request is committed to, and thereafter routed using, one of those trees. So, the forwarding process with multiple trees is identical to that over a single tree for an individual request or notification, but traffic is more evenly distributed and path lengths shortened on average. However, two aspects of the multiple tree scheme are non-trivial: how to combine multiple trees at different levels in hierarchical routing, and how to build and then select trees.

3.3.1 Hierarchical Multi-Tree Routing

There can be multiple trees at different levels in the network (e.g., intra- and inter-AS) and they can be used in a hierarchical routing scheme. We describe the case of two levels, although the two-level hierarchy generalizes to more levels. The scheme is defined by global trees that span the AS-level network, and by local trees that span the internal network of each AS. Conceptually, each tree has a separate FIB, although below we discuss ways to aggregate predicates across trees. The FIB of a global tree contains the aggregate predicates of all the

T	$P_v^T : w \rightarrow P_{T,w}$
T_1	$c \rightarrow$ 00100101 (p_c) 01010000 (p_g) 01000001 (p_h)
	$f \rightarrow$ 00101101 (p_f) {00100100, 01010000} (p_j) {01100100, 10011000} (p_k)
	$e \rightarrow$ 00010000 (p_a) 10000101 (p_d) 00010101 (p_e) 10110000 (p_i)
T_2	$c \rightarrow$ 00100101 (p_c) 01000001 (p_h)
	$e \rightarrow$ 00010000 (p_a) 10000101 (p_d) 00101101 (p_f) 01010000 (p_g) {00100100, 01010000} (p_j) {01100100, 10011000} (p_k)

T	$P_v^T : w \rightarrow P_{T,w}$
T_1	$c \rightarrow$ 00100101 01010000 01000001
	$f \rightarrow$ 00100100 01010000 10011000
	$e \rightarrow$ 00010000 10000101
T_2	$c \rightarrow$ 00100101 01000001
	$e \rightarrow$ 00010000 10000101 00100100

Figure 4: Aggregation of Descriptors on a Per-Interface Basis

ASes. The FIB of a local tree contains the predicates of each internal host, possibly aggregated at the subnet level. An interior router needs to know only the local trees of its AS plus the TZ-labels of at least one gateway router on each local tree. A gateway router needs to know the local trees, the global trees, and the exterior connectivity of all the gateway routers of its AS, including their TZ-labels on the local trees.

With this information, forwarding proceeds as follows. A request or notification is first assigned to a local tree by its access router, and on that tree it is forwarded based on its content descriptor and fan-out limit as explained in Section 3.1. In addition to that, the request or notification is sent to one gateway router using the TZ-label of that gateway as described in Section 3.1.2. When a request or notification reaches a gateway for the first time (from a local tree) the gateway assigns it to a global tree, and then proceeds to forward it on that tree based on its content descriptor. On its global tree, a request or notification reaching a gateway router (or starting from that gateway) may have to cross the AS of that gateway to reach other gateways connected to the next-hop neighbor ASes (on the global tree). This again is done on a local tree based on the TZ-labels of those gateways. And if the request or notification is entering that AS for the first time, then the local forwarding also applies based on the content descriptor.

3.3.2 Choosing Trees

The key to increasing throughput and reducing path lengths is in the choice of trees: first, the routing process must produce a good set of trees; second, when a request or a notification enters a routing domain (local or global) the access or gateway router must assign the request or notification to a tree within that routing domain. The choice of trees, in the way they are built and they are assigned by routers, could also be used to implement various routing strategies and policies.

The problem of covering a network with trees so as to achieve specific design objectives has been studied extensively from a theoretical perspective. For example, Räcke recently formulated a method to cover a network with trees (overlay trees in this case) to achieve the theoretically minimal congestion under unknown traffic [18]. However, such results do not seem to have an immediate practical applicability. For example, Räcke's algorithms produces a very high number of trees, in the order of the number of edges in the network. Also, the algorithm itself is centralized and quite complex, and therefore may be usable for local trees but probably not for global trees.

Our approach to building and selecting trees is therefore based on heuristics. So far we studied two such heuristics for global trees, which are arguably the most crucial, and one for local trees. At the global level, the

two heuristics are intended to minimize latency alone, and latency together with congestion, respectively.

Latency Only (L): We choose a small number of root ASes and then build a shortest-paths (Dijkstra) tree for each root AS. This heuristic is intended to privilege latency over any other routing objective. For the purpose of the analysis presented in this paper we use a uniform-random choice over all ASes, which should give more conservative results. In practice, root ASes can be chosen in a number of ways using a distributed leader-election algorithm, perhaps favoring higher-tier ASes. Another and perhaps better way to select root ASes is to do it off-line through a global administrative body, in a similar way that the top-level DNS servers and structures are set up.

Latency and Congestion (LC): We start with a first shortest-paths tree rooted at the AS with the lowest eccentricity, which represents the center of the network. Then we increase the cost of each link used by the tree, and proceed iteratively with another tree. The weight increase is by a fixed amount and therefore linear in the number of trees. At each iteration we select a new shortest-paths tree rooted at the AS with the lowest eccentricity. Notice that the root may be the same as in previous iterations. However, the new tree is computed with the current adjusted link weights, and therefore is likely to differ from all previous trees. These trees can be constructed using a slightly modified version of the fast distributed algorithm of Almeida et al. [1] that computes the eccentricity $ecc(v)$ of node v in $diameter(G)+ecc(v)+2$ rounds.

