Scalable Routing for Tag-Based Information-Centric Networking

Michele Papalini, Antonio Carzaniga, Koorosh Khazaei
University of Lugano
Lugano, Switzerland

Alexander L. Wolf
Imperial College London
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 2ms on commodity hardware. We conclude that information-centric networking is feasible, even with (or perhaps thanks to) addresses consisting of expressive content descriptors.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*

Keywords

ICN, push/pull, tag-based routing, multi-tree routing

1. INTRODUCTION

A fundamental problem remains open in information-centric networking: there is yet no demonstrably scalable scheme that supports true routing, that is, *packet switching* with multiple sources and destinations, as opposed to a per-flow or per-object lookup followed by a traditional host-based (i.e., location-based) data transfer. In fact, largely because of this gap, the validity and utility of a content-centric network layer has been rightly called into question [6].

The primary approach to routing and forwarding in ICN (as typified by CCN/NDN [10], but also in the earlier work on TRIAD [7]) 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 current network infrastructure, it also has fundamental limitations. First, since it is based on traditional *unicast* routing, it cannot 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 loops, but merely to detect them, tracing each packet throughout the network with per-packet state, thereby increasing the overall cost of forwarding. For analogous reasons, unicast routing/forwarding cannot directly support "push" ICN communication [4]. Here again, the already vast and growing content space is believed to pose a fundamental scalability limitation to traditional routing.

We develop a different approach to routing, one based on *trees* in which edges are annotated with *content descriptors*. This new routing scheme has the following novel and important properties:

- It immediately supports both request/reply ("pull") and publish/subscribe ("push") ICN communication.
- It is compatible with in-network caching, as well as the full range of existing ICN addressing schemes, from content identifiers [12] to structured names [10] to tag sets [3]. We choose tag sets, since they are strictly the most expressive form of descriptor and yet admit to an intuitive and effective aggregation that is fundamentally superior to the aggregation of, say, name prefixes.
- It provides loop-free paths to multiple destinations, meaning that communication can be dynamically assigned an arbitrary fan out, from anycast (forward to any one of many destinations) to m-anycast (any m destinations) to multicast (all destinations).

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. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ICN'14, September 24–26, 2014, Paris, France.
ACM 978-1-4503-3206-4/14/09.
http://dx.doi.org/10.1145/2660129.2660155.

- It provides extremely compact and efficient *locators* that can be used to achieve the throughput of current networks within the content-centric service interface.
- It does not require the presence of per-packet soft state within the network, unlike previous designs.

Clearly, a single tree may not use the most direct paths and would be more vulnerable to congestion and network partitioning. We therefore 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 [3]. 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 (intra-AS) and global (inter-AS) levels.

Our analysis shows that content descriptors indeed aggregate effectively and, therefore, the routing information base remains contained in size even with a growing population of users and, consequently, 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 for additional users enabled by effective aggregation. We also show that this same aggregated routing information can be updated dynamically at a reasonably high frequency (500 updates per second) even on inexpensive, commodity PC hardware.

This work is based on a previously published ICN routing scheme [3], which we extend as follows: we introduce locators and content identifiers, we detail the scheme over a hierarchy of domains, and we develop concrete data structures to represent and aggregate routing information for which we also develop incremental update and maintenance algorithms. We also conduct an extensive and in-depth analysis of the scalability of the scheme.

2. NETWORK ARCHITECTURE

We begin by describing the service model, addressing scheme, and architecture of our information-centric network. The service model extends our prior ICN design [3, 4] and is also a significant superset of other, related models [10, 12]. We review the basic model here for clarity and completeness. We also introduce two extensions to the network architecture not previously described, namely *locators* and *identifiers*.

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 are willing and able to provide.

Request: A consumer requests data by issuing a request packet carrying a content descriptor or a content locator (detailed below). 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 reply packet carrying the requested data.

The publish/subscribe service consists of two primitives:

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.

Both request/reply and publish/subscribe services define and use routing information consisting of *content descriptors*. In request/reply, producers define routing information that attracts request packets towards them, while in publish/subscribe it is consumers that define routing information to attract notifications. Thus, routing for the two services differ only in the sources of routing information, but is otherwise conceptually identical. We therefore propose a network interface with a single *register* function to define routing information.

