Reducing Forwarding State in Content-Centric Networks with Semi-Stateless Forwarding

Christos Tsilopoulos, George Xylomenos and Yannis Thomas
Mobile Multimedia Laboratory, Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 10434, Greece
Email: {tsilochr, xgeorge, thomasi}@aueb.gr

***Abstract*—Routers in the Content-Centric Networking (CCN) architecture maintain state for all pending content requests, so as to be able to later return the corresponding content. By employing stateful forwarding, CCN supports native multicast, enhances security and enables adaptive forwarding, at the cost of excessive forwarding state that raises scalability concerns. We propose a semi-stateless forwarding scheme in which, instead of tracking each request at every on-path router, requests are tracked at every *d* hops. At intermediate hops, requests gather reverse path information, which is later used to deliver responses between routers using Bloom filter-based stateless forwarding. Our approach effectively reduces forwarding state, while preserving the advantages of CCN forwarding. Evaluation results over realistic ISP topologies show that our approach reduces forwarding state by 54%-70% in unicast delivery, without any bandwidth penalties, while in multicast delivery it reduces forwarding state by 34%-55% at the expense of 6%-13% in bandwidth overhead.**

I. Introduction

The networking research community has recently spent considerable effort into *Information-Centric Networking* (ICN) architectures [1]. The goal of ICN is to *primarily* facilitate content distribution by further utilizing in-network *data storage* and *computation* resources. Among the various ICN proposals, the one that has received the most attention is *Content-Centric Networking* (CCN) [2], which is currently the focus of the *Named Data Networking* (NDN) project [3]. In CCN, users request *named content packets* by issuing *Interest* packets and receiving the corresponding *Data* packets. Routers propagate Interests towards the appropriate content sources and store information for each forwarded Interest in a *Pending Interest Table* (PIT). When Data arrive, routers push them towards their requester(s) based on the information stored in their PIT [2]. If an incoming Data has no match in the PIT, routers consider the Data as unwanted traffic and immediately discard it. Packet routing and forwarding is directly performed on content names, without using host addresses.

The stateful name-based forwarding of CCN offers four key advantages. First, the network provides native support for multicast delivery. If multiple users request the same content, their Interests are suppressed by common on-path routers, which later duplicate the received Data [2]. Second, host addresses are omitted, thus avoiding a number of address-related vulnerabilities (e.g. DoS attacks). Third, routers prevent the delivery of unwanted data, i.e. data that has not been explicitly requested (e.g., *spam*) [2]. Fourth, maintaining per-packet forwarding state enables routers to realize *adaptive forwarding* functionalities, i.e. routers may actively participate in functions such as link failure recovery, flow control and detection of malicious user behavior [4].

Tracking each forwarded Interest, however, raises scalability concerns [5], [6]. Per-packet forwarding state can be avoided by adopting *stateless* forwarding [7]: Interests gather path information on their way to the content source; Data are then source-routed by reversing the gathered path information. The removal of forwarding state, however, nullifies the advantages of CCN forwarding: (i) routers cannot aggregate Interests or duplicate Data, thus multicast is not supported, (ii) host addresses – that were omitted on purpose – are required by some source-routing schemes, (iii) routers cannot drop unwanted packets because forwarding state is removed, which is also the reason why (iv) adaptive forwarding is disabled.

In this paper, we propose a semi-stateless forwarding scheme for CCN. Instead of tracking an Interest at either *all* or *none* of the routers, we store forwarding information at *some* routers. An Interest is tracked at every d hops, where d is a predefined system parameter. If a data path is N hops long, an Interest is *on average* tracked at N/d routers and each router tracks *on average* $1/d$ of the forwarded Interests. At intermediate hops, Interests collect reverse path information, which is stored at routers tracking that particular Interest. Data are later forwarded via Bloom filter-based stateless forwarding [8] between the routers tracking the corresponding Interest. Our solution reduces the forwarding state maintained at routers, while preserving the advantages of name-based forwarding. Specifically, native multicast and host anonymity are preserved by the adoption of Bloom filter-based forwarding, while routers can still discard unwanted traffic and support adaptive forwarding for the fraction of Interests that they are tracking. In addition, our approach does not radically change CCN, as it only modifies Interest and Data forwarding, leaving the control plane (routing information exchange and bootstrapping) intact.

The forwarding state reduction due to our semi-stateless scheme comes at the cost of increased bandwidth overhead for multicast (but not unicast) applications. This is caused by (i) additional Interest transmissions, as Interests may not be aggregated at the first common router of the multicast tree and (ii) redundant Data transmissions due to false positives in the

Bloom filters. However, this cost is small in comparison to the gains in state maintenance. In simulations using realistic topologies, we found that forwarding state is reduced by 54%-70% in unicast applications, without any bandwidth penalties, while in multicast application it is reduced by $34\% - 55\%$ at the expense of 6%-13% in bandwidth overhead.

The remainder of this paper is organized as follows. In Section II we present an overview of CCN and its forwarding. Section III presents our forwarding scheme in detail, explaining both how Interests are tracked and how Data are forwarded. We present our evaluation in Section IV for both unicast and multicast applications and conclude in Section V.

II. Background on Content-Centric Networking

All packets in CCN include a *name*: users issue *Interest* packets specifying the desired content name and receive in response *Data* packets with the corresponding content. For each Interest, a user receives *at most one* Data packet. Content names are variable-length hierarchical identifiers similar to file-system path names or URIs, e.g. `/a/b/c.jpg`. Interests are forwarded by routers towards content sources in a hop by hop manner. At each hop, a router first checks its local cache to see if a copy of the requested packet is available. If so, the router immediately transmits the Data packet over the Interest's incoming link. Otherwise, the router checks its *Pending Interest Table* (PIT) to see if an Interest for the same Data has already been forwarded. If a PIT entry exists, the router adds the incoming interface to the PIT entry and suppresses the Interest. Otherwise, the router stores the Interest in the PIT and forwards it based on a *Longest Prefix Match* performed over the *Forwarding Information Base* (FIB). When the Interest reaches the content source, the requested Data packet is transmitted along the reverse path: at each hop, routers check their PIT for matching Interests and transmit the Data packet accordingly. If an Interest had arrived from multiple interfaces, the Data packet is duplicated, thus realizing multicast delivery. Data packets that do not find a match in the PIT are considered redundant and are discarded. After a Data packet is forwarded, the router assumes that the Interest is satisfied and deletes the PIT entry. Essentially, routers maintain state for each requested packet: as Interests are forwarded, breadcrumb-like trails are left inside the PITs. Data are returned to requesters following the reverse path, consuming PIT entries on their way. Figure 1 shows an example of CCN operation, where the Interests of three clients (U_1, U_2 and U_3) for a piece of content named `/a/b/c.jpg` have been forwarded to a content source (S); the PIT entries in the intermediate routers correspond to a multicast tree from the content source to the clients.