At the global level, trees are heavy in terms of memory because they store the aggregated predicates of the whole network. Therefore, we compute a relatively small number of global trees. Also, at the global level we assume a mostly decentralized routing, and therefore we use shortest-paths trees that can be computed efficiently in a completely decentralized manner. Conversely, at the local level, trees are lighter and can be computed in a centralized manner. Since latency is even more crucial at the local level, the heuristic we use for local trees is also based on shortest-paths trees:

Minimal Latency: We build shortest-paths trees rooted at every router within an AS.

To assign trees dynamically, routers select trees uniformly at random at the global level, while at the local level they always choose their own shortest path tree so as to obtain latency-optimal routes. In Section 4 we evaluate our scheme under these heuristics.

3.4 Forwarding

The design of a routing scheme and the choice of a specific ICN service model must be amenable to efficient forwarding. Therefore, even though this paper focuses on routing, we now briefly discuss forwarding, presenting some preliminary experimental results.

The routing scheme uses two types of forwarding decisions: a per-tree, label-based forwarding decision using TZ-labels (Section 3.1.2) and a per-tree, content-based forwarding decision (Section 3.1). Label-based decisions are essentially stateless and extremely efficient (literally, a dozen machine-level instructions). Therefore, we focus on content-based decisions. This form of content-based forwarding amounts to a subset check: forward to interface i if there is a subset of the packet descriptors among the descriptors associated with interface i on the tree on which the packet is forwarded.

Unfortunately, this subset check, which is also known as the partial matching problem, poses a trade-off between space and time complexity [5], and represents the main cost of using the more expressive tag sets rather than name prefixes. Still, in practice this cost can be reduced significantly by engineering the forwarding algorithm and the representation of sets of tag sets. In Section 3.6 we describe how we implement such checks on tries for the purpose of routing. Here we describe a highly parallel GPU-based forwarding algorithm.

We use a minimalistic linear data structure scanned by a batch forwarding algorithm. The data structure consists of a vector of Bloom filters, and can easily fit in the memory of a modern GPU (e.g., 12GB on an NVIDIA Tesla K40) even with the largest forwarding table we used in our experimental analysis. The algorithm exploits the high parallelism of the GPU architecture by repeating very simple bitwise subset checks on each Bloom filter. The algorithm also processes forwarding requests in batches so as to exploit the high memory throughput of the internal registers of the GPU and to reduce the required main memory throughput.

The resulting throughput is around 50000 packets per second with one GPU for the largest forwarding table we analyzed, which consists of about 63 million tag sets compressed through aggregation from an initial

workload of about 10 billion. Not surprisingly, our very preliminary solution is quite far from the throughput of core routers. Nevertheless, we see good margins of improvement, and we conclude that tag-based forwarding is viable.

We reach this conclusion also on the basis of recent positive results obtained for prefix-matching in ICN using GPUs [25] and general-purpose CPUs [20]. The best throughput reported for GPU-based name lookup is much closer to the performance of today's IP routers (millions of packets per second with one GPU [25]). However, notice that the chosen matching problem is fundamentally different (prefix matching vs. the more expressive subset matching) and the table sizes are also substantially smaller (100K vs. 63M entries).

3.5 Structure of the Routing Tables

We now describe a concrete implementation of the routing information base (RIB) for the multi-tree routing scheme. Conceptually, the RIB of a router v stores the following information for each tree T:

- adj_v^T is the adjacency list of T at v, meaning the subset of v's neighbors adjacent to v on T.
- ℓ_v^T is the TZ-label of router v on T used in routing replies and other unicast packets on T (see Section 3.1.2).
- $P_v^T : w \to P_{T,w}$ is a map that associates each neighbor w in adj_v^T with a predicate $P_{T,w}$, where $P_{T,w}$ consists of a set of content descriptors (see Section 3.1 and, in particular, Figure 3).

Our primary goal is to obtain a compact representation of the RIB that also allows for efficient incremental updates. adj_v^T and ℓ_v^T require minimal space and standard data structures, and are also stable with trees. The P_v^T map changes with changing application preferences (content descriptors) and is also by far the heaviest component of the RIB. We therefore focus on the implementation of P_v^T.

3.6 RIB Representation and Maintenance

With only the basic aggregation illustrated by the tables of Figure 4, multiple trees have completely independent predicate maps (P_v^T). However, trees are likely to share many descriptors, simply because the descriptors represent offers or subscriptions that must be reachable from all trees. This suggests a representation of the predicate maps that further compresses the routing information across trees.

To exploit this form of aggregation, we develop a data structure in which routing information is not grouped by interface or tree but rather by tag set. In practice, the RIB consists of a dictionary of tag sets, each associated with a set of tree-interface pairs. In particular, we use a PATRICIA trie to index the Bloom filters representing the tag sets, and we associate each tag set with a table of 16-bit entries representing tree-interface pairs. At the global level, where the RIB must store a few trees but potentially many interfaces (neighboring ASes), we allocate 3–4 bits to identify the tree and the rest to identify the interface. At the local level the allocation is almost reversed, since there can be substantially more trees, but typically also much fewer interfaces. This data structure is depicted in Figure 5.