The semantics of descriptors is also identical for requests and notifications, which means that the matching algorithm used for forwarding requests and notifications is the same. However, the treatment of the two packets differ in other ways. 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, while a notification must be forwarded all the way to interested consumers (although notifications might also be cached for reliability purposes). Figure 1 summarizes the network architecture we propose.

The network also defines opaque host *locators*. Locators are attached to requests so that the corresponding replies can be forwarded back to the requesting application. In addition to replies, we propose to use locators to forward requests. In particular, a data reply can also carry the locator

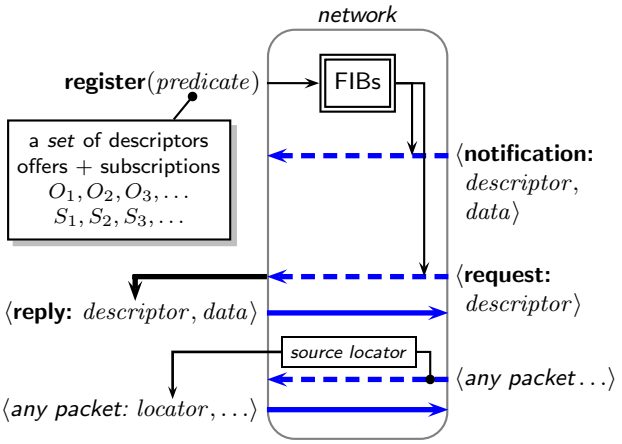

Figure 1: High-Level Network Architecture

of the producer so that the consumer can address follow-up requests (e.g., for the next data blocks) directly to that producer. Locators may be implemented with stable unicast addresses (e.g., IP addresses) or they may be based on transient state (e.g., a nonce that identifies a trail of pending interests in CCN). In Section 3.2 we detail an extremely efficient form of locators usable within our routing scheme.

2.1 Content Descriptors

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".

A descriptor consists of a set of string tags, with the matching relations corresponding to the subset relations between sets of tags: a descriptor R in a request would match a descriptor O in an offer when the request contains all the tags of the offer ($R \supseteq O$). 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 ($N \supseteq S$). For example, an offer for $\{icn14, paper\}$ would match a request $\{paper, routing, icn14\}$ or a request $\{icn14, paper, pdf, n32\}$.

Notice, however, 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 exponentially many prefixes (all permutations) to express the same descriptor.

Tag sets aggregate analogously to prefixes. In particular, a descriptor X subsumes all other descriptors Y that contain X. For example, any descriptor matching $\{music, jazz\}$ would also match its subset $\{music\}$, so a router might combine the two by storing only the more general tag set $\{music\}$. We discuss more about aggregation in Section 3.5.

Tag sets differ from IP prefixes in a crucial way. While IP prefixes are assigned by network designers and administrators, descriptors are assigned by applications. This is also true of other forms of addressing in ICN, including names in a hierarchical name space or flat identifiers. In fact, application-defined addressing is arguably the most important defining property of ICN. Allowing applications to define network addresses empowers applications but at the same time leaves the network and applications themselves vulnerable to conflicts and also abuses in the use of the address space. With tag sets, as for name prefixes, this problem can be greatly reduced through conventions, for example by defining reserved tags and mandatory scoping tags equivalent to host names in URLs.

2.1.1 Content Identifiers

A content descriptor (a tag set) may contain a unique identifier, such as a cryptographic hash of the content, or an object identifier plus a version number and a block number. In this way, a descriptor can identify a data block uniquely. More generally, tag sets can encode meta-data and higher-level protocol information. However, we believe that information that has a specific function at the network or transport level should be represented with specific headers. In particular, at the network level we propose to use cache-control headers, as well as a *content identifier* to refer to a specific data block. The form of such identifiers is defined at higher levels, for example to allow a transport protocol to refer to the next sequence of blocks within a stream.

This separation between descriptors, identifiers, and other headers is consistent with the design of a protocol such as HTTP, where the URI does not identify a piece of immutable content, but other headers can be used for that purpose (e.g., ETags, Modified-Since) and yet other headers can specify additional properties of requests and replies, such as cache controls. This design also allows us to represent descriptors using a compressed, fixed-width header that hides individual tags. We describe this compressed representation next.