CCN's stateful forwarding offers four key advantages. First, CCN natively supports multicast delivery. The first common router towards a content source will aggregate Interests for the same name and duplicate the Data packets returned. This is particularly useful in real-time streaming applications [9]–[11] where users consume the same content in a synchronized manner, i.e. they transmit Interests for the same Data simultaneously. Second, host addresses are omitted and this may reduce (or, even, eliminate) a number of address-related

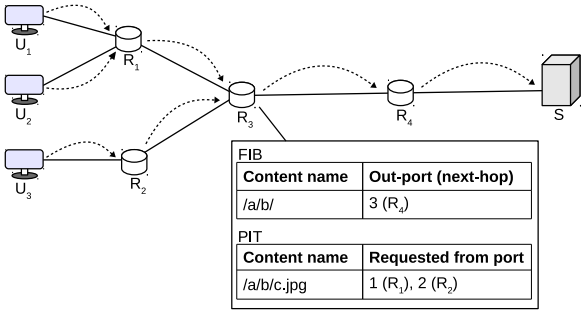


Fig. 1. Basic CCN operation. Arrows show the propagation of Interests towards content source S. The contents of the FIB and PIT at router R_3 are shown. Data follow the reverse path (tree), based on the PIT of each router.

problems, such as address space depletion, address assignment and governance [2]. Third, the network delivers only data that has been requested; routers drop Data packets that do not have a matching Interest in their PIT, thus unwanted traffic (e.g. *spam*) is discarded near the source and not at the recipient [2]. Fourth, maintaining per-packet forwarding state is an enabling factor for *adaptive forwarding* [4], in which routers may exploit forwarding state to assist functions such as fast recovery from link failures, congestion avoidance and early detection of malicious users.

The amount of forwarding state kept in routers, however, raises scalability concerns related to the PIT size. The number of Interests that must be stored in the PIT in order to fully utilize the network depends on the link capacity, the size of Data packets and the average *Round-Trip Time* (RTT) which defines the lifetime of Interests inside the PIT. A rough estimation for the required number of PIT entries per link is $bandwidth \times RTT/data\_packet\_size$. For example, to fully utilize a 40 Gbps link with 1000-byte Data packets and an average RTT of 80 ms, the PIT must contain 400K entries; this must be multiplied by the number of links hosted by the router. Furthermore, real-time applications such as multimedia streaming [9]–[11] and publish-subscribe applications [12], request Data before they are generated. This leads to an increased effective RTT, as Interests remain longer in the PIT. The work in [13] mapped realistic IP traffic onto CCN and estimated that a 20 Gbps access router would require 1.5M of PIT entries.[1] Furthermore, work in [14] estimates that, in an extreme worst case scenario, the PIT may reach 30-60M entries. Taking into account that CCN names are variable-length and, in general, much longer than host addresses, the total memory requirements for the PIT grow significantly.

A very large PIT has grave implications for network throughput. Since the PIT is examined for *every* arriving packet, it should *ideally* reside in the line-cards' on-chip memory, which is very fast but has very limited capacity [5], [6]. Initial investigations reported that a hash table-based implementation is too big to fit inside today's on-chip memories [5], [14]. The fallback option is to place the PIT in the router's main memory which is much slower, thus

[1] In this study, the authors assumed that an Interest can correspond to multiple Data packets. This radically changes the basic CCN behavior of *one Interest per Data* and may heavily underestimate the amount of PIT entries.

causing a performance degradation. Two recently proposed PIT implementations, DiPIT [15] and *Encode Name Prefix Trie* (ENPT) [13], can, under certain assumptions on the traffic mix and average content-name length, substantially reduce the memory-footprint of the PIT, but not enough to fit it into a line-card's on-chip memory. In addition, DiPIT encodes multiple Interests in Bloom filters, thus losing the information of which particular Interests are stored, which is crucial for dropping stale Interests [14] and adaptive forwarding. ENPT, on the other hand, organizes the PIT in a trie-like structure with a *linear* ($O(N)$) complexity for insert and lookup operations (where *N* is the number of components in a CCN name), compared to the *constant* ($O(1)$) complexity of hash tables.

The CONET architecture, which is basically a stateless variant of CCN, addressed the issue of forwarding state by moving the forwarding information from the routers to the packet headers [7]. In CONET, all packets carry a path header listing a sequence of node identifiers. During Interest propagation, routers append their identifier in the packet's path header. When the Interest reaches the content source, the header is reversed and placed as a source-route in the Data header. By removing the PIT entirely, this stateless forwarding approach loses the advantages of CCN's stateful forwarding. First, support for multicast is reduced, if not totally nullified. A router no longer has the required information to suppress Interests and duplicate Data. Interests for the same content are individually forwarded to the content source, which then unicasts the corresponding Data to each requester. Second, forwarding relies on node identifiers (e.g. IPv4 node addresses) that were originally omitted on purpose. Bringing node identifiers back to the network architecture will eventually lead us to the problems that CCN meant to avoid in the first place. Third, nodes cannot drop unwanted traffic since they no longer track which Data have been requested. And fourth, the complete removal of forwarding state from routers does not allow offering adaptive forwarding functionality.

III. Forwarding State Reduction in CCN

A. Overview

In this paper we present a forwarding scheme for CCN that combines stateful and stateless forwarding, so as to reduce the resource requirements of routers, without losing its advantages, i.e. multicast delivery, *address-less* hosts, detection of unwanted traffic and support for adaptive forwarding. Instead of storing forwarding state per Interest in either *all* or *none* of the routers, as in plain CCN and CONET, respectively, we propose tracking Interests at *some* of the on-path routers and using a mix of stateful (in-router) and Bloom filter-based stateless (in-packet) forwarding [8].

During Interest propagation, instead of updating the PIT at each router, the Interest is tracked at every d hops, where d is a predefined system parameter, e.g. $d = 3$ or $d = 4$. We call d the *Forwarding State Reduction Factor*. Intermediate routers add reverse path information inside Interests. When a router tracks an Interest, instead of storing the Interest's incoming interface, the router stores the reverse path (or tree) gathered by the Interest. During Data forwarding, routers that tracked a particular Interest place the source-route for the downstream path (tree) in the Data packet and push it towards the next stateful router(s). Between stateful points, packets are forwarded according to the in-packet source-route.