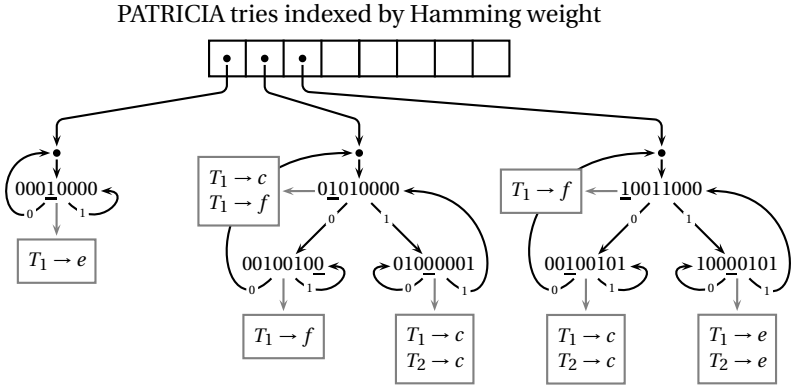

Figure 5: Aggregation of Descriptors Across Trees

PATRICIA tries have the advantage of requiring a minimal amount of memory, and they also allow for simple subset/superset checks implemented as tree walks. These checks are the essential building blocks for the maintenance of the RIB. Notice that such subset/superset checks can be linear in the number of descriptors in the RIB. As discussed above in the context of forwarding, significantly reducing the complexity bounds for these operations is not straightforward [5]. However, in practice, these algorithms can be engineered to obtain acceptable performance.

We choose a trie data structure because this structure already allows us to shortcut the search much like a prefix search: if we are looking for subsets of an input filter f and f contains a zero in a certain position identified by a node n, then we can skip the whole subtree of filters under n that contain a one in that position.

In addition to that, we group filters by Hamming weight (in smaller tries), as shown in Figure 5. This has three advantages. First, it allows us to skip entire tries containing filters that have too many elements to be subsets (or too few to be supersets) of the input filter. Second, it allows us to walk through each trie with a known limit for the number of times we have to split the walk down both the zero- and one-path. For example, if the input filter has three ones and we are looking for subsets in a trie containing filters with two ones, we know that we have to split our search at most once. Third, since tries are independent from each other, subset/superset operations on different tries can proceed in parallel and accelerated using appropriate hardware.

```
void apply_delta (map<int,delta> & result, delta update, int ifx, int tree) {
  for (filter f : update.removals)
    remove_filter(result, f, ifx, tree);
  for (filter f : update.additions)
    add_filter(result, f, ifx, tree);
}

void add_filter (map<int,delta> & result, filter f, int ifx, int tree) {
  if (!exists_subset_of(f, ifx, tree)) {
    add(f, ifx, tree);
    remove_supersets_of(f, ifx, tree);
    for (int i : interfaces[tree])
      if (i != ifx && no_subsets_on_other_ifx(f, i, tree))
        result[i].additions.add(f);
  }
}

void remove_filter (map<int,delta> & result, filter f, int ifx, int tree) {
  if (exists_filter(f, ifx, tree)) {
    remove(f, ifx, tree);
    for (i : interfaces[tree]) {
      if (i != ifx && no_subsets_on_other_ifx(f, i, tree)) {
        result[i].removals.add(f);
        result[i].additions.add(supersets_of(f, tree));
      }
    }
  }
}
```

Figure 6: Incremental Update Algorithm

Figure 6 shows the main maintenance algorithm for the routing information. Routing information propagates through update messages. The main update function *apply_delta* processes an update message received from interface *ifx* that refers to a given *tree* and that announces a set of additions and removals *delta*.

The update message may cause the router to update its own routing information base (shown in Figure 7 with tag sets attached to links) and may also trigger other update messages for the same tree (dark arrows). Such follow-up messages are returned in the *result* map. A tag set f is added in association with the incoming interface *ifx* only if f is not a superset of any existing tag set already associated with *ifx*. Also, when f is added, all supersets of f associated with *ifx* are removed. The router then propagates the addition of f to each interface i on the tree (other than *ifx*) when no subset of f is associated with any another interface. As

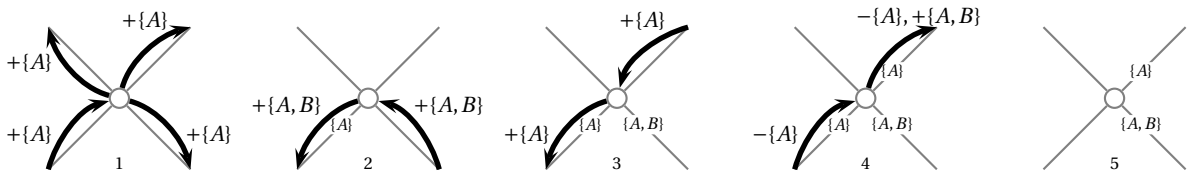

Figure 7: A Sequence of Incremental Updates

an example, see the first three updates depicted in Figure 7.

Removals are similar to additions, except that removing a tag set f may also trigger the addition of supersets of f, as exemplified by the last updates depicted in Figure 7 where the removal of the tag set $\{A\}$ triggers the addition of tag set $\{A, B\}$.

This maintenance algorithm guarantees that routing tables remain minimal, in the sense that tag sets are aggregated as much as possible on a per-interface basis.