2.1.2 Representation of Tag Sets

Conceptually, 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 and routing messages carry Bloom filters, and the aggregation of routing information applies equivalently to them. 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 here that tag sets would most likely contain 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 $(1 - e^{-k|S_2|/m})^{k|S_1 \setminus S_2|}$. 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}. Of course, these are network configuration parameters that can be set as appropriate.

3. ROUTING SCHEME

We introduce a routing scheme based on multiple trees. At the core of the scheme is content-based routing on a spanning tree. We enhance this basic scheme with locators and with multiple trees within routing domains and over a hierarchy of domains (intra/inter-AS).

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. Figure 2 shows an example.

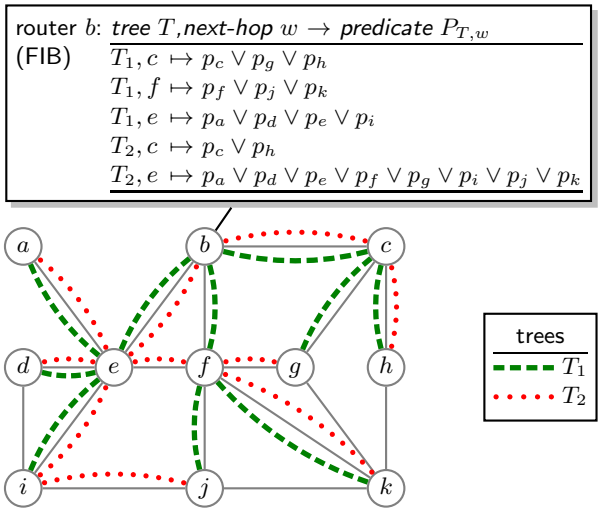

Figure 2: Multi-Tree Routing Scheme

With a FIB representing $P_{T,w}$ for all neighbor routers w in adj_v^T, forwarding proceeds as follows: Router v forwards a packet (notification or request) with descriptor X received from neighbor u on tree T to all neighbors $w \neq u$ in adj_v^T whose associated predicate $P_{T,w}$ matches X. We say that a predicate P matches a descriptor X if one of the descriptors in P matches X.

Since we use trees, we can control the global fan-out of a packet with local decisions. A packet starts with its global fan-out limit k set by the sender. A limit of $k = 1$, which is the default for requests, corresponds to an *anycast* delivery (the network delivers one copy of the packet), while a limit $k = \infty$, which is the default for notifications, corresponds to a *multicast* forwarding (the network delivers as many copies as there are interested receivers). A limit $1 < k < \infty$ can be also used and requires only minimal additional local processing: the router selects at most k matching neighbors and then partitions the fan-out limit over the selected neighbors.

3.2 Unicast Routing on Trees

We combine a locator-based unicast routing service with descriptor-based routing. To realize tree routing, we adapt a theoretical scheme for trees developed by Thorup and Zwick [19]. Within a tree, each router is assigned a short label, which we refer to as a TZ-label, such that given the TZ-label of the destination plus its own TZ-label, a router can compute the next-hop towards the destination.

This scheme is extremely efficient both in space and time. In terms of space, a router needs to store its own TZ-label, which is at most $(1 + o(1)) \log_2 n$-bit long for a network of n nodes. In practice, we found that 46-bits are sufficient to cover the Internet at the AS-level. Thus, a router spends 46 bits for each tree. In terms of time, a forwarding decision requires a few basic operations on two TZ-labels, corresponding to an average of 10 CPU cycles and a throughput of 250M packets per second on a general-purpose, commodity CPU.[1]

The scheme can also be built efficiently. A tree can be labeled with a two-step distributed algorithm. In the first step, which could be combined with the construction of the tree, a converge-cast algorithm calculates the size of descendants of each node on the tree, while the second step consists of a depth-first-search numbering of nodes on the tree.

3.3 Locators and the Request/Reply Service

As discussed in Section 2, the network forwards packets using either an explicit destination locator or a content descriptor if no locator is given. Locators are network-defined quantities that may or may not have permanent validity (like IP addresses).

