Adaptive Semi-Stateless Forwarding for Content-Centric Networks

Christos Tsilopoulos and George Xylomenos
Mobile Multimedia Laboratory, Department of Informatics
School of Information Sciences and Technology
Athens University of Economics and Business
Athens 10434, Greece
tsilochr@aueb.gr xgeorge@aueb.gr

***Abstract*—In Content-Centric Networks, users request content by issuing Interest messages, receiving Data messages in response. Interests leave state on routers on their path, to allow Data to follow the reverse path. Despite the advantages made possible by this state, its storage requirements raise scalability concerns. We propose an adaptive semi-stateless forwarding scheme where Interests are tracked only on a fraction of the routers. Between state-tracking routers, Interests gather reverse path information, which is used to deliver Data via Bloom filter-based forwarding. We show how the fraction of state-tracking routers can be dynamically adapted to reduce state while limiting communication overheads. Our scheme allows this state, along with all other required data structures, to comfortably fit into the fast memory of a practical router.**

***Index Terms*—ICN, CCN, PIT**

I. Introduction

The networking research community has spent considerable effort in *Information Centric Networking* (ICN) architectures which *primarily* facilitate content distribution by 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) [1] and its derivatives, where users request *named content* by issuing *Interest* packets and receiving in response the corresponding *Data* packets. Routers propagate Interests towards content sources, storing back pointers for each forwarded Interest in a *Pending Interest Table* (PIT). When Data are returned, routers push them towards their requester(s) based on information in the PIT [1]. If an incoming Data finds no match in the PIT, routers consider it unwanted and discard it. Routing and forwarding rely exclusively on content names, rather than host addresses.

The stateful name-based forwarding of CCN offers four key advantages. First, the network provides support for native multicast: if multiple users request the same content, their Interests are merged by common on-path routers, by adding back pointers to the same PIT entry, so as to later replicate the received Data [1]. While unicast applications rarely ask for the same content at the same time, multicast is ideal for live streaming applications, such as IPTV or Facebook Live. 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*) [1]. Fourth, maintaining per-packet forwarding state enables routers to realize *adaptive forwarding*, i.e., routers may actively participate in functions such as link failure recovery, flow control and detection of malicious behavior [2].

The need to track each forwarded Interest raises scalability concerns [3]. These have led to proposals for stateless schemes that remove the PIT [4], [5], [6], while other proposals adopt semi-stateless solutions that drop the per Interest forwarding information of CCN [7], [8], [9], [10], [11], [12]. This, however, undermines the advantages of CCN's stateful forwarding: (i) routers cannot aggregate Interests, thus multicast is not supported, (ii) host addresses are brought back in some schemes, (iii) routers cannot drop unwanted packets since forwarding state is removed, which is also why (iv) adaptive forwarding is prevented.

In this paper, we propose an adaptive semi-stateless forwarding scheme for CCN. Instead of tracking an Interest at either *all* or *none* of the routers, we store forwarding information on every d hops, thus each router tracks *on average* $1/d$ of the Interests. At intermediate hops, Interests collect reverse path information, which is stored at routers tracking that particular Interest. Data are later forwarded between routers tracking the corresponding Interest via Bloom filter-based stateless forwarding [13]. The resulting forwarding state reduction comes at the cost of increased bandwidth overhead for multicast (but not unicast) applications, caused by (i) additional Interest transmissions, as Interests may not be aggregated at the first common router and (ii) redundant Data transmissions, due to false positives in the Bloom filters. As this overhead can grow quite large in some topologies, we extend our solution from Tsilopoulos et al. [13] by dynamically adapting d, so as to balance state reduction against messaging overheads.

Despite this state reduction, our scheme preserves the advantages of CCN: native multicast and host/route anonymity are fully preserved, while routers can discard unwanted traffic and support adaptive forwarding for the fraction of Interests that they are tracking. Furthermore, only the Interest and Data forwarding of CCN need to be modified, leaving the control plane intact. In simulations using a wide range of realistic topologies, we found that forwarding state is reduced to $18.9\% - 27.4\%$ of CCN in unicast applications, with negligible bandwidth penalties, while in multicast applications

state is reduced to $27.3\% - 63.2\%$ of CCN, at the expense of $1.6\% - 12.1\%$ in bandwidth overhead. Therefore, with the small, but not negligible, amounts of multicast traffic that CCN could enable (e.g., $5 - 10\%$ of total traffic), our scheme manages to reduce the PIT to no more than 30% of its original size, thus managing to fit it, along with all other required data structures, in the fast memory of a practical router.

The remainder of this paper is organized as follows. In Section II we outline CCN forwarding and related work on reducing forwarding state. Section III presents our scheme in detail, assuming a fixed d, explaining how Interests are tracked and Data are forwarded. Our evaluation in Section IV shows that while PIT size is reduced, the messaging overhead can be large in some topologies. We therefore present a method for dynamically adapting d depending on tree density in Section V and show in Section VI that this approach balances the benefits and overheads of semi-stateless forwarding. We conclude in Section VII.

II. Background and related work

A. Content Centric Networking

All communication in CCN revolves around named data. Users issue *Interest* packets specifying a 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, e.g., `/a/b/c.mp4`. Interests are forwarded by routers hop by hop. At each hop, a router first checks its *Content Store* (CS) to see if a copy of the Data is available; if so, the Data packet is returned over the link that the Interest came through. Otherwise, the router checks its *Pending Interest Table* (PIT) to see if an Interest for the same Data has already been forwarded. If so, the router adds the Interest's incoming interface to the PIT entry and drops the Interest. Otherwise, the router stores the Interest and the interface it arrived from in the PIT, and forwards the Interest based on its *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 entries and transmit the Data packet through the appropriate interface; if multiple interfaces are listed, the Data packet is replicated, thus achieving multicast. Data packets that have no match in the PIT are discarded. After a Data packet is forwarded, the router considers the Interest satisfied and deletes the PIT entry. Timers are used to purge PIT entries that have not been satisfied for a long time. Essentially, routers maintain per packet state: as Interests are forwarded, breadcrumb-like trails are left in the PITs. These are consumed by Data packets as they reverse the path of the Interests. Figure 1 shows an example of CCN operation, where the Interests of three clients (U_1, U_2 and U_3) for content `/a/b/c.mp4` have been forwarded to a content source (S); the PIT entries form a multicast tree (from S to U_1, U_2 and U_3).

CCN's stateful forwarding offers four advantages. First, identical Interests are merged at the first common router, thus enabling Data to be returned via multicast. Although Interests

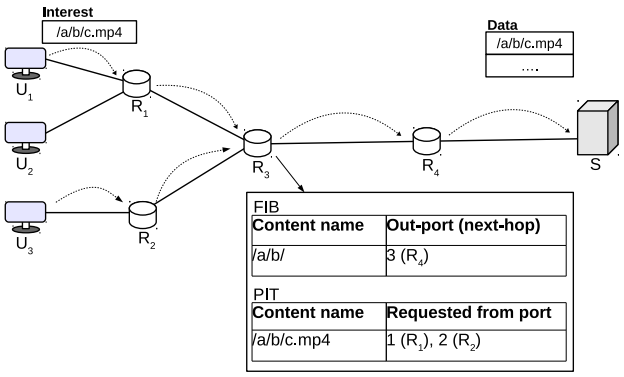


Fig. 1. CCN operation. Arrows show the Interests propagating towards content source S.

from unicast applications may benefit from this [14], native multicast is ideal for live streaming, where users consume content in a synchronized manner, e.g., IPTV or Facebook Live. Second, as no host addresses are used, a number of address-related problems (address space depletion, address assignment and governance) can be mitigated [1]. Third, routers drop Data packets that do not match a PIT entry, thus unwanted traffic (e.g., *spam*) is discarded early [1]. Fourth, per-packet state enables *adaptive forwarding* [2], where routers exploit forwarding state to assist functions such as fast recovery from link failures, congestion avoidance and early attack detection.