Our solution reduces forwarding state requirements, while preserving the desired properties of CCN's forwarding. Specifically, native multicast and host anonymity are preserved due to the adoption of Bloom filter-based forwarding for source-routing, while dismissal of unwanted traffic and adaptive forwarding is still supported for the fraction of Interests that each router is tracking. Though the latter two are supported in a more coarse manner, our approach compares favorably to either fully stateless forwarding solutions [7] or Interest compression schemes that drop fine-grained forwarding information [15]. Our forwarding scheme consists of two logical parts: (i) updating the PIT upon the arrival of an Interest and (ii) tracking reverse path information in Interests and using it in Data forwarding. We elaborate on these below.

B. Interest tracking

The main idea in our approach is to track an Interest at every d hops. We describe three *Interest tracking policies* that can achieve this.

1) Probabilistic tracking: In this policy, a router decides to track every incoming Interest with probability $1/d$. For example, when $d = 4$, an Interest is tracked with probability $\frac{1}{4} = 0.25$, thus routers track 25% of the received Interests. The probabilistic decision alone is not sufficient for two reasons. First, it obstructs the aggregation of Interests at common on-path routers in multicast applications. When an Interest reaches a router, the probability of *not* storing the Interest is $\left(\frac{d-1}{d}\right)$. Considering that $d \geq 2$, when a new Interest reaches a router where an Interest for the same content has already been stored, it is more probable that the new Interest will not be stored there, therefore it will not be suppressed. To avoid this, upon receiving an Interest, routers first check their PIT and proceed with the algorithm only if no match is found. The second inefficiency regards the probability of not tracking an Interest at all. In a N-hop path, the probability of *not* storing an Interest at any intermediate hop[2] is $\left(\frac{d-1}{d}\right)^N$. When $d = 4$ and $N = 8$, the probability of not storing an Interest at any router is $0.75^8 \approx 0.1$, i.e. 10% of the Interests issued by that particular application are source-routed on an end-to-end basis. As we will discuss later in Section III-C, this has a negative impact on the stateless forwarding part of our protocol due to *false positives* in Bloom filters: when the amount of source-routing information exceeds a limit, Bloom filter-based forwarding degenerates and causes excessive redundant traffic [8]. We therefore bound the number of stateless hops with the use of a *Hop Counter* (HC) placed in the Interest header. HC is increased at every hop and *reset* to 0 when a router decides to store the Interest. If HC reaches d, indicating the Interest was not stored for the previous d hops, the router *definitely* stores the Interest. Figure 2 shows the detailed algorithm for the Probabilistic tracking policy.

[2]We assume a unicast path, i.e. the same Interest has not previously crossed part (or all) of the path.

procedure PROB_TRACKING($interest, incoming\_port$)
  $pit\_entry := PIT\_lookup(interest)$
  if pit_entry *not null* **then**
    $storeInPIT(interest, incoming\_port)$
    $return$ ▷ Interest suppressed
  end if
  $hc := incrementHopCounter(interest)$
  $rand := random()$
  if $rand \leq (1/d)$ *or* $hc = d$ **then**
    $storeInPIT(interest, incoming\_port)$
    $resetHopCounter(interest)$
  end if
  $out\_port := FIB\_lookup(name)$
  $forward(interest, out\_port)$
end procedure

Fig. 2. Probabilistic Interest tracking policy.

2) Hash-based tracking: This policy tackles the *rendezvous* inefficiency of Probabilistic tracking, i.e. aggregating Interests for the same Data in common on-path routers, with the use of a hash function. When router *i* receives an Interest, it performs the following computation

$$\nu = hash(content\_name + suffix_i) \bmod d$$

If $\nu = 0$, we say that the Interest made a *rendezvous* at this node and the router stores it in the PIT. We hash the content name appended with a router-specific $suffix_i$, in order to produce a different ν for the same Interest at each hop. If we did not use the suffix, hashing the content name alone would produce the same result at all routers, thus the Interest would either be stored in all routers (if $\nu = 0$) or in none (if $\nu \neq 0$). If the selected hash function has good uniformity properties, ν is uniformly distributed in $[0, d-1]$ and Interests are stored at each router with probability $1/d$, as desired. If all routers use the same hash function (e.g. MD5) and each router uses a fixed $suffix_i$ (e.g. its MAC address), then multiple Interests for the same Data will *deterministically* rendezvous at the exact same routers.[3] Hence, routers will be able to suppress Interests without performing a PIT lookup. The cost of this policy is the hash computation at each router. As in Probabilistic tracking, the issue of not storing an Interest at all remains, thus we apply the hop counter-based upper bound here as well.

3) Hop Counter-based tracking: In this policy, routers track Interests based on the value of the *Hop Counter* (HC) inside the Interest header. HC is incremented at each hop and when $HC = d$, routers store the Interest in their PIT. Routers always perform an initial PIT lookup in order to suppress multicast Interests regardless of the HC value, as in Probabilistic tracking. The initial value for the HC is set by the issuing host but instead of setting it to 0, the initial HC is randomly selected in the range $[0, d-1]$ so as to distribute forwarding state to all routers. If the initial HC was always set to 0, routers with distance $d-1$ from hosts would be kept stateless. In the example of Figure 1, if $d = 3$, all Interests issued by hosts would be tracked by R_4, while R_1 to R_3 would have an empty PIT. PIT state would thus be unevenly distributed: R_4 would be a bottleneck point and R_1 to R_3 would not participate in (say) adaptive forwarding at all. In contrast, with a randomly selected HC, there is a $1/d$ probability for each on-path router to track the Interest.

[3]Note that the suffix is not communicated, thus host anonymity is preserved.

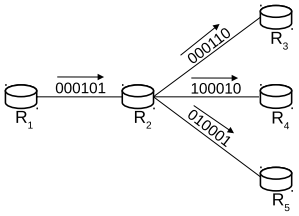


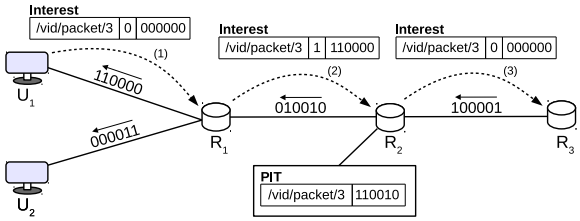
Fig. 3. An example of Bloom filter-based forwarding. Links are annotated with LIDs ($m = 6$ and $k = 2$). Only left-to-right LIDs are shown.