4 Evaluation

We now present the results of an extensive experimental evaluation of our ICN routing scheme. The goal of this evaluation is twofold. On the one hand we want to assess the *effectiveness* of the scheme in routing information over the Internet using trees, specifically the ability of the scheme to obtain minimal paths and maximal throughput using a small number of trees. On the other hand we want to study the *scalability* of the scheme both in terms of the memory requirements posed on routers, and also in terms of the cost of maintaining routing information for a number of trees and for large numbers of content descriptors.

A crucial difficulty in conducting this analysis is that there is no known deployment of an information-centric network at the scale we are targeting. Therefore, we must use synthetic workloads. Below, we explain how we created such workloads.

4.1 Application Workloads

Our objective is to create workloads corresponding to the *plausible* behavior of applications over a global-scale information-centric network. To do that, we build models of future applications by extrapolating the behavior of existing applications (and their users) for which we have significant real traces.

For the purpose of this paper we are only interested in the part of such workloads that are relevant for routing, namely (1) content descriptors used in offers issued by information producers in "pull" information flows, and (2) content descriptors used in subscriptions issued by consumers in "push" flows. We consider four classes of applications: (1) "pushing" generic Web content and blog posts; (2) "pulling" video content; (3) "pushing" short messages and following short-message publishers; and (4) "pulling" BitTorrent. We now discuss each class of application and the corresponding network workload.

4.1.1 Active Web

We envision a future information-centric network used to actively distribute Web content. Rather than analyzing traditional Web requests in terms of access to individual servers, we try to understand what users are interested in, which in turn defines the descriptors used in subscriptions that would populate the routing tables. Since we could not gain access to comprehensive per-user Web-access logs, we instead infer user interests by analyzing the content that users bookmark. We use the bookmark collection of the Delicious website,[1] which contains the public bookmarks of about 950,000 users retrieved between December 2007 and April 2008 [26]. The data set contains about 132 million bookmarks and 420 million tag assignments posted between September 2003 and December 2007. We assume that users are interested in the content they bookmark, and that they describe the content with the tags they assign. Therefore, we derive plausible subscriptions from user tag sets. We slightly clean the data by applying a simple language-based summarization using stemming and removing duplicate tags. In total we derive 123,248,896 subscriptions for 922,651 users.

[1] http://delicious.com

We also analyze data collected from blogs. In particular, we study the Blog06 collection from the Text Retrieval Conference (TREC),[2] which contains 3,215,171 blog posts from 100,649 unique blogs. We use the latent Dirichlet allocation (LDA) algorithm to extract 400 topics that cover these blog posts. We then assume that an author has an active interest in a specific topic if they write more than two relevant posts on that topic, and consider a post to be relevant only if the probability of the post being classified under that topic is more than 20%. For each topic, we select the 10 most relevant tags and use them as a descriptor of the blogger's interests. Ignoring irrelevant posts and users with no significant interest in any topic, we identify 59,185 blogs with 178,189 relevant posts from which we could derive subscriptions.

4.1.2 Video Content

A future information-centric network will facilitate decentralized distribution of video content. In order to determine which content could be offered by users, which in turn determines the descriptors used in offers, we analyze data from YouTube. Uploaders of a YouTube video can assign keywords to their videos to allow viewers to find those videos with keyword searches. These keywords were publicly visible until three years ago. In particular, we analyze a data set derived from 10,351 videos published by 782 uploaders in the "Politics" category.[3]

4.1.3 Social Messaging

We analyze two different aspects of a Twitter data set to generate workloads for a plausible future messaging service. We take into account the structure of the social graph of followers as well as the content of tweets. We assume that followers are generally interested in the messages posted by the authors they follow. We therefore derive plausible subscriptions issued by the followers. We use a graph of 41.7 million Twitter users and 1.47 billion follower relations. For the content we use a collection of 16 million tweets recorded during two weeks in 2011, corresponding to 1% of the total tweets during that period. This dataset was provided again by the TREC conference (2011-2012). A Twitter user can attach a number of "hashtags" to each tweet so that other users can issue searches by hashtag. A user can also include links to other content on the Web. Out of the 16 million tweets, we consider those that have both hashtags and links. We collect the hashtags assigned to each link as a descriptor for that link, and then we use these descriptors as the subscriptions for the users who tweeted that link. In total we collect 446,370 subscriptions for 349,753 users.

4.1.4 BitTorrent

BitTorrent constitutes a considerable portion of the traffic of the Internet today. Therefore, it is important to analyze how users describe and access BitTorrent data. We use a dataset of 9,669,035 queries collected over a period of 3 months from kickasstorrents.com by the Computer Networking Research Laboratory of Colorado State University. This dataset contains 1,353,662 unique tags.[4]

4.1.5 General Data Normalization

Some datasets are characterized by large sets of tags (e.g., Delicious). However, when those sets are used to express interests in subscriptions, the specificity of those sets might be excessive, meaning that those descriptors are very unlikely to match any published data. Therefore, in order to normalize those descriptors, we always include at least 5 tags, and then for those descriptors with more tags, we add up to 10 of the remaining tags by selecting them with exponentially decreasing probabilities.

4.1.6 Data Amplification

The extrapolated workloads suffer from two limitations: they are small for the kind of experiments we want to conduct, and they are biased due to the fact that the English language is disproportionately represented in the application traces.