The amount of PIT state kept in routers, however, raises scalability concerns. The work in [15] mapped realistic IP traffic onto CCN and estimated that a 20 Gbps access router would require 1.5 M PIT entries.[1] Work in [16] estimated that, in an extreme worst case scenario, the PIT may reach 30–60 M entries. Since CCN names are variable-length and, in general, longer than host addresses, the total memory requirements for the PIT grow significantly. Although other work argues that PIT sizes are manageable, growing to no more than 2 M entries in a worst case scenario [17], it assumes flows with congestion-controlled Interests (as in TCP), large Data packets and bottlenecks at the access network only; unfortunately, Interests can be issued in advance at fixed rates [18], Data packets can be very small to reduce latency [19] and core links can become the bottleneck as access speeds grow [17], invalidating these assumptions.

A rough estimate of the number of PIT entries needed to fully utilize a link that only transmits Data packets is $bandwidth \times RTT / data\_packet\_size$, since PIT entries are active at least for the *Round-Trip Time* (RTT) to the content source and back. For example, to fully utilize a 40 Gbps link with 1500-byte Data packets and an average RTT of 80 ms, the PIT must hold around 266 K entries per link. Since each direction of a link normally carries both Interests and Data, with symmetric traffic and flow balance between Interests and Data, the PIT drops to 252 K entries [16]. Assuming an optimized PIT implementation, the actual memory footprint of this PIT would be 15.6–23 MB, depending on the size

[1] In that 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.

of the content names used [16]. Clearly, this cannot fit into a router's on-chip *static RAM* (SRAM) which is at most 4.5 MB but offers an access time of 1 ns. Although it is possible to fit such a PIT into off-chip SRAM, which has an access time of 4 ns and a size of up to 27 MB, we also need to fit in there the FIB and the index for the CS (the actual CS can reside in DRAM), therefore in a practical router implementation, the PIT should take up no more than one third of the off-chip SRAM [3]. This means that the PIT memory footprint must be reduced to at most 30% of its current size. Otherwise, we will need to use RLDRAM, whose size is only limited by cost, but has an access time of 15 ns; this is too slow to serve cache hits at a line rate of 40 Gbps [3].

B. Reducing PIT size

Since the PIT is checked for *every* arriving packet, it should reside in the line-cards' fast memory along with the FIB and CS, but as we saw above, it cannot even fit in the off-chip SRAM [3], [16]. Some alternative implementations can, under certain assumptions on the traffic mix and content-name length, substantially reduce the memory-footprint of the PIT. DiPIT [8] encodes multiple Interests in counting Bloom filters, thus losing information on individual Interests, which is crucial for dropping stale Interests [16] and adaptive forwarding. *Encode Name Prefix Trie* (ENPT) [15] organizes the PIT in a trie-like structure with *linear* ($O(N)$) insert and lookup costs (N is the number of name components), compared to the *constant* ($O(1)$) complexity of hash tables; the multiple memory accesses per packet make it too slow for high speed routers. Finally, by storing compact fingerprints of content names in the PIT, we can reduce its size by up to 50% [7]; not only this is insufficient to fit it into off-chip SRAM, but the collisions in the fingerprint space prevent Interest merging (and multicast).

Another way to reduce PIT state requirements is through *Persistent Interests* (PIs) [20]. With PIs, users send Interests for *channels* or *notifications*, grouped by a *prefix* of the content-name. Since a PI matches a number of Data packets, PIs are not deleted after a matching Data packet is forwarded; instead, they remain in the PIT until they expire or are explicitly revoked. The corresponding Data packets must be specially marked, so that forwarding can be performed on a prefix of their full name. If multiple users send PIs for the same prefix, CCN can group them into a single multicast tree. Although PIs can significantly reduce the number of Interests transmitted, the lifetime of a PI is much longer than that of a plain Interest, thus it is unclear whether the PIT size is reduced.

In the HyPOP scheme, a fixed-size PIT is used as a cache for the most popular Interests, supporting multicast only for the corresponding Data items [9]; non-popular content uses Bloom filters to perform source routing, inspired by our original semi-stateless scheme [13]. Since the PIT size is fixed, HyPOP's evaluation focuses on its cache hit ratio, ignoring the overhead due to redundant Data messages, which can be very high (see Section IV). In the CCN-GRAM scheme [10], routers adopt anonymous identities and instead of tracing Interests in the PIT, an *Anonymous Request Table* (ART) tracks the next hop to each identity, essentially creating a virtual circuit towards the requesting router; the router maintains a PIT-like structure to demultiplex Data to local clients. This negates most of the advantages of stateful forwarding, including native multicast support, as state is tracked per router. For multicast applications, a separate *Multicast ART* (MART) maintains the next hops for each group, as in IP multicast, as well as a *next message* counter; this is similar to the PI scheme, but Interests must be sent continuously to ask for the next message [11]. In addition to requiring applications to know whether they are unicast or multicast, the long lifetime of the MART entries may not reduce memory size requirements.

C. Making CCN stateless

Another approach to avoid the problems posed by the PIT, is to make CCN stateless. The CONET architecture, essentially a stateless variant of CCN, moves all forwarding information from the routers to the packet headers [4]. During Interest propagation, routers append their identifier in the packet's header. When the Interest reaches the content source, the header is reversed and placed as a source-route in the Data header. The pit/LESS proposal also includes the reverse path in the header, but instead of router identifiers, Interests encode all path links in a counting Bloom filter large enough to hold any unicast path without false positives, using entropy coding to keep filter sizes small; a PIT-like structure is used to demultiplex Data to local clients [5]. Another proposal is to use a routable name to identify the *originator* of an Interest, so that Data can be returned using the same FIB as for Interests [6]. Finally, another scheme uses a PIT to store the IP address for an Interest only when it enters the network or when it branches to multiple hops; at these points, the current router's IP address is stored in the Interest [12]. Data are returned by retracing this path, using the IP addresses in the PIT entries for routing.

In all these proposals, the motivation for the (partial or full) removal of the PIT is that the claimed advantages of CCN's stateful forwarding are not important enough to justify maintaining such state. Regarding Interest aggregation, analytical studies indicate that it is rare with unicast applications, as content that tends to be requested often is served by the CS without creating a PIT entry. Estimates on the actual fraction of merged Interests ranging from less than 1% to up to 5% [6], [14]. These studies, however, do not take into account the live streaming applications enabled by native multicast, such as IPTV or Facebook live. Assuming that video will remain the dominant type of traffic, at least for the foreseeable future, we would expect the adoption of CCN to increase the fraction of multicast traffic due to live video streaming, therefore CCN should be prepared to deal with a rate of Interest aggregation of up to 5–10%, primarily due to multicast. Regarding security, while it is true that the PIT is itself vulnerable to attacks [14], the stateless alternatives rely on global identifiers (e.g. router addresses, routable names, or even full paths), which lead us to the same problems that CCN meant to avoid by removing node identifiers: we lose node and path anonymity and the ability to drop unwanted traffic, since we can no longer track which Data

have been requested. Finally, regarding adaptive forwarding, the argument that routers do not have the autonomy to make their own decisions is not persuasive [6], since route flapping due to autonomic router decisions is a common problem with IP, and we would expect CCN-based schemes to exploit their content awareness to improve upon IP.

III. Semi-Stateless Forwarding

A. Overview

We now present a forwarding scheme for CCN that combines stateful and stateless operation, in order to reduce the resource requirements of routers without losing the advantages of statefull operation. Instead of storing per Interest state in either *all* or *none* of the routers, as in plain CCN or its stateless variants, respectively, we track Interests at *some* of the on-path routers, using a mix of stateful (in-router) and Bloom filter-based stateless (in-packet) forwarding [21].

During Interest propagation, instead of updating the PIT at each router, the Interest is tracked at every d hops, where d is a system parameter, e.g., $d = 3$ or $d = 4$. Non state-tracking routers add reverse path information inside Interests. When a router tracks an Interest, instead of storing the Interest's incoming interface, it 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 state-tracking routers, packets are forwarded according to the in-packet source-route.