With all policies, the reduction of forwarding state is achieved at the cost of additional Interests in multicast applications, compared to CCN. That happens because Interests are not necessarily aggregated at the *first* common router of the multicast tree. For example, assume that U_1 and U_2 in Figure 1 consume the same content. If $d = 3$, their Interests will be aggregated at either $R1$, $R2$ or $R3$, although R_1 is the nearest common point. When Interests are aggregated at R_2, an additional Interest is transmitted, compared to CCN. For Interests that rendezvous at R_3, two additional Interests are transmitted. This is a penalty our scheme pays in order to reduce forwarding state in routers. The amount of additional Interests depends on the Forwarding State Reduction Factor d. For larger values of d, forwarding state is further reduced, but Interests may be suppressed further away than the first common on-path router. The overhead caused by additional Interests is also subject to the group size and the density of the multicast tree. When the multicast tree is sparse, Interests are rarely aggregated anyway, thus few additional Interests can be transmitted. Note that this penalty affects multicast delivery only; with unicast, there is no Interest aggregation, hence there is no such penalty. We further discuss the impact of increased Interest transmissions in Section IV.

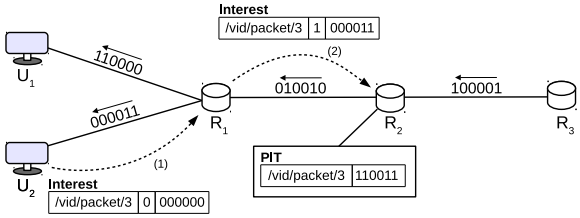
C. Semi-stateless data forwarding

We now describe how semi-stateless packet forwarding can be incorporated into CCN, without sacrificing the native multicast and node anonymity of CCN. In our scheme, the network supports multicast efficiently, even though the forwarding state may be stored in non-branching points of the multicast tree. That is, the source-routing scheme duplicates data *only* at branching points, even when the source-route is maintained elsewhere, without resorting to multiple unicast transmissions and without re-introducing host addresses to CCN.

In Bloom-filter based forwarding, the links of a delivery path (or *tree*) are encoded in a Bloom filter which is then placed as a source-route in the packet, hence the term *in-packet Bloom filter* (iBF). To encode path links in iBFs, links are assigned with a *Link Identifier* (LID). An LID is an m-bit string with only k bits set to 1 ($k << m$). The k bits are determined using k hash functions. LIDs are unidirectional (a bi-directional link is assigned two LIDs)



(a) Interest from U_1: R_1 updates the Interest iBF. R_2 tracks the Interest in its PIT with the iBF for $R_2 \rightarrow R_1 \rightarrow U_1$. R_2 resets the Interest iBF to 0 and further forwards the Interest.



(b) Interest from U_2: R_1 updates the Interest iBF. R_2 adds the iBF to its *existing* PIT entry and suppresses the Interest. The stored iBF contains the multicast tree to U_1 and U_2.

Fig. 4. Interest propagation using iBFs to track reverse paths. Only right-to-left LIDs are shown.

and need not be unique in the network. A delivery path is encoded in an iBF by ORing the constituent path link LIDs. The source-route is usually constructed *centrally*, either by the source node or by a separate routing module [8]. In Figure 3, the iBF for transmitting packets from R_1 to R_3 is $LID_{R_1 \rightarrow R_2} | LID_{R_2 \rightarrow R_3} = 000111$. In the forwarding plane, routers extract the iBF from packets, examine which of their outgoing links are part of the iBF and transmit the packet over those links. If the expression

$$iBF \& LID_i == LID_i$$

evaluates to true, then the router *assumes* that LID_i is part of the Bloom filter and transmits the packet over link i. For multicast delivery, we simply add the LIDs of all the tree links; the forwarding logic remains the same. In Figure 3, the iBF for multicasting packets from R_1 to $\{R_3, R_4\}$ is $LID_{R_1 \rightarrow R_2} | LID_{R_2 \rightarrow R_3} | LID_{R_2 \rightarrow R_4} = 100111$.

To integrate Bloom filter-based forwarding in CCN, we extend Interest and Data packets to carry an iBF in their headers. Interest packets accumulate the iBF for the traversed (reverse) path and Data packets carry the iBF for the delivery path (or tree). Specifically, upon receiving an Interest, routers update the Interest's traversed path by adding (ORing) the *outgoing* LID of the packet's incoming link. If a router decides to store an Interest in its PIT, it also stores the iBF and then *resets* the iBF in the Interest before further forwarding it. When the respective Data packet arrives, the router acts as a relay point by inserting the stored iBF in the Data packet and then forwarding it based on the iBF. Figure 4(a) shows an example where $d = 2$, U_1 and U_2 are two multicast users and the network uses Hop Counter-based tracking. At some point, U_1 requests the packet `/vid/packet/3` with initial $HC = 0$. U_1 creates the Interest with an *empty* iBF (i.e. all bits are set to 0) and transmits the packet. R_1 receives the Interest, increases the HC and adds the LID for the reverse direction, i.e. $LID_{R_1 \rightarrow U_1}$, to the Interest iBF (step 1). R_1 forwards the Interest to R_2. Node R_2 increases the HC to 2 and adds $LID_{R_2 \rightarrow R_1}$ to the Interest iBF (now containing the path $R_2 \rightarrow R_1 \rightarrow U_1$). Since $HC = 2$ $(= d)$, R_2 stores the Interest along with the iBF in its PIT (step 2). R_2 then resets the Interest's HC and iBF and forwards the Interest (step 3). This continues until the Interest reaches the data source.

During Interest forwarding, if a router finds a matching PIT entry, it adds the Interest's iBF to the iBF already stored in the PIT. The resulting iBF is the union of the already stored and the additional path links, which form a multicast tree. This is shown in Figure 4(b). U_2 transmits an Interest for the same content as U_1 did, with initial $HC = 0$. The request arrives at R_1 which updates the Interest's HC and iBF (step 1) and forwards the Interest to R_2. At that point, R_2 updates the Interest's iBF, adds it to the iBF already stored in the PIT and suppresses the Interest (step 2). The PIT entry at R_2 now contains the iBF for the multicast tree $R_2 \rightarrow R_1 \rightarrow \{U_1, U_2\}$.