In our routing scheme we use TZ-labels to implement node locators. A node locator consists of a tree identifier plus the TZ-label of the destination on that tree. We now sketch a simple request/reply protocol that combines locators and descriptors, and that can be the basis for a full transport protocol for ICN.

The general idea is to use descriptors to find an object—that is, to forward a request towards a producer capable of satisfying the request—and then to use the more efficient locators to return the data block back to the consumer and also to request other data blocks from the same producer. To implement this idea, a request packet must carry the locator S of the source application (the consumer). When a request reaches a producer capable of satisfying the request or a router with a valid cached copy of the data, the producer or caching router sends back a data reply with destination locator S, which the network forwards back to the requesting application.

The advantage of locators within our scheme is that requests, unlike interest packets in CCN [10] which create a trail of pending interests, do not require any per-packet in-network state. Without per-router pending-interests tables, our scheme does not support the aggregation of simultaneous identical requests. However, identical requests that are not exactly simultaneous can still be effectively aggregated by caching data along the forwarding path.

A data reply may also specify one or more locators of the producer as its origin, as well as the identifiers of one or more follow-up data blocks. Specifically for our scheme, the multiple locators can be obtained using multiple trees, an approach that we detail below. A consumer receiving a data reply with an origin locator may then use that locator to send follow-up requests directly towards the same origin.

[1]These results are highlights of an extensive experimental evaluation, not reported here, of all-pairs traffic forwarded using TZ-labels on multiple trees at the AS level.

This, in particular, can substantially reduce the overhead of transferring large files.

Locators built on TZ-labels are relatively stable, since they change only when trees are rebuilt, for example in response to a topology change. Still, locators may also change within a flow if producers or consumers move within the network. A transport protocol that intends to support such mobility must correctly switch locators as applications move.

Lastly, a limitation of TZ-labels is that they may reveal the identity of consumers. If anonymity is required, then locators should be based on an appropriate anonymity-preserving routing scheme, such as onion routing [8].

3.4 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, reducing the overall network throughput. It is well known that these problems can be alleviated by using multiple trees, and therefore we extend our routing scheme to use multiple trees. A notification or request is committed to, and thereafter routed using, one of those trees. Therefore, the forwarding process 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 build and then select trees, and then how to combine multiple trees at different levels in hierarchical routing.

3.4.1 Building and Selecting 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, the access router must assign the request or notification to a tree in that domain. The choice of trees, the way they are built and then 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 formulated a method to cover a network with trees to achieve the theoretically minimal congestion under unknown traffic [16]. However, such results are not applicable in practice, primarily because they can require an extremely high number of trees.

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

H1: 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 favor 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, similar to the way top-level DNS servers and structures are configured today.

H2: Latency and Congestion (LC): We start with a first shortest-paths tree rooted at the AS with the lowest eccentricity representing the center of the network. We then increase the cost of each link used by the tree, and proceed iteratively to find 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. The new tree is computed with the current adjusted link weights and, therefore, it 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. [2], which computes the eccentricity 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. Furthermore, 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 efficiently computed in a centralized manner. Since latency is crucial at the local level, the heuristic we use for local trees is also based on shortest-paths trees.

H3: 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-paths tree so as to obtain latency-minimal routes. In Section 4 we evaluate our scheme under the three heuristics.

3.4.2 Hierarchical Multi-Tree Routing

There can be multiple trees at different levels in the network, leading to a hierarchical routing scheme. We describe the case of two levels (intra- and inter-AS), although the scheme generalizes to more levels.

Routes are 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, but concretely we aggregate predicates across trees so as to reduce space (Section 3.5). The FIB of a global tree contains the aggregate predicates of all the 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 for each global 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, the network forwards packets either based on content descriptors or based on locators. We describe these two algorithms in turn.

Descriptor-Based Forwarding: A packet (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 packet is assigned to a global tree and sent to a gateway router that belongs to that tree using the TZ-label of that gateway on the local tree, which is known by the access router. On its global tree, a packet 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 packet is entering that AS for the first time, then the local forwarding is performed via the content descriptor.