Our solution reduces forwarding state, while preserving the properties of CCN. Native multicast, host and route anonymity are preserved due to the adoption of Bloom filter-based source-routing, while dismissal of unwanted traffic and adaptive forwarding is supported for the fraction of Interests that each router tracks. Though the latter are supported in a more coarse manner, our approach compares favorably to both fully stateless [4], [5], [6] and semi-stateless solutions that drop fine-grained forwarding information [7], [8], [9], [10], [11], [12].

Our scheme has two parts: (i) updating the PIT on the arrival of an Interest and (ii) tracking reverse path information in Interests and using it for Data.

B. Interest tracking

In previous work, we considered three different *Interest tracking policies* [13]. In *probabilistic tracking*, each router randomly chooses to store $\frac{1}{d}$ of the incoming Interests in the PIT. In *hash-based tracking*, each router hashes the name in the Interest concatenated with a fixed local suffix and if the result is divisible by d it stores the Interest in the PIT. Finally, in *hop counter-based tracking* each Interest includes a *hop counter* (HC) which counts the hops until the Interest must be stored in the PIT. The first two policies, due to their probabilistic nature, also require a HC to prevent very long paths between routers tracking an Interest. For this reason, this paper only considers the third policy, which, although simple, has been shown to offer good overall performance [13].

procedure HC_Track($interest, incoming\_port$)
  $pit\_entry := PIT\_lookup(interest)$
  if pit_entry *not null* **then**
    $store\_in\_PIT(interest, incoming\_port)$
    $return$ ▷ Interest suppressed
  end if
  $hc := increment\_hop\_counter(interest)$
  if $hc = d$ **then**
    $store\_in\_PIT(interest, incoming\_port)$
    $reset\_hop\_counter(interest)$
  end if
  $out\_port := FIB\_lookup(name)$
  $forward(interest, out\_port)$
end procedure

Fig. 2. Hop Counter-based Interest tracking.

In the hop counter-based policy, an HC is stored inside the Interest header and is incremented at each hop. When $HC = d$, routers store the Interest in their PIT and reset the HC to 0. To ensure that Interests for the same Data will be merged, upon receiving an Interest, routers first check their PIT and proceed with the HC check only if no match is found; otherwise, they merge the new Interest into the existing PIT entry and drop it. The initial value for the HC is randomly selected by the issuing host 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 $d-1$ or less from hosts would be kept stateless. In the example of Figure 1, if $d = 3$, all Interests would be tracked by R_4, making it a bottleneck, while R_1 to R_3 would have an empty PIT. With a randomly selected HC, there is a $1/d$ probability for each on-path router to track the Interest. Figure 2 shows the detailed algorithm for the Hop Counter-based tracking policy.

C. Data forwarding

We now describe how semi-stateless packet forwarding can be incorporated into CCN, without sacrificing native multicast or host and route anonymity. In our Bloom-filter based forwarding scheme, the network duplicates multicast data *only* at branching points, even when the route is maintained elsewhere. To encode the set of links representing a path (or *tree*), each link is assigned a *Link Identifier* (LID), which 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) and need not be unique in the network. Each LID is in itself a Bloom filter representing a set with itself as the only member. A set of links representing a delivery path is encoded in a Bloom filter by ORing the constituent path link LIDs [21]. We then place this Bloom filter in a packet's header and call it an *in-packet Bloom filter* (iBF). For example, in Figure 3, the Bloom filter for transmitting packets from R_1 to R_3 is $LID_{R_1 \to R_2} | LID_{R_2 \to R_3} = 000111$; this is used as the iBF of these packets. During forwarding, routers extract the iBF from packets and examine which of their outgoing links are part of the iBF. If the expression $iBF \ \& \ LID_i == LID_i$ is true,

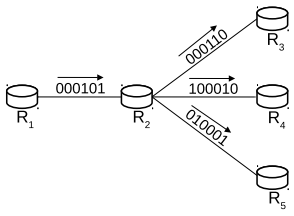


Fig. 3. Bloom filter-based forwarding. Links are annotated with LIDs ($m = 6$ and $k = 2$).

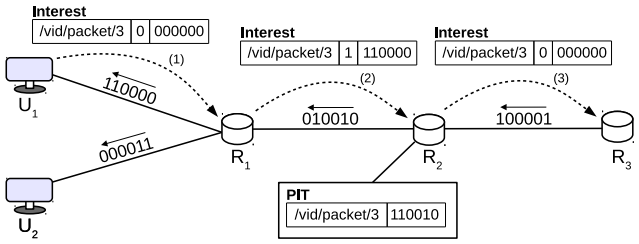


Fig. 4. Interest from U_1: R_1 updates the Interest iBF. R_2 tracks the Interest with the iBF for $R_2 \rightarrow R_1 \rightarrow U_1$. R_2 resets the iBF to 0 and forwards the Interest.

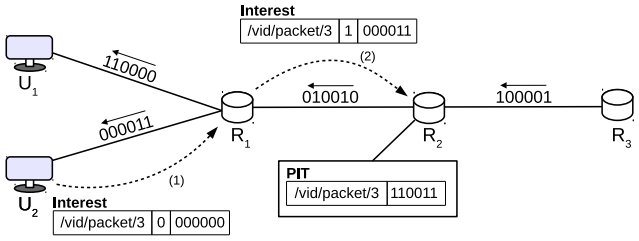


Fig. 5. Interest from U_2: R_1 updates the Interest iBF. R_2 adds the iBF to its *existing* PIT entry and drops the Interest. The stored iBF contains the tree to U_1 and U_2.

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 to the iBF the LIDs of all the tree links; the forwarding logic remains the same, but a node can expect multiple outgoing links to match an iBF. For example, 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. 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 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 shows an example where $d = 2$ and U_1 and U_2 are two multicast users. First, U_1 requests `/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

procedure SEMI_STATE_FWD($data, incoming\_port$)
  $pit\_entry := PIT\_lookup(data.name)$
  if pit_entry *not null* **then**
    $data.iBF := pit\_entry.iBF$
  end if
  $iBF\_Forward(data, incoming\_port)$
end procedure
procedure IBF_FORWARD($data, incoming\_port$)
  for $port$ *in* all_ports **do**
    if $port = incoming\_port$ **then**
      $continue$
    end if
    $lid_{port} := link\_id(port)$
    if $data.iBF$ & $lid_{port} = lid_{port}$ **then**
      $forward(data, port)$
    end if
  end for
end procedure

Fig. 6. Semi-stateless Data forwarding.

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 simply adds the Interest's iBF to the one in the PIT. This is shown in Figure 5, where 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 it to R_2. R_2 updates the Interest's iBF, adds it to the iBF stored in the PIT and suppresses the Interest (step 2). The PIT at R_2 now contains the iBF for the 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 it. Finishing the example of Figures 4 and 5, 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 it to U_1 and U_2. Note that even though the iBF is stored at a non-branching router (R_2), Bloom filter-based forwarding ensures that Data packets are only duplicated at branching nodes (R_1). Figure 6 shows the Data forwarding algorithm.

Our forwarding scheme requires only 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 control plane, as there is no need for any additional routing information exchange. General PIT behavior, including timeouts, is exactly

the same as in plain CCN. Routers only need to know their own outgoing LIDs, which can be autonomously computed, e.g., by Double Hashing [22] of 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. Source-routes are constructed in a distributed manner, in contrast to other Bloom filter-based forwarding schemes which require a centralized routing module [21]. 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 [21]. 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 end-to-end path[2], thus preserving route anonymity.

D. Performance tradeoffs

There are five performance tradeoffs involved when incorporating Bloom filter-based forwarding in CCN. First, all Interest and Data packets must carry iBFs, hence bandwidth overhead is increased due to the extra field in packet headers. Second, the PIT stores iBFs instead of interface identifiers (or, *ports*). While iBFs are typically 128–256 bits long [21], up to x ports can be encoded with an x-bit map, e.g., 32-bits for 32 ports; the full port bitmap is needed to support multicasting. Hence, the actual reduction in the memory footprint of the PIT is not equivalent to the reduction in PIT entries. 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.[3] Fourth, multicast applications suffer from redundant Interest transmissions, since 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 $R1$, $R3$ or $R4$, although R_1 is the nearest common point. When Interests are aggregated at R_3, one extra Interest is transmitted, while for Interests that rendezvous at R_4, two extra Interests are transmitted. The additional Interests depend on d and tree density: as d grows, Interests may be suppressed further from the optimal point; when the tree is sparse, Interests are rarely aggregated, thus extra Interests are also rare. Fifth, our scheme adds a very small per packet processing overhead (2–3 add/compare operations), but as it reduces the number of (costlier) insert/remove operations on the PIT, it should actually reduce the average packet processing delay.