In order to compensate for language bias, but also to expand the size of the workload, we consider other languages that have a meaningful influence on Internet traffic, and we extend the data to include them. We

[2] http://ir.dcs.gla.ac.uk/test_collections/blog06info.html
[3] http://www.infochimps.com/datasets/11000-youtube-videos
[4] http://www.cnrl.colostate.edu/Projects/CP2P/BTData/

consider the 25 most-spoken languages in the world and amplify all of the datasets according to the distribution of the number of native speakers of those languages.

We do not want to lose the semantic correlations between tags in the datasets, therefore we derive new tags for each artificial language-specific dataset by adding a prefix indicating the corresponding language. For example, if the English data set contains the tag "Journalist", we amplify that dataset by creating a dataset for Japanese where we insert the tag "Japanese_Journalist".

To further expand the workload and also to avoid creating exact replicas of the original dataset, we create additional descriptors using synonyms. We assume that at most two tags in every descriptor might be replaced by synonyms, and we choose to replace one, two, or none with equal probability. We also assume that each tag has two synonyms and with equal probability we choose one among them. (We use artificial synonyms.) As an example, the tag "Japanese_Journalist" might be replaced by "sy1_Japanese_Journalist" in some sets.

4.1.7 Assigning Interests to Users

We distribute populations of between 50 and 500 million users in total over the various ASes, assigning the users to ASes according to the estimated population of real users for each autonomous system. We then elect among them the users that use each class of application, with an estimated probability derived from the current user population for each of the applications. In order to calculate these probabilities, we use the estimated number of users of each service over the total number of Internet users. For YouTube, the number of unique uploaders is estimated to be 10% of the user population [7]. With the current 800 million users, this yields 87.2 million uploaders. In the case of blogs, NM Incite[5] estimates that there were 181 million blogs at the end of 2011. With no data on the number of blog readers, we used the total number of blogs as the number of users of this service. In case of Twitter data, Semiocast[6] estimates the number of users exceeds 500 million as of June 2012. In January 2012, BitTorrent.com announced that the number of monthly active users reached 150 million.[7] To complete our population figures, we assume that browsing the Web is extremely common, but conservatively assume that 50% of our total user population engages in this activity.

4.2 Effectiveness with k Trees

Here we consider the topological aspects of routing, and more specifically we evaluate the ability of our scheme to use the underlying network effectively. We use two measures of cost:

Stretch: the factor by which the distance between two nodes is extended by the routing scheme. Since our scheme routes each packet on a tree, this is the ratio between the distance on the tree and the distance on the full graph. Given a set of k trees, the stretch for the path between two nodes is the expected stretch given the choice of trees, which, given our uniform choice of trees, is simply the average stretch.

Congestion: the factor by which the usage of a link would grow using the routing scheme as compared to an optimal usage of the full network graph. The optimal usage here refers to the link usage with a distribution of traffic that achieves the best possible throughput. In practice, for each tree T, given a link (u, v) in T, we compute the cut defined by that link on T, meaning the partition of the nodes that are on the two sides of the link on T. We then compute the number of links that cross that cut on the original graph, which is the total capacity of the network over that cut. Thus we assume that, for the portion of traffic routed on T ($1/k$ of the total traffic for k trees), the link (u, v) would need to carry the traffic that could instead go over all the links that cross the cut. So, for a cut of size $s_{T,u,v}$ on a tree T out of k trees, link (u, v) is given a congestion of $s_{T,u,v}/k$, and the total congestion of that link is the sum of its congestion for all the k trees.

We conduct our analysis on the Internet AS-level topology consisting of a graph of 42113 nodes and 118040 edges.[8] In Figure 8 we show the expected stretch for various sets of global (AS-level) trees. We generate sets of 2, 4, 8, 16, and 32 trees using the two heuristics discussed in Section 3.3.2. The label "L" in the plots refers to the latency-only heuristic while "LC" refers to the latency-and-congestion heuristic.

[5] http://www.nielsen.com/us/en/newswire/2012/buzz-in-the-blogosphere-millions-more-bloggers-and-blog-readers.html
[6] http://semiocast.com/publications/2012_07_30_Twitter_reaches_half_a_billion_accounts_140m_in_the_US
[7] http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users
[8] http://irl.cs.ucla.edu/topology/ (retrieved 29/06/2012)

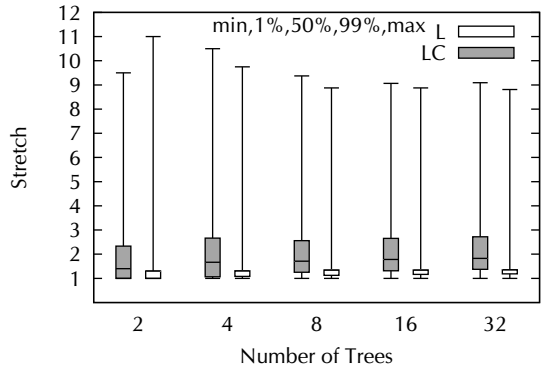

Figure 8: Path Stretch

For LC we simulate the routing scheme on only one set of trees, because LC is a deterministic algorithm. For L trees, which are generated with a randomized algorithm, we plot the aggregated results obtained for 10 different sets. Each box plot in the chart shows the minimum, the 1-percentile, the median, the 99-percentile, and the maximum. The plot clearly shows that, as anticipated, the maximum expected stretch decreases with more trees. The plot also shows that more trees lead to a minimal increase of the median (expected) stretch. Despite the growth, we can see that the stretch remains quite low, and even in the case of 32 trees the median remains under 2 and the 99-percentile is under 3. We also note that the maximum stretch is quite low, considering also that at most 1% of the paths may experience those levels of stretch.