Locator-Based Forwarding: In our hierarchical routing scheme, a locator consists of a stack of node locators, each one consisting of a pair (T, ℓ) where T is a tree identifier and ℓ is the TZ-label of the destination node on T. With two levels, a destination locator contains the node locator (T_{AS}, ℓ_{AS}) of the destination AS on an AS-level tree T_{AS} plus the node locator (T_r, ℓ_r) of the destination router r on an inter-AS tree. Given a destination $(T_{AS}, \ell_{AS})/(T_r, \ell_r)$, forwarding proceeds as follows: If already in the destination AS, the access router pops the (T_{AS}, ℓ_{AS}) locator from the locator stack and forwards the packet on tree T_r using TZ-label ℓ_r. Otherwise, the router pushes a locator (T, ℓ_g) of a gateway router of its AS using any intra-AS tree T, and then forwards the packet accordingly. When a packet reaches the destination at the top of the stack, the router pops the locator and proceeds with what is left on the stack. If the top locator is at the AS level, then the gateway router might have to cross its AS to reach another gateway, in which case it would push a locator of that gateway onto the stack.

3.5 RIB Representation and Maintenance

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.
- $P_v^T : w \to P_{T,w}$ 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 2).

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.

With a naive implementation (depicted in Figure 2), multiple trees would have completely independent predicate maps P_v^T with only the basic aggregation of descriptors (described in Section 2.1). 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. 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. An example is shown in Figure 3.

PATRICIA tries have the advantage of requiring a minimal amount of memory, while also allowing for simple subset/superset checks implemented as tree walks. These checks

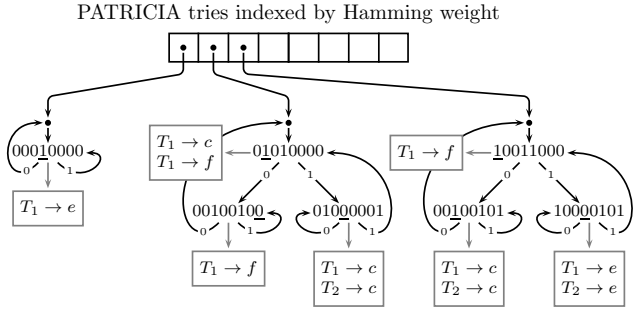

Figure 3: PATRICIA Trie Used for the RIB

are the essential building blocks for the maintenance of the RIB. The trie 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, we group filters by Hamming weight (in smaller tries). This 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. Since tries are independent of each other, subset/superset operations on different tries can also proceed in parallel.