IV. Evaluation of the basic scheme

A. Simulation setup

We evaluated the effectiveness of our approach through simulations, using synthetic scale-free graphs generated with the Barabási-Albert algorithm [23] and ISP topologies obtained from Rocketfuel [24] and the Internet Topology Zoo [25]. Table I shows the graph characteristics of the tested topologies. For each one, we considered the graph to represent the backbone network of a *Content-Delivery Network* (CDN) provider. Nodes with a single link ($degree = 1$) are considered to be gateways providing access to local ISPs. We attached 5 additional nodes in each access gateway, each representing the aggregate demand generated by one ISP. In our experiments, iBFs are 16 bytes long ($m = 128$). LIDs are computed autonomously by each router: a router i uses Double Hashing [22] to generate the k_i hash functions over the value $k_i = \lceil log_2(deg_i - 1) \rceil$ as in [26], where deg_i is the degree of node i. In all tests, routing information (the FIB) was pre-populated, allowing Interests to reach content sources over the shortest paths, with hop count as the metric.

TABLE I
Graph characteristics of the experimental topologies.

Topology	Nodes	Access	Links	Diameter	Avg (Max) deg.
AS-20965	40	8	61	8	3 (10)
AS-224	74	15	101	9	2.7 (8)
AS-3967	79	7	147	10	3.7 (12)
AS-1755	87	10	161	11	3.7 (11)
AS-1221	104	51	151	8	2.9 (18)
AS-6461	138	9	372	8	5.4 (20)
scale-free-50	50	24	62	6	2.5 (18)
scale-free-100	100	46	133	7	2.6 (33)

We tested our scheme using the same application for both unicast and multicast: a HTTP-like live streaming application where a server generates data chunks at a constant rate and clients request Data packets in a Stop-and-Wait fashion. For unicast, a single client downloads the content, while for multicast many clients simultaneously download it. Both server and client(s) are located at the edges of the graph, i.e., inside regional ISPs, and are randomly placed in each experiment. Experiments last until 1000 Data packets are delivered. For each topology t we ran our application with group sizes ranging from 1 to C_t receivers, where C_t is the number of edge nodes (regional ISPs) attached to the backbone's access nodes. Each experiment was repeated 20 times, changing the random generator's seed, thus selecting different server and client(s). Unless otherwise indicated, results are from the AS-20965 topology, and are representative of all topologies. Detailed results from all topologies are provided in [27].

B. Evaluation metrics

We focused on the reduction of forwarding state in routers in terms of (a) PIT entries and (b) actual memory footprint. We assumed a hash table-based implementation, reportedly the most suitable data structure for the PIT [16]. We assumed 32-bit memory pointers and interface ports encoded with 32-bit maps. For the size of content names, we adopted the real-world measurements of [15] which reported two sizes: small content names, 20 bytes on average, and large content names, 56 bytes on average. Apart from the reduction of forwarding state, we also measured the bandwidth overhead due to (a) additional

[2]The path has to be smaller than d hops, minus the initial value of HC.

[3]With unicast paths, the probability of false forwarding decisions is negligible for the values of d considered, as we will show in Section IV.

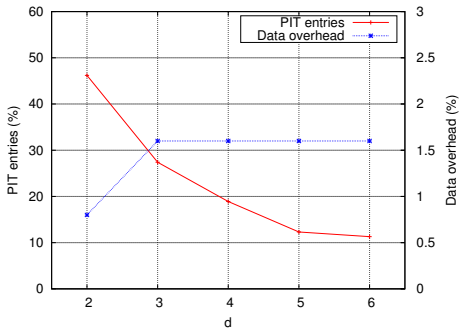


Fig. 7. PIT entries and Data overhead for unicast traffic as a function of d.

Interests caused by not storing PIT entries at the first common router and (b) redundant Data caused by false positives in the Bloom filters. Note that additional Interests can only occur with multicast scenarios, while redundant Data may occur in both unicast and multicast. All results shown are normalized against the basic CCN behavior, which is the performance baseline.

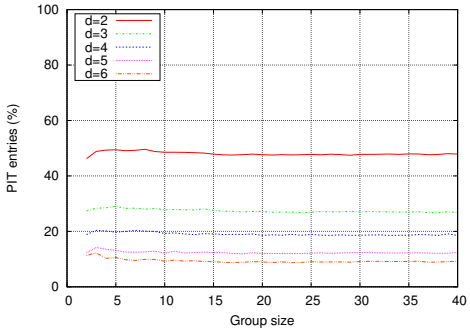
C. Unicast

Figure 7 shows the performance of our scheme for unicast flows. The left axis shows the average number of PIT entries in each router normalized against baseline CCN as a function of d. For $d = 3$, the PIT stores 27.4% of the entries compared to basic CCN, while for $d = 4$, it drops to 18.9%. The right axis shows the amount of additional Data packets transmitted due to false positives, with respect to d. Data overhead is very low, below 1.7% in the worst case, since the number of links inserted in unicast Bloom filters (up to $d - 1$) is quite small and very rarely leads to false positives. The stepwise behavior of Data overhead, which is evident in all topologies tested (see [27]), is due to the fact that nodes use the same LIDs for their attached links in all experiments. As a result, if a false positive appears with $d = 2$, it will reappear with any higher d. In our case, one false positive appears with $d = 2$ and a second one appears with $d = 3$, but no more false positives appear with d up to 6. It is clear that with unicast traffic only, our scheme can easily fit the PIT into a realistic router, by reducing its footprint to no more than 30% of baseline CCN.

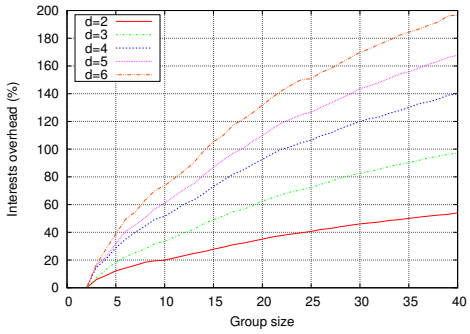
D. Multicast

For multicast, we first examine performance with respect to group size, and then consider different group size distributions, before looking at overall memory footprint and communication overhead. In each run, both the server and all clients are randomly placed at edge nodes of the network (see Section IV-A).

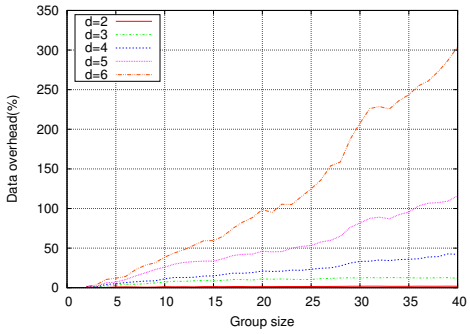
1) The effect of group size: Figure 8 shows the performance of our scheme with respect to multicast group size, for various values of d. Figure 8(a) shows that the gains in the number of PIT entries do not depend on group size, only on d. On the other hand, additional Interests grow with group size, as shown in Figure 8(b), as well as with d. As discussed in Section III-D, as groups grow and/or multicast trees become dense, the possibility of not storing state at the optimal points increases,



(a) PIT entries



(b) Interests overhead



(c) Data overhead

Fig. 8. PIT entries, Interests and Data overhead depending on d against group size.

and so do the additional Interests. For the redundant Data, shown in Figure 8(c), the effect of d is even more pronounced. As the group size grows, larger trees are encoded in iBFs (with tree height up to $d - 1$), increasing the number of false positives. Overall, there is a common pattern: semi-stateless forwarding is more effective for small groups, as it causes less bandwidth overhead in terms of redundant Interests and Data packets compared to baseline CCN, with d determining the balance between PIT gains and communication overhead. To understand aggregate performance then, we need to make some assumptions on the distribution of group sizes.