The plot also shows a clear difference between the two heuristics, with the scheme achieving better results with the latency-only heuristic than with the latency-and-congestion heuristic.

Our experimental analysis is consistent with another study on the approximability of the AS-level topology with trees. Krioukov et al. [16] studied two compact routing schemes that have a theoretical expected stretch of 3 [24, 6], and found that in practice, on the AS-level topology, their average stretch is close to 1. Notice, however, that the two schemes require $O(n^{1/2}\log^{1/2} n)$ and $O(n^{2/3}\log^{4/3} n)$ trees, respectively, while we also achieve an average stretch very close to 1, but with significantly smaller sets of trees.

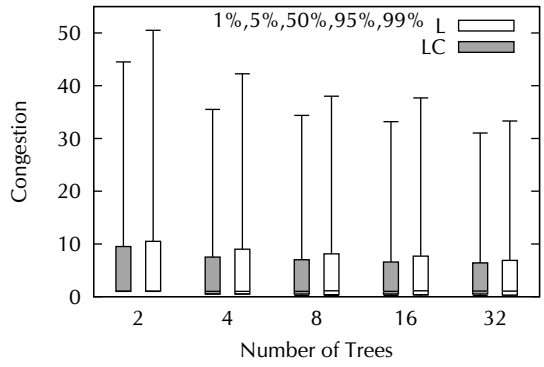

Figure 9: Link Congestion

Figure 9 shows the congestion for the same set of trees of Figure 8. In this plot each bar shows the 1-percentile, 25-percentile, median, 75-percentile, and 99-percentile of the distribution. The salient result is that most links experience no congestion penalty at all, with a congestion factor of 1. We can also see that for both heuristics, extreme levels of congestion are reduced when using more trees. Notice that the congestion factor is a very conservative cost measure, since it uses the globally optimal allocation of flow for all network cuts as a baseline. As expected, the median congestion factors for the latency-only heuristic are higher as compared with the latency-and-congestion heuristic.

The analysis of stretch and congestion shows that different tree-building strategies may be used to achieve different design goals. More importantly, the general conclusion we can draw from this analysis is that even small sets of trees can cover the Internet at the AS-level topology quite well, with only minimal cost in terms

of path length and congestion.

4.3 Scalability: Memory and Maintenance

Now we look at the memory requirements of our ICN routing scheme. In figures 10 and 11 we show the total memory requirements of particular routers in the network. This analysis is based on simulations of the routing scheme with 8 trees and under a workload generated for 50 million users spread over the ASes.

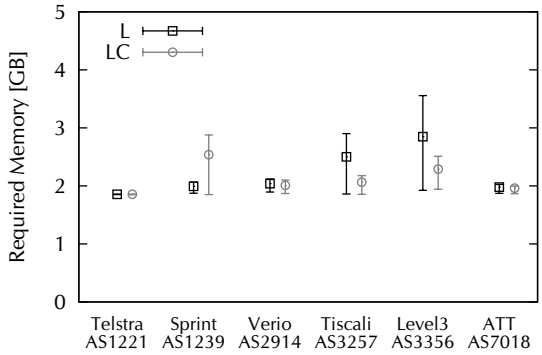

Figure 10: RIB Sizes for Global Inter-AS Trees

In Figure 10 we show the memory used by the RIBs of the gateway routers of different ASes. Since we do not know the exact connectivity between the ASes at the level of their gateway routers, we cannot determine how many trees are handled by each router. We therefore simulate all the possible cases. In particular, we simulate the pessimistic cases in which a router must hold information for all possible subsets of the 8 trees, and we show a distribution of the memory requirement for every case. The plot shows the minimum, the average and the maximum amount of memory needed to store the routing information related to 1 up to 8 trees. We show the data for two set of trees labeled "L" and "LC", as above.

The most important result is that the most demanding case, which is Level3 with the L heuristic, is less then 3.6GB of memory. The high variation is due to the different degree and location of the ASes on different trees. Usually an AS with many neighbors experiences less compression. Notice, however, that the absolute values are relatively low, and furthermore that the maximum values are less than twice the minimum in the worst case, which is well below 8 times the minimum. This indicates that aggregation is indeed effective across trees, since otherwise the maximum values, which are obtained for the full combination of 8 trees, would be 8 times the minimum values, which correspond to the combinations of 1 tree.

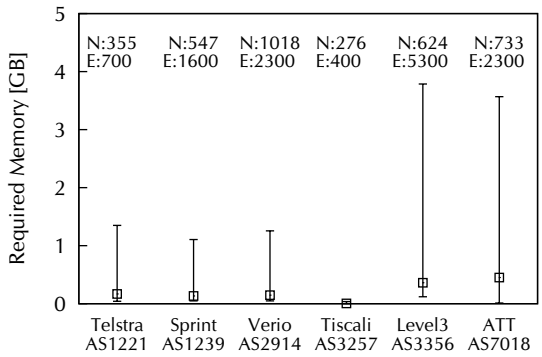

Figure 11: RIB Sizes for Local Intra-AS Trees

For each AS we also analyze the memory requirements at the intra-AS level. We use the internal AS topologies available from the Rocketfuel project [21]. The data are presented in Figure 11. The "N" and "E" labels in the graph represent the number of nodes and edges in each AS, respectively. We plot the minimum, average, and maximum sizes of the RIBs used to store local trees. Recall that for the local (intra-AS) trees we store all

the shortest-paths rooted at every node. The number of users inside each AS depends on the distribution of the 50 million over the AS-level topology. Considering the largest results, namely Level3 and AT&T on the right side of the plot, we can see that even using a large number of trees (both have hundreds of routers) we still obtain good levels of aggregation and good results in absolute terms, with a maximum memory requirements of less than 4GB.