Upon the arrival of a Data packet, a router checks its PIT and if a matching entry exists, it replaces the Data iBF with the stored iBF and further forwards the packet. If no PIT entry exists, the router forwards the Data packet according to its iBF. If no LID matches the Data packet's iBF, the router drops the packet. Finishing the example of Figure 4, when the Data packet `/vid/packet/3` arrives at R_2 , the router replaces the Data iBF with the one stored in the PIT. The iBF now contains $LID_{R_2 \rightarrow R_1}$, $LID_{R_1 \rightarrow U_1}$ and $LID_{R_1 \rightarrow U_2}$, therefore the packet is delivered to R_1 which then duplicates the Data packet to U_1 and U_2. It is important to note that even though the iBF is stored at a non-branching router (R_2), Bloom filter-based forwarding ensures that the Data packets are only duplicated at branching nodes (R_1).

Our forwarding scheme requires slight changes in the CCN architecture. Apart from the modified Interest and Data handling operations, the incorporation of Bloom filter-based stateless forwarding does not affect the architecture's control plane. That is, there is no need for any additional routing information exchange. Routers only need to know their own outgoing LIDs, which can be autonomously computed, e.g. by Double Hashing [16] the MAC address of the network interface during node bootstrap. There is no need to coordinate LID assignment, as LIDs do not need to be globally unique. In addition, source-routes are constructed in a distributed manner and no separate centralized module is required to construct the Bloom filters [17]. Nodes also remain anonymous, as in CCN. Security is not downgraded, as due to the Bloom filter-based and source-specific representation of the source-routes, it is very difficult to perform targeted attacks to nodes. Hosts are unaware of router LIDs and it is highly improbable that a host can *guess* a valid iBF to attack a particular node [8]. Content sources obtain valid iBFs only when Interests arrive at them, but they have no idea where these iBFs lead and they rarely obtain an iBF for an entire end-to-end path.[4]

There are three performance tradeoffs involved when incorporating Bloom filter-based forwarding in CCN. First, all

[4]The path has to be smaller than d hops (minus the initial value of HC in Hop Counter-based tracking).

TABLE I
GRAPH CHARACTERISTICS OF TOPOLOGIES USED IN EXPERIMENTS.

AS	1221	1755	3257	224
Nodes	104	87	161	74
Links	151	161	328	101
Diameter	8	11	10	9
Avg (Max) degree	3 (18)	4 (11)	4 (29)	3 (8)

packets (both Interests and Data) must carry iBFs, hence bandwidth overhead is increased due to the extra field in packet headers. Second, the PIT stores iBFs instead of interface ports. While iBFs are typically 128-256 bits long [8], up to x ports can be encoded with an x-bit mask, e.g. 32-bits for 32 ports; note that a full port mask, rather than a port number, is needed at the PIT in order to support multicasting. Hence, the actual memory reduction of the PIT is not equivalent to the reduction of PIT entries, e.g. a 50% reduction of the PIT entries does not lead to a 50% reduction in the actual memory footprint. Third, iBF-based forwarding is susceptible to false forwarding decisions which cause redundant traffic, especially as more LIDs are added to the iBF. The scale of this overhead depends on the size of the multicast group and the value of d.[5] We elaborate on these tradeoffs in Section IV.

IV. EVALUATION

A. Simulation setup

We evaluated the effectiveness of our approach through simulations, using ISP topologies obtained from Rocketfuel [18] and the Internet Topology Zoo [19]. Table I shows the graph characteristics of the tested topologies. Due to lack of space, we report results for topology 224, though we observed similar behavior in all tested topologies. In each test, we considered routers with a single link to be access routers and then attached 500 hosts in total, uniformly distributed across access routers. We tested our scheme with both unicast and multicast applications. The same experiments were performed with basic CCN, the results of which were used as a point of reference. Each experiment was repeated 20 times, changing the random generator seed in each repetition. In the experiments, iBFs are 16 bytes long ($m = 128$ bits) and LIDs are computed using Double Hashing [16] with $k = 4$. In all tests, routing information in routers (FIBs) was pre-populated, allowing Interests to reach content sources over the shortest paths, with hop count as the routing metric.

Our evaluation focused on the reduction of forwarding state in routers in terms of (a) the number of PIT entries and (b) the actual memory consumed by the PIT. For the memory footprint, we considered a hash table-based implementation which is reported to be the most suitable data structure for the PIT [14]. We assumed that memory pointers are 32 bits and that the interface ports in basic CCN were encoded with 32-bit masks. For the size of content names, we adopted the real-world measurements of [13] which reported two sizes: *small* content names that are *on average* 20 bytes and *large* content names which are *on average* 56 bytes. For multicast applications, apart from the reduction of forwarding state, we measured the bandwidth overhead due to (a) additional Interests caused by not storing PIT entries at the first common router on the multicast tree and (b) redundant Data caused by false positives in the Bloom filters.

[5]With unicast paths, the probability of false forwarding decisions is negligible for the values of d considered.

B. Unicast

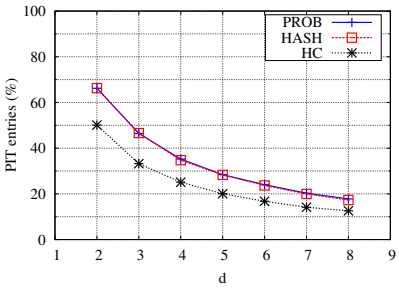
We simulated unicast delivery with a file transfer application, assuming that the majority of the requested content is located in a few (large) content servers located in the *center* of the network[6] and hosts download files from these sources. For simplicity, transfers are performed in a Stop-and-Wait fashion, i.e. hosts transmit one Interest at a time. We ran the simulation and took a snapshot of the network state at time $t = 1$ min while all connections are active, for both basic CCN and our scheme. The results are shown in Figure 5. We present results for the three Interest tracking policies: *PROB* for Probabilistic tracking, *HASH* for Hash-based tracking and *HC* for Hop Counter-based tracking.

Figure 5(a) shows the average number of PIT entries (tracked Interests) per router as a function of d, normalized against basic CCN. On average, HC reduces the number of PIT entries to $1/d$, that is, when $d = 3$, routers track $1/3 = 33.3\%$ of the forwarded Interests, i.e. a reduction of approximately 66%. When $d = 4$, the fraction of tracked Interests drops to 25%, i.e. a 75% reduction. PROB and HASH perform similarly to each other, but worse than HC, due to the upper bound of stateless hops that we set in order to limit the amount of path information in source-routes, which makes them maintain state at additional routers. Figure 5(b) shows the cumulative distribution (CDF) of the tracked Interests across routers for $d = 4$. In all three tracking policies, state is distributed evenly, avoiding state concentration points. Figure 5(c) shows the actual memory occupied by the hash table-based PIT for *small* and *large* content names as a function of d for HC, the best performing tracking policy, again normalized against basic CCN. The reduction of memory is not equivalent to the reduction of PIT entries as, for each Interest, the PIT stores a 16-byte iBF instead of a 4-byte (32-bit) port mask. Still, the reduction is noticeable: when $d = 3$, the PIT memory footprint is reduced to 46% (a 54% reduction) and to 39% (a 61% reduction) of basic CCN, for small and large content names, respectively. For $d = 4$, the PIT is reduced to 35% (a 65% reduction) and 30% (a 70% reduction) of basic CCN, for small and large content names, respectively.