Routing information propagates through update messages containing multiple descriptors, divided into an *addition delta*, a set of filters that we need to add in the RIB, and a *removal delta*, a set filters that we need to remove from the RIB. Figure 4 shows the main maintenance algorithm for the routing information. The main update function, *apply_delta*, processes an *update* message (of type delta) received from interface *ifx* that refers to a given *tree*.

```
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 4: Incremental Update Algorithm

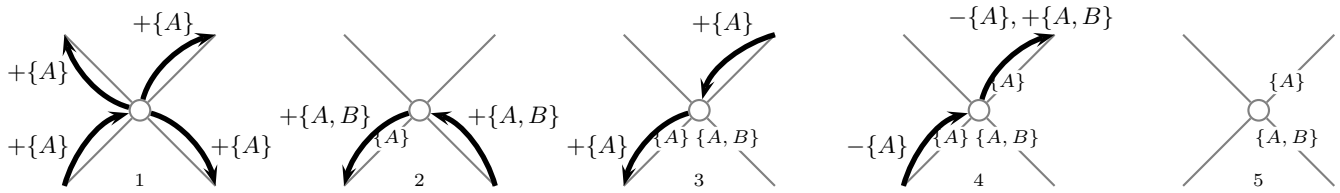

Figure 5: A Sequence of Incremental Updates

The update message may cause the router to update its own routing information base (shown in Figure 5 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 an example, see the first three updates depicted in Figure 5.

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 5, where the removal of the tag set $\{A\}$ triggers the addition of tag set $\{A, B\}$. This maintenance algorithm ensures 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. We first assess the *effectiveness* of the scheme in routing information over the Internet using a few trees. We then 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 large numbers of content descriptors.

4.1 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 conduct our analysis on the Internet AS-level topology, consisting of a graph of 42113 nodes and 118040 edges.[2] We use two measures of cost: *stretch* and *congestion*.

Stretch is 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; since we choose trees uniformly at random, it is simply the average stretch.

Congestion is 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 the 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. Notice that this congestion factor is a very conservative measure, since it uses the globally optimal allocation of flow for all network cuts as a baseline.

[2]http://irl.cs.ucla.edu/topology/ (retrieved 29/06/2012)

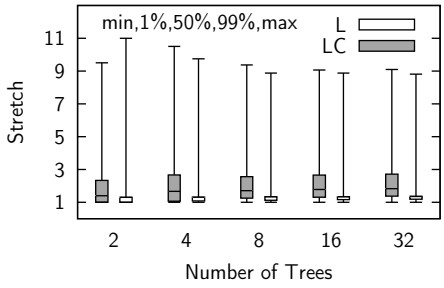

Figure 6: Path Stretch

In Figure 6 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 heuristics H1 and H2 discussed in Section 3.4.1. The label L in the plots refers to the latency-only heuristic, H1, while LC refers to the latency-and-congestion heuristic, H2. Each box plot in the chart shows the minimum, the 1-percentile, the median, the 99-percentile, and the maximum. The plot shows that the maximum expected stretch decreases with more trees, while more trees lead to a minor increase of the median (expected) stretch. Despite the growth, we can see that the stretch is low: the median always remains under 2 and the 99-percentile under 3. There is also a clear difference between the two heuristics: heuristic L achieves better results than LC.

Our experimental analysis is consistent with another study on the approximability of the AS-level topology with trees. Krioukov et al. [13] studied two compact routing schemes that have a theoretical expected stretch of 3 [5, 19], and found that in practice, on the AS-level topology, their average stretch is instead close to 1. However, the two schemes require $O(n^{1/2}\log^{1/2} n)$ and $O(n^{2/3}\log^{4/3} n)$ trees, respectively. Our scheme also achieves an average stretch very close to 1, but under significantly smaller sets of trees.

Figure 7 shows the congestion for the same set of trees of Figure 6. This plot shows the 1-percentile, 5-percentile, median, 95-percentile, and 99-percentile of the distribution.

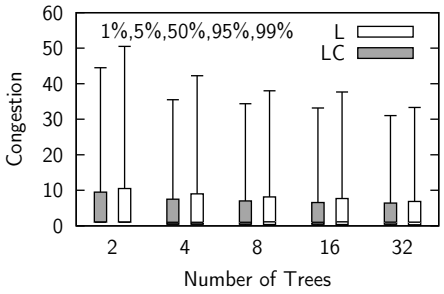

Figure 7: Link Congestion

The salient result is that most links experience no congestion penalty at all, experiencing a congestion factor of 1, and further that extreme levels of congestion are reduced when using more trees. As expected, the congestion factors for the L heuristic are higher as compared with the LC 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 stretch and link congestion.

4.2 Scalability: Memory and Maintenance

We now evaluate the memory requirements of our ICN routing scheme. For this assessment we develop a synthetic workload corresponding to the plausible behavior of users of different applications over a global-scale information-centric network [3]. We first analyze real traces from four classes of applications: active Web content and blog posts ("push"); video ("pull"); short messages and micro-blogging ("push"); and large BitTorrent downloads ("pull"). We then synthetically expand the resulting workload to 25 other languages that have a meaningful influence on Internet traffic, while preserving the semantic correlation between tags. The full description of these workloads, the methods used to normalize and expand them, and the way we associate users with different applications is detailed in a technical report [15].