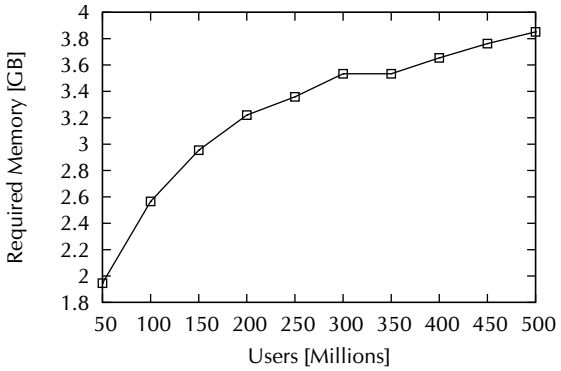

Figure 12: Scalability of the Memory Requirements (One Tree on AS 3257)

The results discussed so far are limited to a number of users that is relatively low compared to the current population of Internet users. Therefore, in order to prove the scalability of our routing scheme, we focus on a particular node (AS 3257) and only on one tree, and study the memory requirements under a workload of almost 10 billion content descriptors corresponding to 500 million users. Figure 12 shows the memory required on a gateway router for larger and larger user populations. We can see that the growth of the memory requirements is high at the beginning but then it start to grow less and less, reaching 3.8GB for 500 million users. This is due to the aggregation of tags. This means that even with higher numbers of users, the memory required to store all the routing information is likely to remain practically constant, since most of the new descriptors will be aggregated at no additional cost in the table. Notice that even assuming a constant growth after 350 million users, we would still be able to store the routing information for 2.5 billion users in less than 8GB. An important result of this analysis is that the aggregation scheme we propose is extremely effective: out of a workload of almost 10 billion tag sets, we only need to store 63.6 million of them. In this simulation we used one tree, but we can also speculate that this value would remain small across multiple trees, as demonstrated by the analysis of Figure 10.

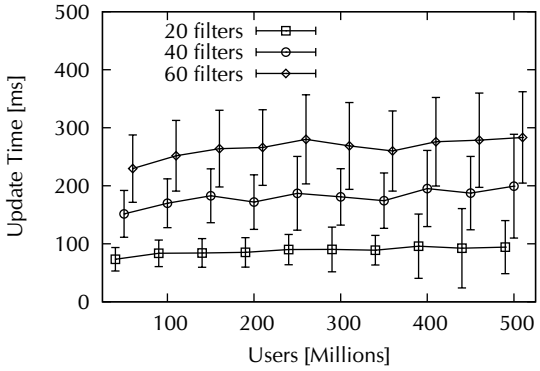

Figure 13: Scalability of the Maintenance Times

In Figure 13 we show the time we need to update the RIB using the algorithm described in Section 3.6. We run this simulation on an Intel Xeon with two quad core 2.53GHz CPUs and 16GB of RAM. In this analysis we start from the RIBs computed for the experiment of Figure 12 and then apply update packets of size 20, 40, and 60 filters. The plots show the median and the standard deviation of the update time computed over 1000 updates. The data points on the three lines are for the same table sizes (X-axis) but they are slightly shifted to avoid overlapping the standard deviation bars. The main result is that update times exhibit an

almost flat growth with the size of the RIB. On the other hand, the plots clearly show that the update time increases linearly with the size of the update packet. In particular, on average we need around 5ms per filter, which means that although we are running the algorithm on a commodity PC, we can still handle around 200 updates per second even on large RIBs.

5 Related Work

Communication in our model of ICN is to some extent similar to IP multicast, simply because one packet can be implicitly addressed to several destinations. However, beyond that, the two addressing models are radically different. IP multicast creates a partition of the information space into groups, so that one packet is associated with exactly one group of destinations. By contrast, tag sets can express many more relations between packets and destinations, and mapping those relations onto a multicast partition is fundamentally inefficient [13]. This fundamental inefficiency is not solved by multicast address aggregation. For example, Thaler and Handley [23] propose an interface-centric implementation model applicable to shared-tree protocols that allows some aggregation. However, in the absence of careful address allocation, forwarding state remains fundamentally linear in the number of active groups, and can therefore impose non-trivial costs [19].

Still, almost all multicast routing schemes are based on trees. In particular, a few schemes suggest the use of more than one independent multicast tree for each multicast group [14, 27]. The idea is to use multiple core trees to reduce delay and avoid congestion. In this case there are two design choices. *Senders-To-All* means that members join one of the cores but senders send to all cores. The opposite is *Members-To-All* where senders transmit to one core but members join all cores. Either way, this duplication in the use of network resources is once again a scalability limitation. These same considerations on the inherent limitations of the multicast model and its implementations apply to application-level multicast systems, whether they use a single tree [22] or multiple trees per group [4].

Turning now to the specific context of ICN, we note that there is surprisingly little work on this crucial aspect of network design.