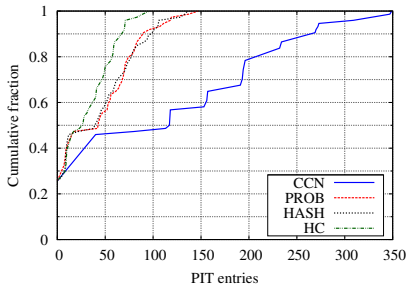
C. Multicast

We simulated multicast delivery with a live media streaming application. Sources transmit packets at a constant rate and hosts *consume* the live stream by proactively transmitting Interests for subsequent streaming packets [9]–[11]. In multicast applications, group sizes usually follow a Zipf-like distribution [20] and we adopted this in our experiments. For the Zipf distribution, we follow the methodology of [21]: the

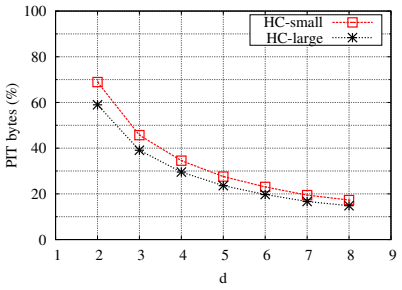
[6]Servers are attached to the router with the highest degree.



(a) Average PIT entries per router.

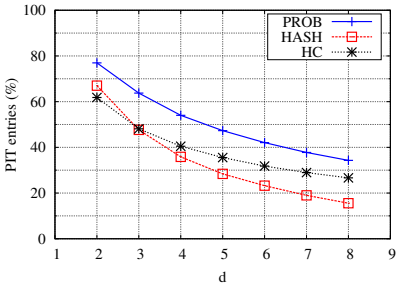


(b) Distribution of PIT entries (CDF) for $d = 4$.

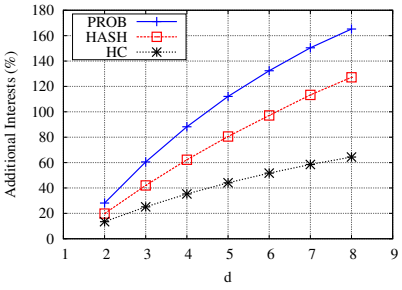


(c) Estimated PIT size (in bytes) for small (20-bytes) and large (56-bytes) content names with Hop Counter-based tracking.

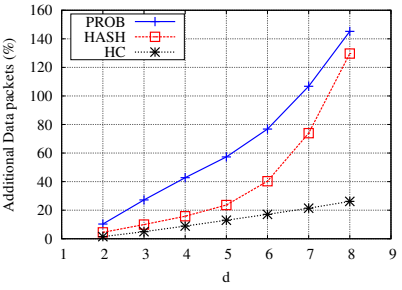
Fig. 5. Evaluation results for unicast applications. In (a) and (c) results are normalized against basic CCN.



(a) Average PIT entries per router.



(b) Additional Interests transmitted.



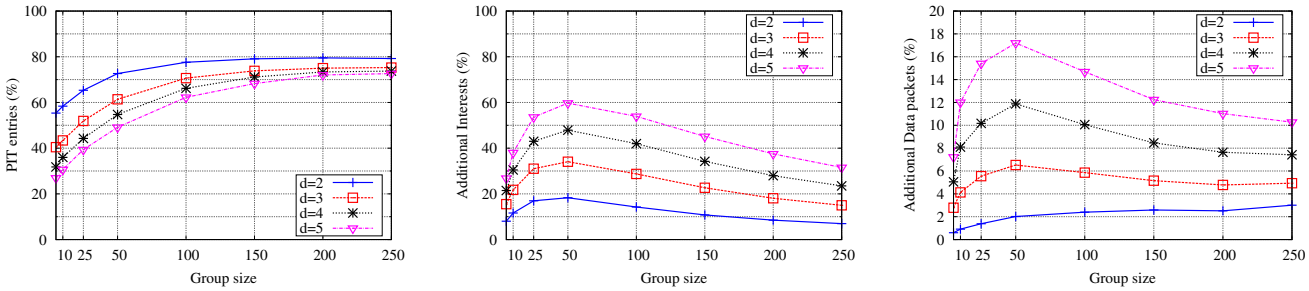
(c) Additional Data transmitted.

Fig. 6. Evaluation results for multicast applications. Group sizes follow a Zipf distribution. Results are normalized against basic CCN.

size of the i^{th} group is $group\_size(i) = \lfloor Ni^{\alpha} + 0.5 \rfloor$. We ran the experiments for $N = 1000$ multicast groups and set $\alpha = -0.51$ so that the smallest group size is 7. For each group, we ran the streaming application using both our scheme and basic CCN. In each experiment, we took a snapshot of the network state at $t = 5$ min, when the streaming applications have reached a steady state.