4.2.1 Memory Requirements for Inter-AS RIBs

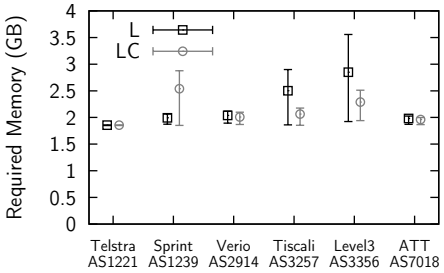

Figure 8: Sizes of Inter-AS RIBs

In Figure 8 we show the memory used by the RIBs of the gateway routers of different ASes. This analysis is based on simulations of the routing scheme with 8 trees and under a workload generated for 50 million users. However, since the exact connectivity between the ASes at the level of their gateway routers is not publicly available, we cannot determine how many trees would actually need to be known by each gateway. We therefore simulate all the possible cases and derive the distribution of the memory requirement for every case. The plot shows the minimum, the average, and the maximum amount of memory that would be needed to store the routing information for between 1 and 8 trees. We show the data for the two sets of heuristically derived trees labeled L and LC, as above. The 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: the most demanding case, which is Level3 with the L heuristic, is less then 3.6GB of memory. Furthermore, the memory required by 8 trees (maximum value), is always less than twice the memory required by a single tree (minimum value). This means that under our scheme, descriptors aggregate well across trees.

4.2.2 Memory Requirements for Intra-AS RIBs

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 [18]. The data are presented in Figure 9. The N and E labels in the graph

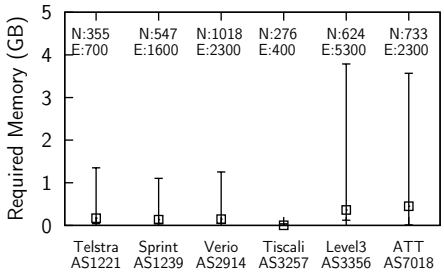

Figure 9: Sizes of Intra-AS RIBs

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 (heuristic H3). The number of users inside each AS depends on the distribution of the 50 million users over the AS-level topology. Considering the largest results, namely Level3 and AT&T, we can see that even using a large number of trees (since both have hundreds of routers) we still obtain good levels of aggregation and good results in absolute terms, with a maximum memory requirement of less than 4GB.

4.2.3 Scalability Analysis

The results discussed so far are limited to a relatively low number of users compared to the current population of Internet users. In order to better demonstrate the scalability of our routing scheme, we focus on a particular tier 1 AS (3257) and on a shortest-paths tree derived using heuristic H1 to study the memory requirement under a workload of almost 10 billion content descriptors corresponding to 500 million users. Figure 10 shows the memory required for a gateway

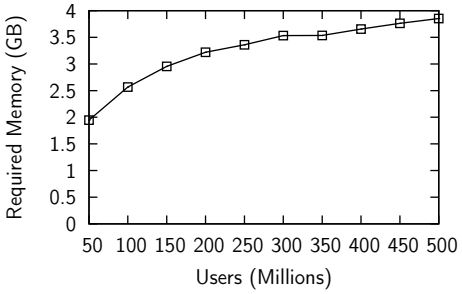

Figure 10: RIB Scalability

router for increasingly larger user populations. We can see that the growth of the memory requirement is relatively high initially, but steadily flattens, reaching 3.8GB for 500 million users. This is due directly to the aggregation of tags under our scheme: even with high 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.

4.2.4 RIB Incremental Updates

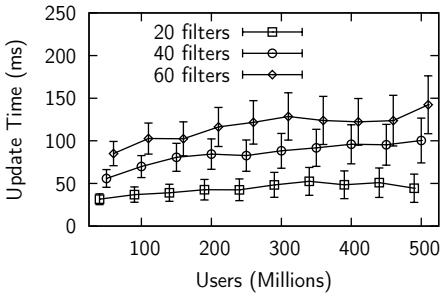

Figure 11: Scalability of the Maintenance Times

In Figure 11 we show the time needed to update the RIB using the algorithm described in Section 3.5. In this analysis we start from the RIBs computed for the experiment of Figure 10 and then apply updates of 20, 40, and 60 Bloom filters. The plot shows mean and standard deviation of the update time computed over 1000 updates. (The data points on the three lines are for the same values of the x-axis, but for purposes of readability have been slightly shifted to avoid obscuring each other.) The update time does not increase significantly with the size of the RIBs and is almost constant for each Bloom filter: 2.04ms on average. Our algorithm can handle 500 updates per second on large RIBs, even when run on a commodity PC (Intel Xeon with two quad core 2.53GHz CPUs and 16GB of RAM).

5. RELATED WORK

Although routing is one of the crucial aspects for the development of the notion of an information centric network, there is surprisingly little work on this topic. The NDN project proposes NLSR [9], a link-state routing protocol for NDN. NLSR is a traditional link-state protocol that uses NDN itself to transport routing information. NLSR realizes only a traditional unicast routing scheme that can support multiple paths with multiple runs of the Dijkstra algorithm.

Another interesting work is presented by Papadopoulos et al. [14] who developed two greedy forwarding algorithms in a hyperbolic space. This approach seems promising for routing in ICN and particularly 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.