2) Uniform distribution of group sizes: Let us first assume that multicast group sizes follow a uniform distribution. For each topology t, we took into account all experiments of multicast groups with $[3, C_t]$ participants,[4] where C_t is the number of edge nodes (regional ISPs) attached to the backbone's

[4]The smallest multicast group consists of 1 sender and 2 receivers, while the largest multicast group consists of 1 sender and $C_t - 1$ receivers.

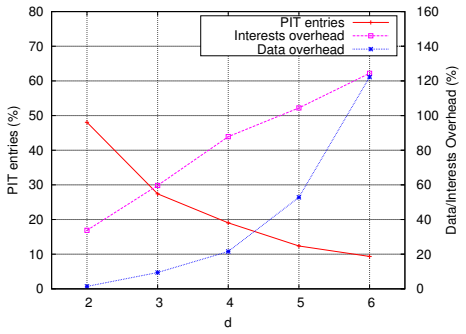


Fig. 9. PIT entries, Interests and Data overhead against d (uniform group sizes).

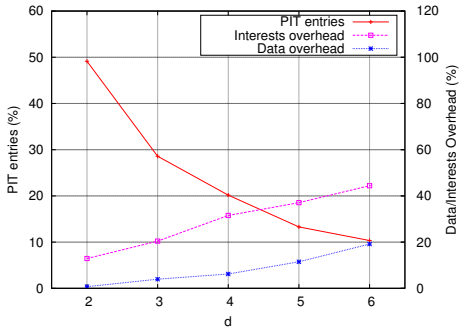
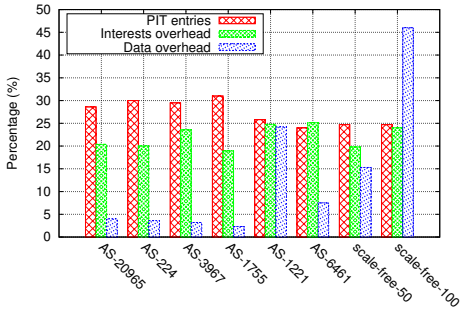
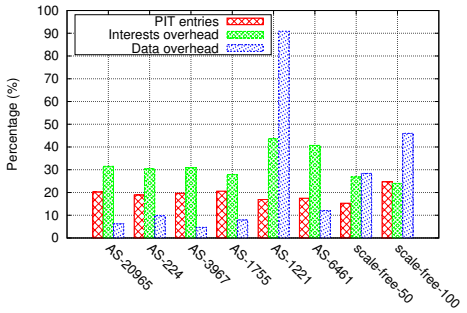


Fig. 10. PIT entries, Interests and Data overhead against d (Zipf group sizes).



(a) $d = 3$



(b) $d = 4$

Fig. 11. PIT entries, Interests and Data overhead for all topologies (Zipf group sizes).

access nodes. Recall that, for each group size, each experiment was repeated 20 times, randomly choosing different sender and receivers. Therefore, for each topology t, we examine the average system behavior of $20 * (C_t - 2)$ experiments with group sizes uniformly distributed in $[3, C_t]$. Figure 9 shows performance with respect to d, with PIT entries on the left axis and Interests/Data overhead on the right axis. PIT state is reduced as d grows, while both types of communication overhead grow: while Interests overhead is much higher with smaller d, Data overhead also becomes significant with higher d. For $d = 3$, our scheme reduces the average PIT state to 27% compared to basic CCN, at the cost of 60% additional Interests and 9% additional Data transmissions. For $d = 4$, the average number of entries is reduced to 19%, at the cost of 88% additional Interests and 22% additional Data. Clearly, the overheads are significant when groups sizes are uniform.

3) Zipf distribution of group sizes: Let us now assume that multicast group sizes follow a Zipf-like distribution, a more realistic scenario [28]. For the Zipf distribution, we follow the methodology of [29]: for each tested topology t, the size of the i^{th} group is $group\_size(t, i) = \lfloor i^{-\alpha_t} * C_t + 0.5 \rfloor$ where C_t is the maximum group size in topology t (equal to the number of ISPs), and α_t is selected so that the smallest multicast group size is 3 (one server and two receivers). Figure 10 shows the performance of our scheme with respect to d. The patterns are similar to the previous case, but the results are far more encouraging, as the transmission overheads are manageable for moderate values of d. When $d = 3$, PIT entries are on average reduced to 29% at the cost of 20% additional Interests and 4% additional Data transmitted. For $d = 4$, the PIT is on average reduced to 20% at the cost of 32% additional Interests and 6% additional Data transmitted.

Although the PIT is reduced equally in both group size distributions, there are significant differences in the communication overheads. Our scheme provides a significant PIT reduction with relatively manageable bandwidth overheads when group sizes follow a Zipf distribution. More worrying is the fact that performance also depends on network topology, with some topologies being especially problematic. Figure 11 shows performance data for all topologies for $d = 3$ and $d = 4$ with a Zipf distribution of group sizes. In most topologies, the Data overhead is below 10%, while the Interests overhead is around 30%. However, the costs in *AS-1221*, *scale-free-50* and *scale-free-100* are quite high.

4) PIT memory footprint: We now assess the actual memory size reduction of the PIT. Recall from Section III-D that our PIT contains fewer entries, but each entry occupies more memory due to the need to store a Bloom filter instead of a port mask. Figure 12 shows the size reduction of a hash table-based PIT as a function of d for AS-20965. For $d = 3$, the actual memory footprint for the PIT is reduced to 39% for small content names and 34% for large content names. For $d = 4$, the memory footprint for the PIT is reduced to 28% for small content names and 24% for large content names. The results across all topologies (not shown) are $28.3\% - 42.6\%$ of basic CCN for $d = 3$ and $15.1\% - 28.3\%$ for $d = 4$.

5) Bandwidth overhead: As already discussed, PIT state reduction comes at the cost of additional Interest and Data transmissions with multicast. Most of these are Interests not

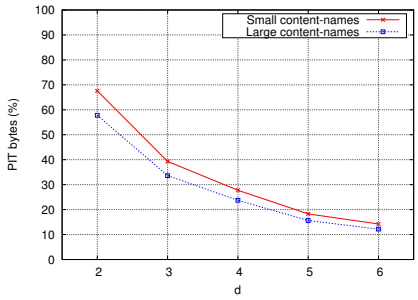


Fig. 12. PIT size reduction (in bytes) for small (20-bytes) and large (56-bytes) content names (Zipf group sizes).

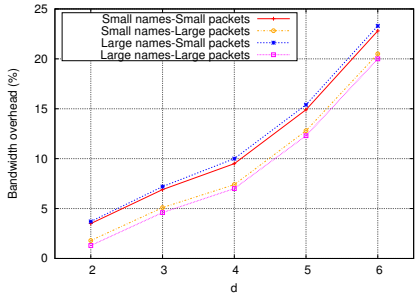


Fig. 13. Bandwidth overhead for small (36-byte Interests) and large content names (70-byte Interests), with small (1500 bytes) and large (7500 bytes) Data (Zipf group sizes).

aggregated at the optimal point, while the rest are Data transmitted due to false positives in the Bloom filters. In terms of actual bandwidth, however, as Interests are much smaller than Data, the overhead depends on the size of Interests relative to Data. In Figure 13 we present the overall bandwidth overhead in topology AS-20965, in terms of the fraction of additional bytes transmitted compared to basic CCN. We assume small (20 bytes) and large (56 bytes) content names, so with the additional CCN meta-data, Interests are on average 36 and 70 bytes long, respectively. 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 7500 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 in the Interest and Data packet headers (iBF and HC). When $d = 3$, the additional bandwidth is $4.6\% - 7.2\%$, while for $d = 4$ the additional bandwidth is $7\% - 10\%$, both quite reasonable considering the corresponding gains in PIT size. The results across all topologies (not shown) are $1.1\% - 11\%$ of basic CCN for $d = 3$ and $5.4\% - 16.2\%$ for $d = 4$, if we exclude the problematic topologies (*AS-1221*, *scale-free-50* and *scale-free-100*); in these topologies, overheads are up to 48.6% with $d = 3$ and 172.8% with $d = 4$, which is clearly an issue.