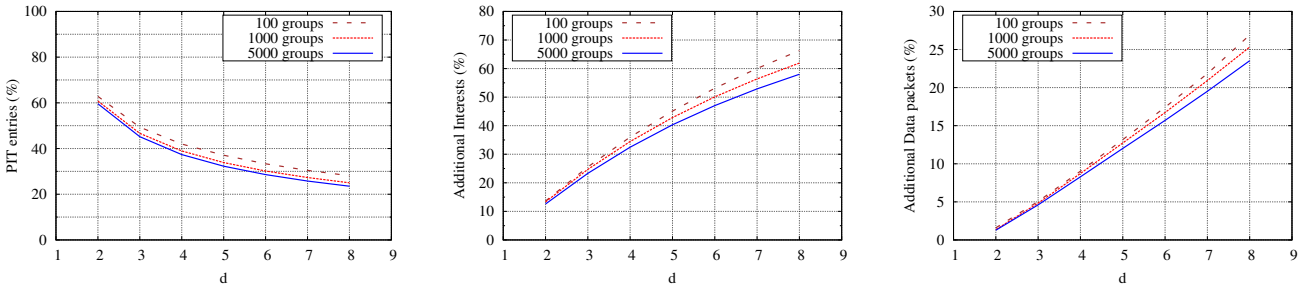
Figure 6 shows our scheme's performance, normalized against basic CCN. Figure 6(a) shows the average number of PIT entries (tracked Interests) per router as a function of d. Unlike with unicast, here the various Interest tracking policies have more pronounced differences, with HASH being the best performing policy. Figure 6(b) shows the fraction of additional Interests transmitted, normalized against basic CCN. In HC, checking the PIT at each hop, in conjunction with the random value of the initial hop counter, ensures that Interests are aggregated more effectively near the actual branching points of the tree, thus significantly reducing the additional Interests compared to the other two policies. Figure 6(c) shows the fraction of redundant Data due to false positives in Bloom filters, normalized against CCN. Since HC aggregates Interests more effectively than the other policies, fewer links are encoded in iBFs, resulting in less false positive forwarding decisions. The other two policies fail to aggregate Interests early enough, therefore too many links are stored in iBFs, thus producing more redundant traffic. As a result, even though the HC policy was not the best in terms of forwarding state reduction, it is the policy that creates the least amount of additional Interests and redundant Data. Overall, when $d = 3$, the HC policy *on average* reduces the amount of tracked Interests to 48% of CCN (i.e. a reduction of 52%) at the expense of 25% additional Interests and 5% additional Data. When $d = 4$, tracked Interests are reduced to 40% of CCN (i.e. a reduction of 60%) at the expense of 35% additional Interests and 9% additional Data.

In order to better understand the behavior of our scheme, we then analyzed the effectiveness of the HC policy with respect to group size, for various values of d. For this study, we re-ran the multicast experiments with group sizes following a *Uniform* distribution in order to get sufficient samples for all group sizes: we created $N = 1000$ multicast groups with sizes uniformly distributed in $[5, 250]$. For each group, we ran the multicast application for HC and basic CCN and took a snapshot of the simulation at time $t = 5$ min. We binned the results for presentation purposes and show them in Figure 7. With respect to PIT entries, Fig. 7(a) shows that HC works best with smaller groups; as group size grows, state eventually converges to a value depending on d. On the other hand, the fraction of additional Interests grows with group size up to a certain threshold, after which it starts decreasing, as shown in Fig. 7(b). In general, since the initial HC values are selected randomly, Interests may not be suppressed at the first common router, as described in Section III-B. This is a minor problem for very small groups where the multicast trees are relatively sparse and there exist only a few aggregation points anyway. As groups grow and the multicast tree becomes denser, the possibility of not storing state at aggregation points increases, and so do the additional Interests. As more users issue Interests



(a) Average PIT entries per router. (b) Additional Interests transmitted. (c) Additional Data transmitted.

Fig. 7. Evaluation results of the HC policy with respect to multicast group size. Results are normalized against basic CCN.



(a) Average PIT entries per router. (b) Additional Interests transmitted. (c) Additional Data transmitted.

Fig. 8. Evaluation results of the HC policy for various numbers of groups with a Zipf distribution of group sizes. Results are normalized against basic CCN.

for the same Data, the probability of aggregating an Interest closer to the first branching router increases, therefore, once the group size passes a threshold, the amount of additional Interests starts decreasing. The same pattern is observed for the redundant Data, as shown in Fig. 7(c). For very small group sizes, the multicast trees have very few aggregation points that could lead to redundant Data. As group size grows, larger trees are encoded in iBFs since Interests are not suppressed at the first branching router, increasing the number of false positives. Beyond a group size threshold, the multicast tree has become dense enough so that common Interests are suppressed early, hence fewer links are encoded in the iBFs. In addition, as group size increases, more links are added to the multicast tree, hence fewer links can produce false positives.

Overall, our scheme is more effective for small groups: it reduces PIT state more and it causes less bandwidth overhead. When group sizes follow a Zipf distribution, which is the common case, the vast majority of groups are small, thus the protocol's *average* behavior is heavily influenced by small groups, while the impact of large groups is much lesser. Hence, as the number of groups increases, the overall performance converges towards that of small groups. We illustrate this in Figure 8 which shows results from running the multicast experiments with 100, 1000 and 5000 groups with the group sizes following the Zipf distribution[7] for the HC policy. As the number of groups increases, the protocol's performance slightly improves. With 5000 multicast groups, when $d = 3$, PIT entries are reduced to 45% of CCN, i.e. a reduction of 55% (Fig. 8(a)), with 23% additional Interests (Fig. 8(b)) and

[7]We were able to simulate up to 5000 co-existing streaming applications.

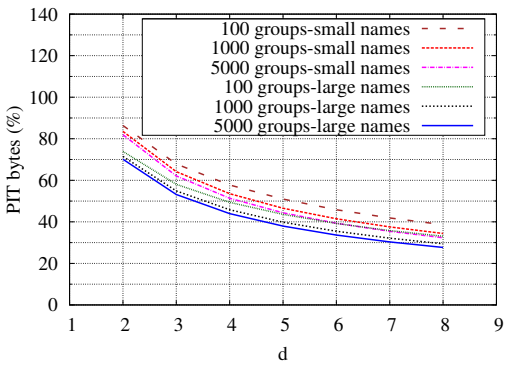


Fig. 9. Estimated PIT size (in bytes) for small (20-bytes) and large (56-bytes) content names with a Zipf distribution of group sizes in the HC policy. Results are normalized against basic CCN.

5% additional Data (Fig. 8(c)). When $d = 4$, PIT entries drop to 37% of CCN, i.e. a reduction of 63%, with 32% and 8% of additional Interests and Data, respectively.

Regarding actual table sizes, as opposed to PIT entries, Fig. 9 shows the size reduction of a hash table-based PIT for the HC policy with 100, 1000 and 5000 groups (with a Zipf distribution of group sizes) and the two content name sizes. When applications use large content names, for 5000 multicast groups and $d = 3$, the memory footprint drops to 55% of CCN (a 45% reduction), while for $d = 4$ the PIT occupies 45% (a 55% reduction) of the memory occupied by CCN. For applications using large content names, for 5000 multicast groups and $d = 3$, the memory footprint drops to 66% of CCN (a 34% reduction), while for $d = 4$ the PIT

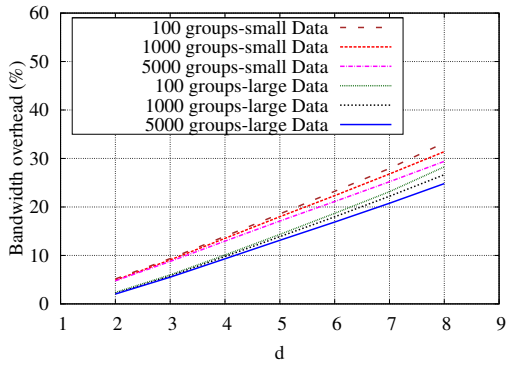


Fig. 10. Overall bandwidth overhead for large content names (70-byte Interests) with a Zipf distribution of group sizes in the HC policy. Data packets may use small (1500 bytes) or large (jumbo) frames (8000 bytes). Results are normalized against basic CCN.

occupies 55% (a 45% reduction) of the memory occupied by CCN.