A number of ICN proposals do not implement a routing scheme based on content names or identifiers, but instead map names or identifiers to network-level addresses. The PURSUIT project[3] uses flat names to identify each object together with a topology/resolution service to obtain a form of network-level unicast or multicast address used for the actual packet switching [11]. DONA [12] also uses flat names and uses a network of special "resolution handlers" to locate the content at the IP network level. NetInf[4] provides a name resolution scheme in combination with a name-based routing scheme similar to CCN/NDN. However, this name-based routing remains localized, so that *global* reachability is only supported through the name resolution scheme [1].

6. CONCLUSION AND FUTURE WORK

We have examined the fundamental problem of routing in an information-centric network, and the essential question of the scalability of routing state. We presented and evaluated a concrete scheme based on trees that supports a rich service model, including "push" communication and expressive content descriptors consisting of tag sets. Our evaluation confirms two intuitions: first, that the Internet can be approximated effectively with trees and, second, that tag-based content descriptors, which are more expressive than name prefixes, aggregate well under our scheme.

A crucial open question 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. We plan to explore this question by developing heuristic algorithms to build sets of trees that satisfy a given set of routing policies.

Another crucial problem that we only touched upon is tag-based forwarding. We reduce the overall complexity of forwarding with efficient locators. However, we are also working on highly parallel forwarding algorithms that combine hardware and software solutions to support high-speed forwarding with tables of hundreds of millions of tag sets. Related to the problem of forwarding, we plan to extend our evaluation, to cover forwarding but also to complement the workloads we used to evaluate the routing scheme. The workloads we used consist of content descriptors, extrapolated from traces of existing applications, that feed into the FIBs and RIBs. We will also consider other representative workloads, also extrapolated from real application traces, that are specifically intended to generate network traffic [17].

[3]http://www.fp7-pursuit.eu/
[4]http://www.sail-project.eu/

Acknowledgments

The work of M. Papalini was supported in part by the Swiss National Science Foundation under grant 200021-132565. The work of A.L. Wolf was partially sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001, and by the European Commission under grant number 318521 (Project HARNESS).

7. REFERENCES

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36, 2012.

[2] P. S. Almeida, C. Baquero, and A. Cunha. Fast distributed computation of distances in networks. In *51st IEEE Annual Conference on Decision and Control*, Dec. 2012.

[3] A. Carzaniga, K. Khazaei, M. Papalini, and A. L. Wolf. Is information-centric multi-tree routing feasible? In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking*, Aug. 2013.

[4] 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*, Aug. 2011.

[5] L. J. Cowen. Compact routing with minimum stretch. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Jan. 1999.

[6] 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*, Nov. 2011.

[7] 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.

[8] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, Feb. 1999.

[9] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. NLSR: Named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, Aug. 2013.

[10] 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*, Dec. 2009.

[11] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: Line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, Aug. 2009.

[12] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Computer Communications Review*, 37(4):181–192, Aug. 2007.

[13] D. Krioukov, K. C. Claffy, K. Fall, and A. Brady. On compact routing for the Internet. *SIGCOMM Computing Communications Review*, 37(3):41–52, July 2007.

[14] 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 IEEE Conference on Computer Communications (INFOCOM)*, Mar. 2010.

[15] M. Papalini, K. Khazaei, A. Carzaniga, and A. L. Wolf. Scalable routing for tag-based information-centric networking. Technical Report 2014/01, University of Lugano, Feb. 2014.

[16] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, May 2008.

[17] 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*, Oct. 2013.

[18] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1), Feb. 2004.

[19] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedigns of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, July 2001.