E. Unicast - Multicast traffic mix

In order to evaluate the overall benefits and costs of semi-stateless forwarding, we need to consider a mix of unicast and multicast traffic. Figure 14 shows the overall PIT size

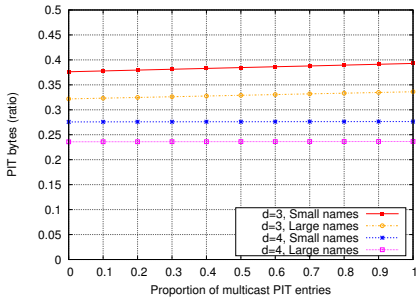


Fig. 14. PIT size reduction (in bytes) depending on the fraction of multicast Interests.

reduction for small and large names depending on the fraction of multicast traffic. A value of 0.0 in the x-axis means that 0% of PIT entries in each router are generated by multicast applications while a value of 1.0 means that 100% of PIT entries are generated by multicast applications. Although our method reduces the PIT more effectively for unicast applications, there are no significant differences with increasing fractions of multicast traffic.

In a similar manner, we computed the bandwidth overheads as a function of the traffic mix (not shown). The traffic mix plays a more important role on bandwidth overheads, which roughly double when all traffic is multicast. For $d = 3$, the PIT size in AS-20965 is reduced to $33\% - 40\%$ of regular CCN at a cost of $2\% - 7\%$ additional bandwidth, while for $d = 4$, the PIT requires $24\% - 28\%$ of the memory of regular CCN at a bandwidth cost of $2\% - 10\%$, as the fraction of multicast interests ranges from 0% to 100%. For the expected fraction of multicast traffic (up to $5-10\%$), our scheme achieves the goal of reducing PIT size to no more than 30% of baseline CCN with very low overhead costs. However, with some topologies (*AS-1221*, *scale-free-50* and *scale-free-100*), the overheads can become problematic even with that level of multicast traffic; with unicast traffic only, the overheads are negligible in all topologies.

V. Adaptive Semi-stateless Forwarding

A. Overview

As shown in Section IV, the communication overheads of our semi-stateless scheme can be problematic with some topologies; in addition, for a given topology, they become worse as multicast group size grows. The root cause of both these problems is tree density: as multicast trees become denser, more links need to be encoded in each iBF, leading to more Data transmissions due to false positives; there are also more opportunities to store PIT state at non-branching tree nodes, leading to more Interest transmissions. Some topologies have high degree nodes, making such problems unavoidable; in others, the problem only appears as group size grows. A *relatively* large d, for example $d = 3$ or $d = 4$, favors sparse trees; PIT size is significantly reduced with almost negligible costs in bandwidth overhead. For denser trees though, a smaller d, for example $d = 2$, avoids excessive bandwidth overheads, at the cost of smaller gains in PIT size. Ideally, instead of setting a fixed value for d, we should set d depending on tree

```
procedure HC_TRACK_AD(interest, incoming_port)
    d := extract_d(interest)
    pit_entry := PIT_lookup(interest)
    if pit_entry not null then
        d_min := min(d, pit_entry.d)
        store_in_PIT(interest, d_min, incoming_port)
        return                          ▷ Interest suppressed
    end if
    hc := increment_hop_counter(interest)
    if hc = d then
        store_in_PIT(interest, d, incoming_port)
        reset_hop_counter(interest)
    end if
    out_port := FIB_lookup(name)
    forward(interest, out_port)
end procedure
```

Fig. 15. Hop Counter-based Interest tracking with adaptive d.

density. This is not easy within the architectural context of CCN, where routers have no global knowledge of topology or routing. It is even more difficult if we consider that density may vary at different areas of the same tree.

Since the dominant overhead is redundant Data transmissions, a simple way to reduce overhead is to limit the number of links inserted into the iBFs. Previous studies have shown that the *false positive probability* (fpp) of iBFs must be kept below 0.5% [26]. The fpp is equal to $= (1 - e^{-kn/m})^k$ where m is the Bloom filter size (bits), k is the number of hash functions and n is the number of items inserted (LIDs). While m and k are known to each router, n is not, since Interests carry an accumulated iBF. Fortunately, the fpp can be estimated by examining an iBF's *fill factor*, the portion of bits set to 1, as follows [30]:

$$fpp = (fill\_factor)^k = (\frac{\#\ bits\ set}{m})^k$$

A router can thus estimate an iBF's fpp by counting the number of bits set to 1. For $k = 5$ or $k = 6$, we can infer that an iBF is congested when its $fill\_factor$ exceeds 0.4. We will exploit this to extend our scheme with the *dynamic* adaptation of d, based on the iBFs constructed in the network.

At a high level, our *adaptive* semi-stateless scheme tries to reduce PIT size, until it detects that redundant traffic thresholds may have been violated. The issuing host sets d on a per-Interest basis, starting with a default value. During Data forwarding, routers inspect the downstream iBFs. If the $fill\_factor$ of an iBF is above a predefined threshold, e.g., $fill\_factor \geq 0.4$, the router assumes that this iBF is *congested* and inserts a *Bloom filter Congestion Notification* (BCN) in the Data packet. The BCN is carried downstream to all receivers and instructs them to lower their d. This leads to smaller iBFs, which eventually stops the BCNs. If a host does not receive any BCNs for a number of consecutive Data packets, it increases its d in order to reduce PIT size, based on an exponential back-off scheme. We expand on this basic idea below.

```
procedure SEMI_STATE_FWD_AD(data, incoming_port)
    pit_entry := PIT_lookup(data.name)
    if pit_entry not null then
        data.iBF := pit_entry.iBF
        if fillfactor(data.iBF) >= fillfactor_max then
            d_BCN := max(1, pit_entry.d_min − 1)
            if BCN_is_set(data) then
                d_BCN = min(d_BCN, data.d_BCN)
            end if
            data.d_BCN = d_BCN
        end if
    end if
    iBF_Forward(data, incoming_port)
end procedure
```

Fig. 16. Semi-Stateless Data Forwarding with adaptive d.

B. Interest tracking

Each time a host issues an Interest, it selects an appropriate d and places it in the Interest header, *in addition to* the HC. During Interest forwarding, routers perform the same operations as before, except that d is obtained from the Interest's header, rather than being a system-wide constant. For each Interest stored in the PIT, routers store the *minimum* d among those received in the merged Interests for the same Data packet. For example, a user may request `/a/b/c.mp4` with $d = 3$ while another may request it with $d = 4$. If the two Interests are merged on the same router, the router will store in the PIT $d_{min} = 3$. Algorithm 15 shows the adaptive HC-based Interest tracking policy.

C. Data forwarding

Routers forward Data packets as before, adding a $fill\_factor$ check whenever a new iBF is inserted, which takes place only at routers where the corresponding Interest is tracked. Upon the reception of a Data packet, if a PIT entry for it exists, the router checks the new iBF's $fill\_factor$ and if it exceeds the threshold, the BCN feedback field is set in the Data packet header. The BCN feedback is an integer number set to $d_{BCN} = max(1, d_{min} - 1)$, where d_{min} is the value of d stored in the PIT entry. If the Data packet already contains BCN feedback, indicated by a non-zero value in the BCN field, the field is set to the minimum between the value in the header and the value calculated from the PIT. The BCN feedback propagates all the way to the receivers, instructing them to set their d to d_{BCN}. For example, if a Data packet has a BCN field set to 3 and along the path another router must set BCN to 2, then the field will be set to $d_{BCN} = min(2, 3) = 2$. The details for adaptive semi-stateless forwarding are shown in Algorithm 16 (IBF_FORWARD is the same as in Algorithm 6).