The NDN project proposes NLSR [11], a link-state routing protocol for NDN. However, NLSR does not really embody a specific routing scheme. Instead, NLSR is a traditional link-state protocol that uses NDN itself to transport routing information, but ultimately provides a traditional unicast routing scheme.

Eum et al. [8] try to address the content in the caches at the routing level. They advertise the cache content locally in order to attract interest related to that content. Although they try to flood the advertisements only locally, this may generate a huge amount of traffic in the case of frequent cache updates. This may also cause some delay in the FIB updates, leading to reachability problems.

Another interesting work is presented by Papadopoulos et al. [17] who developed two greedy forwarding algorithms in a hyperbolic space. This seems to be a promising approach for routing in ICN, in particular with NDN naming. However, in order to work well in practice, the name space must be hyperbolic, and right now there is no evidence that that is the case. Another problem with this scheme is the relation between the name space and the network topology, meaning how names are distributed over the network. In fact, if names do not follow the same distribution (within the hyperbolic space) then paths can be stretched significantly. Finally, it is not clear how to compute the hyperbolic coordinates of routers and content using only local information.

6 Conclusions and Future Work

In this paper we addressed the fundamental problem of routing in an information-centric network, and in particular the essential question of the scalability of routing state. We propose a concrete scheme based on trees that supports a rich service model. Our experimental evaluation confirms two main intuitions: first, that the Internet can be approximated quite effectively with trees and, second, that tag-based content descriptors, which are more expressive than name prefixes, also aggregate better than name prefixes.

One of the most crucial open questions regarding routing is whether a multi-tree scheme, and in particular one that uses a few trees at the global level, can effectively support routing policies. Another crucial problem that we only touched upon is tag-based forwarding. We will continue to work on highly parallel forwarding algorithms capable of processing forwarding tables of hundreds of millions of tag sets. More generally, the partial matching problem is a fundamental recurring problem in tag-based routing. We will revisit this problem from a theoretical perspective and from an engineering perspective, perhaps exploring combinations of hardware and software solutions.

References

[1] P. S. Almeida, C. Baquero, and A. Cunha. Fast distributed computation of distances in networks. In *CDC*, 2012.

[2] A. Carzaniga, M. Papalini, and A. L. Wolf. Content-based publish/subscribe networking and information-centric networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking (ICN-2011)*, Aug. 2011.

[3] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proc. of ACM SIGCOMM*, Aug. 2003.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, 2003.

[5] M. Charikar, P. Indyk, and R. Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP 2002)*, 2002.

[6] L. J. Cowen. Compact routing with minimum stretch. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '99, pages 255–260, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[7] Y. Ding, Y. Du, Y. Hu, Z. Liu, L. Wang, K. Ross, and A. Ghose. Broadcast yourself: Understanding YouTube uploaders. In *Proceedings of the Internet Measurement Conference*, 2011.

[8] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga. Catt: potential based routing with content caching for icn. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, ICN '12, 2012.

[9] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: Seeing the forest for the trees. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, Nov. 2011.

[10] M. Gitter and D. R. Cheriton. An architecture for content routing support in the Internet. In *3rd USENIX Symposium on Internet Technologies and Systems*, Mar. 2001.

[11] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. Nisr: named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, ICN '13, 2013.

[12] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009.

[13] P. Ji, Z. Ge, J. Kurose, and D. Towsley. Matchmaker: Signaling for dynamic publish/subscribe applications. In *11th IEEE International Conference on Network Protocols (ICNP '03)*, Nov. 2003.

[14] W. Jia, W. Tu, W. Zhao, and G. Xu. Multi-shared-trees based multicast routing control protocol using anycast selection. *Parallel Algorithms Appl.*, 20(1), 2005.

[15] 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 *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, 2007.

[16] D. Krioukov, k. c. claffy, K. Fall, and A. Brady. On compact routing for the internet. *SIGCOMM Comput. Commun. Rev.*, 37(3):41–52, July 2007.

[17] F. Papadopoulos, D. Krioukov, M. Boguñá, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, 2010.

[18] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th annual ACM symposium on Theory of computing (STOC'08)*, 2008.

[19] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting ip multicast. *SIGCOMM Comput. Commun. Rev.*, 36(4), Aug. 2006.

[20] W. So, A. Narayanan, and D. Oran. Named data networking on a router: Fast and dos-resistant forwarding with hash tables. In *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '13, Oct. 2013.

[21] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1), Feb. 2004.

[22] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. *IEEE/ACM Trans. Netw.*, 12(2), Apr. 2004.

[23] D. G. Thaler and M. Handley. On the aggregatability of multicast forwarding state. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, Mar. 2000.

[24] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '01, 2001.

[25] Y. Wang, Y. Zu, T. Zhang, K. Peng, Q. Dong, B. Liu, W. Meng, H. Dai, X. Tian, Z. Xu, H. Wu, and D. Yang. Wire speed name lookup: A gpu-based approach. In *10th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2013.

[26] R. Wetzker, C. Zimmermann, and C. Bauckhage. Analyzing social bookmarking systems: A del.icio.us cookbook. In *Proceedings of the ECAI Workshop on Mining Social Data*, July 2008.

[27] D. Zappala, A. Fabbri, and V. M. Lo. An evaluation of shared multicast trees with multiple cores. *Telecommunication Systems*, 19(3-4), 2002.