As already discussed, with 5000 groups whose sizes follow a Zipf distribution, in the HC scheme 23%-32% of additional Interests and 5%-8% of redundant Data are generated compared to basic CCN, depending of the value of d ($d = 3$ or $d = 4$). In terms of actual bandwidth, however, the more numerous Interests are much smaller than Data, thus the overall bandwidth overhead depends on the size of Interests relative to Data. In Figure 10 we present the overall bandwidth overhead of the HC scheme in terms of the fraction of additional bytes transmitted compared to basic CCN. We assume large content names (56 bytes), so with the additional CCN meta-data, an Interest is on average 70 bytes long. We then consider two types of Data packets: a small Data packet that carries 1500 bytes of payload, targeting a CCN deployment over Ethernet, and a large Data packet that carries 8000 bytes of payload, targeting a CCN deployment over either Ethernet with jumbo frames or a UDP-based overlay. We also take into account the extra fields required by our scheme in the Interest and Data packet headers (iBF and HC). With 5000 groups (with sizes following a Zipf distribution), the overhead is 9% for small Data packets and 6% for large Data packets when $d = 3$, while with $d = 4$ the bandwidth overhead is 13% and 9% for small and large Data packets, respectively.

Overall, our evaluation results indicate that for multicast applications the memory requirements of a hash table-based PIT can be reduced to 45%-66% of basic CCN, at the cost of a 6%-13% bandwidth overhead compared to CCN, by using a moderate Forwarding State Reduction Factor, e.g. $d = 3$ or $d = 4$; smaller values may not effectively reduce forwarding state, while with larger values the bandwidth overhead begins to overshadow the state reduction achieved.

V. Conclusion and Future Work

We proposed a semi-stateless forwarding scheme for CCN that reduces the amount of forwarding state kept in routers by combining a mix of stateful (in-router) and Bloom filter-based stateless (in-packet) forwarding. Our protocol maintains the advantages of CCN's forwarding, i.e. support for multicast, enhanced security through address-less hosts, detection of unwanted traffic and support for adaptive forwarding. In addition, our solution requires few changes in the CCN architecture; apart from the extended Interest and Data forwarding operations, it requires no other extensions. A simulation-based evaluation over realistic ISP topologies showed that forwarding state can be reduced by 54%-70% in unicast applications, without any bandwidth penalties, while in multicast applications, forwarding state can be reduced by 34%-55% at the expense of a 6%-13% in bandwidth overhead.

We are particularly interested in improving the scalability of multicast in the CCN context, as our work in *Networked Music Performance* (NMP) relies on achieving ultra low delay communication between the participating musicians. While NMP applications on the current Internet are forced to direct all media streams to a centralized NMP server that redistributes them to all participants via unicast, in a CCN-based future Internet NMP participants would directly multicast media to each other, thus minimizing end-to-end delays without incurring excessive transmission costs [11]. Apart from applying our scheme to optimize NMP applications, our future work also involves studying adaptive scenarios where (i) routers employ our scheme only when they approach their memory limits and (ii) the value of d is individually selected by each router so as to minimize its own overhead.

Acknowledgment

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS - University of Crete - MUSINET. The authors would also like to thank Danai Argyrakopoulou for her comments and support.

References

[1] G. Xylomenos, C. Ververidis, V. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. Katsaros, and G. Polyzos, "A Survey of Information-Centric Networking Research," *IEEE Communications Surveys and Tutorials*, (to appear).

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of the ACM CoNEXT*, 2009, pp. 1–12.

[3] NDN project. [Online]. Available: http://named-data.net

[4] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, pp. 779 – 791, 2013.

[5] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proc. of the ACM SIGCOMM ICN Workshop*, 2011, pp. 44–49.

[6] H. Yuan, T. Song, and P. Crowley, "Scalable NDN forwarding: Concepts, issues and principles," in *Proc. of the IEEE ICCCN*, 2012, pp. 1–9.

[7] A. Detti, N. Blefari Melazzi, S. Salsano, and M. Pomposini, "CONET: a content centric inter-networking architecture," in *Proc. of the ACM SIGCOMM ICN Workshop*, 2011, pp. 50–55.

[8] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," in *Proc. of the ACM SIGCOMM*, 2009, pp. 195–206.

[9] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: voice-over content-centric networks," in *Proc. of the ACM ReArch Workshop*, 2009, pp. 1–6.

[10] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang, "ACT: audio conference tool over named data networking," in *Proc. of the ACM SIGCOMM ICN Workshop*, 2011, pp. 68–73.
[11] C. Stais, Y. Thomas, G. Xylomenos, and C. Tsilopoulos, "Networked music performance over information-centric networks," in *Proc. of the IEEE ICC IIMC Workshop*, 2013, pp. 647–651.
[12] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. Ramakrishnan, "COPSS: An efficient content oriented publish/subscribe system," in *Proc. of the ACM/IEEE ANCS*, 2011, pp. 99–110.
[13] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in named data networking," in *Proc. of the ACM/IEEE ANCS*, 2012, pp. 211–222.
[14] M. Varvello, D. Perino, and L. Linguaglossa, "On the design and implementation of a wire-speed pending interest table," in *Proc. of the IEEE INFOCOM NOMEN Workshop*, 2013.
[15] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "DiPIT: A distributed bloom-filter based PIT table for CCN nodes," in *Proc. of the IEEE ICCCN*, 2012, pp. 1–7.
[16] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better bloom filter," in *Proc. of the European Symposium on Algorithms*, 2006, pp. 456–467.
[17] C. Tsilopoulos and G. Xylomenos, "Scaling bloom filter-based multicast via filter switching," in *Proc. of the IEEE ISCC*, 2013.
[18] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *Proc. of the ACM SIGCOMM*, 2002, pp. 133–145.
[19] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.
[20] T. W. Cho, M. Rabinovich, K. Ramakrishnan, D. Srivastava, and Y. Zhang, "Enabling content dissemination using efficient and scalable multicast," in *Proc. of the IEEE INFOCOM*, 2009, pp. 1980–1988.
[21] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, 2002.