D. Receiver-side adaptation

In addition to decreasing d based on BCN feedback, hosts also try to increase d during an application session, for two reasons. First, with no prior knowledge of network conditions and/or application type (unicast or multicast), hosts must start

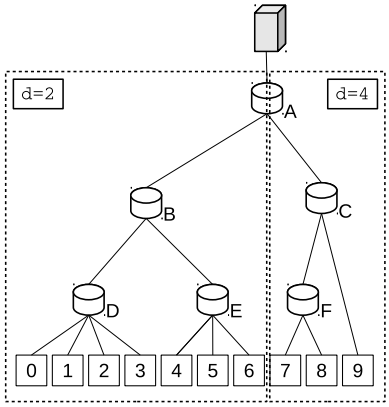


Fig. 17. Adaptation of d on a sub-tree granularity.

with a conservative value for d, $d_{default} = 2$, to avoid excessive bandwidth overheads. However, a low value for d is less effective for unicast and sparse multicast trees, where d can be increased without significant bandwidth penalties. Second, when a router sends BCN feedback, it affects all downstream receivers, even though the decision is made *based only on the locally stored iBF*. Consider, for example, the multicast tree of Figure 17, which is quite imbalanced; the subtree formed by receivers 0–6 is denser than the subtree formed by receivers 7–9. Assume that router A sets the BCN feedback to $d_{BCN} = 2$, causing all downstream receivers to set their d to 2. This will reduce the opportunities for PIT state reduction in routers C and F. In this region of the tree, $d = 4$ would produce smaller PITs with little overhead. Therefore, after A sends its BCN feedback, receivers 7, 8 and 9 have an incentive to increase d.

Hosts attempt to increase their d based on an exponential back-off scheme. At a high-level, the goal is to avoid a *collision* when all hosts increase their d together, e.g., every 10 Interests, as this might lead to very congested Bloom filters. We thus use randomization to allow the system to gradually approach a steady state. Specifically, when hosts receive Data packets with BCN feedback, they update their d. Hosts monitor when the last BCN-enabled Data packet arrived and after a number of consecutive non-BCN Data packets are received, they assume that the network is in a steady state. Hosts then increase their own d by 1, trying to achieve higher PIT state reductions. The number of consecutive non-BCN Data packets that will trigger an increase in d is calculated as

$$exp\_backoff = random(1,\ slot\_size * 2^{1+d_{BCN}}) \quad (1)$$

where the $slot\_size$ is selected by the application and may depend on the application packet transmission rate. Once a host receives a Data packet with BCN set, after setting its d, it selects a random back-off interval, which dictates the number of consecutive non-BCN Data packets that should arrive before an attempt to increase its d. If, in the meantime, another BCN-marked Data packet arrives, the whole process is restarted: d is updated and the back-off is reset using the new d. Similarly, if d is increased, the process restarts. The host will next attempt to increase d after waiting for *statistically* more time, unless BCN feedback arrives again. As we show

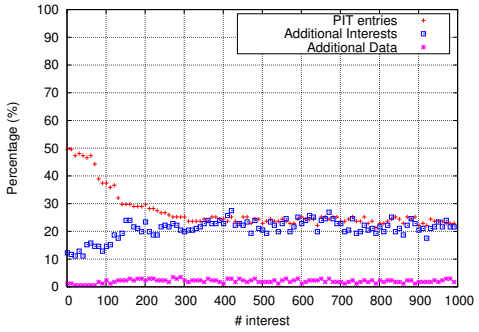


Fig. 18. Performance over time with 5 receivers.

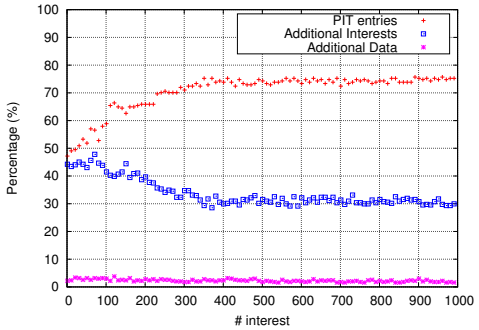


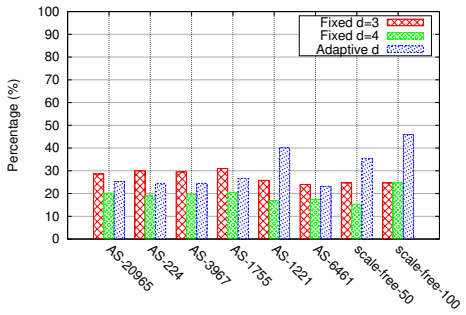
Fig. 19. Performance over time with 30 receivers.

in Section VI, this adaptive scheme keeps additional Data packets low. However, the number of additional Interests is hard to detect in a distributed manner. Based on the results of Section IV, we decided to limit d to $d_{max} = 4$, to bound the number of additional Interests.

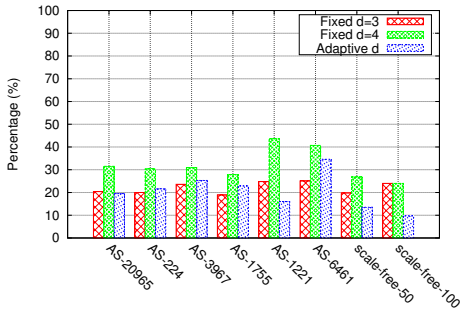
VI. Evaluation of the Dynamic Scheme

We repeated the experiments of Section IV using our adaptive semi-stateless scheme. Hosts start with $d_{default} = 2$ and an upper bound of $d_{max} = 4$. For the HTTP-like streaming application, we set the $slot\_size$ to 20 Interests; note that other applications may choose a different $slot\_size$. Again, all results shown are normalized against basic CCN and the default topology is AS-20965.

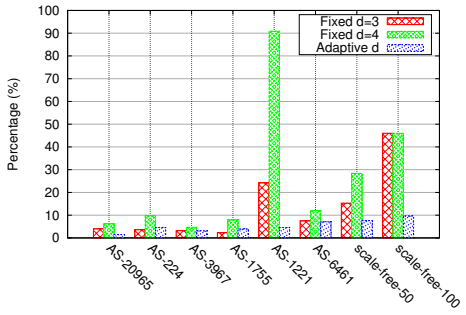
Figures 18 and 19 show the behavior of the adaptive scheme over time for multicast groups with 5 and 30 receivers, respectively, in terms of PIT entries, additional Interests and Data packets. In both cases, hosts start with $d_{default}$ and adjust d according to the BCNs received, eventually converging to a steady state. Data overhead is kept low in both cases, never exceeding 5%, at the cost of varying gains in PIT state reduction. Analysis of the data reveals that for the smaller group (Figure 18), the PIT is reduced to less than 30% of basic CCN, since all hosts are able to increase their d to 4, with no BCN feedback. On the other hand, for the larger multicast group (Figure 19), the PIT was reduced to 75% of basic CCN due to the density of the multicast tree: almost half of the receivers set $d = 1$, resorting to stateful forwarding, while other receivers used larger values. The Interests overhead also varies: $\approx$ 20% for the small multicast group and $\approx$ 30% in



(a) PIT entries



(b) Interests overhead



(c) Data overhead

Fig. 20. Comparison of fixed-d with adaptive d for all topologies (Zipf group sizes).

the large multicast group. Recall, however, that Interests, due to their small size, affect bandwidth overhead much less than Data.

Figure 20 compares the performance of the adaptive and fixed-d schemes across all topologies, when group sizes follow a Zipf distribution. In the topologies that performed well with fixed d, the PIT reduction has a similar behavior: the adaptive scheme reduces the PIT similarly to the fixed scheme with $d = 3$ or $d = 4$. The difference is in the three under-performing topologies, where PIT size is reduced less with adaptive d: 40% for *AS-1221*, 35% for *scale-free-50* and 45% for *scale-free-100*. However, we have a tremendous improvement on Data overhead (Fig. 20(c)) in these topologies, which drops below 10% in all cases; the less important Interests overhead also remains below 35% (Fig. 20(b)).

Due to lack of space, we do not show the detailed results for PIT memory size reduction and bandwidth overhead, across all topologies; results for all topologies are available in [27]. The reduction in bytes to the PIT size with the adaptive scheme is $27.3\%-63.2\%$ of basic CCN with both small and large content names, while the combined Interests and Data overhead with the adaptive scheme is $1.6\% - 12.1\%$ over basic CCN with small and large packets, combined with small and large content names. Since unicast traffic is not affected by the adaptation of d, as false positives are very rare there, for the expected fraction of multicast traffic (up to $5-10\%$) and even with worst case assumptions on PIT reduction and bandwidth overhead for multicast, our adaptive semi-stateless scheme manages to reduce the PIT size to no more than 30% of baseline CCN with all topologies, thus managing to fit it into a router's fast memory.

VII. Conclusion

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. A simulation-based evaluation over realistic ISP topologies showed that forwarding state can be reduced to $18.9\% - 27.4\%$ of basic CCN in unicast applications, with negligible bandwidth penalties. However, in multicast applications, while in most topologies the forwarding state was reduced to $15.1\% - 42.6\%$ of basic CCN at a cost of $1.1\% - 16.2\%$ of bandwidth overhead, in some topologies the overheads reached very high values, making the scheme impractical. For this reason, we extended our scheme to dynamically adapt to local tree density, so as to keep overheads low. Our semi-stateless scheme reduced forwarding state to $27.3\% - 63.2\%$ of basic CCN at the expense of only $1.6\% - 12.1\%$ of overhead, across a wide range of topologies. This allows fitting the PIT in a router's fast memory, even with a non-negligible fraction of multicast traffic, while maintaining the advantages of CCN's stateful forwarding and without requiring any changes to the CCN architecture, apart from modifications in Interest and Data forwarding.

Acknowledgments

This research was co-financed by the European Union (ESF) and Greek national funds through the Research Program THALIS - MUSINET, the EU-funded H2020 ICT project POINT under contract 643990, and by the RC-AUEB funded "Original Scientific Publications" project under contract ER-3013-01.

References

[1] 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 International Conference on Emerging networking experiments and technologies (CoNext)*. New York, NY, USA: ACM, 2009, pp. 1–12.

[2] 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.

[3] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proc. of the SIGCOMM Information Centric Networking (ICN) Workshop*. New York, NY, USA: ACM, 2011, pp. 44–49.

[4] A. Detti, N. Blefari Melazzi, S. Salsano, and M. Pomposini, "CONET: a content centric inter-networking architecture," in *Proc. of the SIGCOMM Information Centric Networking (ICN) Workshop*. New York, NY, USA: ACM, 2011, pp. 50–55.

[5] A. Azgin, R. Ravindran, and G. Wang, "pit/LESS: Stateless forwarding in content centric networks," in *Proc. of the Global Communications Conference (GLOBECOM)*. Piscataway, NJ, USA: IEEE, Dec 2016, pp. 1–7.

[6] C. Ghali, G. Tsudik, E. Uzun, and C. A. Wood, "Living in a pit-less world: A case against stateful forwarding in content-centric networking," *CoRR*, vol. abs/1512.07755, 2015. [Online]. Available: http://arxiv.org/abs/1512.07755

[7] H. Yuan and P. Crowley, "Scalable pending interest table design: From principles to practice," in *Proc. of the Conference on Computer Communications (INFOCOM)*. Piscataway, NJ, USA: IEEE, 2014, pp. 2049–2057.

[8] W. You, B. Mathieu, P. Truong, J.-F. Peltier, and G. Simon, "DiPIT: A distributed Bloom-filter based PIT table for CCN nodes," in *Proc. of the International Conference on Computer Communications and Networks (ICCCN)*. Piscataway, NJ, USA: IEEE, 2012, pp. 1–7.

[9] X. Wang, W. Wang, C. Zeng, R. Dai, S. Wang, and S. Xu, "Reducing the size of pending interest table for content-centric networks with hybrid forwarding," in *Proc. of the International Conference on Communications (ICC)*. Piscataway, NJ, USA: IEEE, May 2016, pp. 1–6.

[10] J. Garcia-Luna-Aceves and M. M. Barijough, "Content-centric networking using anonymous datagrams," in *Proc. of the IFIP Networking Conference*. Piscataway, NJ, USA: IEEE, May 2016, pp. 171–179.

[11] J. Garcia-Luna-Aceves and M. Barijough, "Efficient multicasting in content-centric networks using datagrams," in *Proc. of the Global Communications Conference (GLOBECOM)*. Piscataway, NJ, USA: IEEE, Dec 2016, pp. 1–6.

[12] A. Carzaniga, M. Papalini, and A. L. Wolf, "Content-based publish/subscribe networking and information-centric networking," in *Proc. of the SIGCOMM Information Centric Networking (ICN) Workshop*. New York, NY, USA: ACM, 2011, pp. 56–61.

[13] C. Tsilopoulos, G. Xylomenos, and Y. Thomas, "Reducing forwarding state in content-centric networks with semi-stateless forwarding," in *Proc. of the Conference on Computer Communications (INFOCOM)*. Piscataway, NJ, USA: IEEE, 2014, pp. 2067–2075.

[14] A. Dabirmoghaddam, M. Dehghan, and J. J. Garcia-Luna-Aceves, "Characterizing interest aggregation in content-centric networks," *CoRR*, vol. abs/1603.07995, 2016. [Online]. Available: http://arxiv.org/abs/1603.07995

[15] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in named data networking," in *Proc. of the Symposium on Architectures for Networking and Communications Systems (ANCS)*. New York, NY, USA: ACM, 2012, pp. 211–222.

[16] M. Varvello, D. Perino, and L. Linguaglossa, "On the design and implementation of a wire-speed pending interest table," in *Proc. of the Conference on Computer Communications (INFOCOM) Workshops*. Piscataway, NJ, USA: IEEE, 2013, pp. 369–374.

[17] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Pending interest table sizing in named data networking," in *Proc. of the Information-Centric Networking Conference (ICN)*. New York, NY, USA: ACM, 2015, pp. 49–58.

[18] C. Stais, Y. Thomas, G. Xylomenos, and C. Tsilopoulos, "Networked music performance over information-centric networks," in *Proc. of the International Conference on Communications (ICC) Workshops*. Piscataway, NJ, USA: IEEE, 2013, pp. 647–651.

[19] 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 Workshop on Re-architecting the Internet (ReArch)*. New York, NY, USA: ACM, 2009, pp. 1–6.

[20] C. Tsilopoulos and G. Xylomenos, "Supporting diverse traffic types in information centric networks," in *Proc. of the SIGCOMM Information Centric Networking (ICN) Workshop*. New York, NY, USA: ACM, 2011, pp. 13–18.

[21] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," in *Proc. of the SIGCOMM Conference on Data Communications*. New York, NY, USA: ACM, 2009, pp. 195–206.

[22] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better Bloom filter," in *Proc. of the European Symposium on Algorithms*. Berlin, Heidelberg: Springer, 2006, pp. 456–467.

[23] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[24] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proc. of the SIGCOMM Conference on Data Communications*. New York, NY, USA: ACM, 2002, pp. 133–145.

[25] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[26] M. Särelä, C. Esteve Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott, "Forwarding anomalies in Bloom filter-based multicast," in *Proc. of the Conference on Computer Communications (INFOCOM)*. Piscataway, NJ, USA: IEEE, 2011, pp. 2399–2407.

[27] C. Tsilopoulos, "Multicast forwarding in future information-centric network architectures," Ph.D. dissertation, Athens University of Economics and Business, Department of Informatics, 2016.

[28] T. W. Cho, M. Rabinovich, K. Ramakrishnan, D. Srivastava, and Y. Zhang, "Enabling content dissemination using efficient and scalable multicast," in *Proc. of the Conference on Computer Communications (INFOCOM)*. Piscataway, NJ, USA: IEEE, 2009, pp. 1980–1988.

[29] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, 2002.

[30] S. Tarkoma, C. Esteve Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *Communications Surveys and Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.