IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 6, 2021, accepted March 10, 2021, date of publication March 17, 2021, date of current version March 23, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3066133*

# Efficient Pooling and Collaborative Cache Management for NDN/IoT Networks

**BASHAER ALAHMRI, SAAD AL-AHMADI, (Member, IEEE), AND ABDELFETTAH BELGHITH, (Senior Member, IEEE)**
Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
Corresponding author: Saad Al-Ahmadi (salahmadi@ksu.edu.sa)

This work was supported by the Deanship of Scientific Research, King Saud University, through the Research Group, under Grant RG-1439-023.

**ABSTRACT** Named Data Networking (NDN) has been recognized as a lever to the Internet of Things (IoT). One of the most founding features of NDN is in-network caching to improve data availability and reduce retrieval delays and network load. Despite the existence of several caching decision algorithms, the fetching and distribution of contents with minimum resource utilization remains a great challenge. In this article, we propose an efficient caching technique named PoolCache that augments the effective caching capacity of some defined conglomerates of nodes. This is accomplished by pooling the various caches and manage them in a way to insure zero content redundancy within any defined node conglomerate. The resulting high diversity of cached contents throughout the network tacitly amounts to much better overall performances. We conducted extensive simulations using the CCNsim simulator to evaluate the performance of PoolCache and compare it to that of some well known caching strategies. Simulations using a large Transit Stub topology show that PoolCache clearly outperforms the other caching strategies in terms of a much greater content diversity and consequently a limited number of content evictions, a much better cache hit ratio, and a much lower content retrieval delay. Simulations also showed that PoolCache benefits from any eventual content popularity.

**INDEX TERMS** Information-centric network (ICN), named data network (NDN), Internet of Things (IoT), in-network caching, transit-stub topology, clustering, hashing.

## I. INTRODUCTION

The Internet of Things (IoT) is a revolutionizing and challenging ecosystem that is drastically changing our work styles, our ways to interact, behave, and reason about our surrounding. IoT is becoming a strong driver in our interconnected world through ubiquitous advanced connectivity of smart devices, services and even data contents all of them are simply called objects or things. IoT is providing an advanced platform using a wide variety of networking protocols, where smart objects communicate, learn, anticipate and perceive in a much more effective way to respond to issues, and challenges. Several IoT-enabled applications already exist, including smart cities, e-healthcare, and e-education [1]. Billions of smart objects are already connected; Cisco forecasts a global mobile data traffic growing at a rate of 46% per year and will total nearly 77.5 exabytes of traffic per month in 2022 [2].

Embedding a regular TCP/IP protocol stack on constrained IoT devices and smart objects is quite problematic and raises several issues. The seminal Internet host-centric model is no more appropriate as IoT traffic, and also that of the current Internet, consists in its majority in content dissemination and retrieval. To this end, Van Jacobson proposed a new vision of the Internet centered on the information or the content called Information Centric Networking (ICN) [3]. Several ICN architectures have been proposed, including DONA, Content Centric Networking (CCN), Named Data Networking (NDN), PURSUIT, NetInf, Convergence, CONET, MobilityFirst, and Green ICN [4]. Many studies have already pointed out that Named Data Networking (NDN) has the potential to become the key technology for data dissemination in IoT [5]–[7]. NDN promotes IoT networking and data dissemination, and reduces

The associate editor coordinating the review of this manuscript and approving it for publication was Antonino Orsino.

the protocol stack complexity by eliminating the need of the CoAP/RPL/6LowPan/802.15.4 stack. NDN proposes a pull-based, receiver-driven, robust connection-less communication model which fulfills the basic requirements of IoT traffic and systems in terms of easy and scalable data access, security, energy efficiency, and mobility support [8], [9].

In-network caching is a fundamental feature of the NDN architecture to reduce traffic load and increase data availability by satisfying client requests from close cache nodes rather than from a remote producer. A plethora of caching schemes have been proposed [10], [11]. A caching strategy determines where to cache and which content to replace when the cache is full. As the content travels through the Interest reverse path in NDN, the caching strategy has then to decide at each traversed node whether to cache the arriving content or not [6], [10].

Caching in the NDN/IoT framework should take into consideration the nature of IoT devices and objects which are heavily constrained in terms of memory, battery and computation power. As such, the caching strategy should require limited resources, yet delivers and maintains good system performances. Caching allows to temporarily cache contents at nodes on the reverse path from content providers (or producers) to consumers. Systematically caching at every node on this path amounts to a huge redundancy which degrades the performance. In addition, the caching capacity of routers is usually relatively small compared to the catalogue of contents. Controlling the redundancy level of caching with minimum collaboration and communication efforts is then a fundamental and challenging issue. In this article, we aim to efficiently use and share the different caching resources available at nodes within a given router conglomerate (we here interchangeably use the words conglomerate or neighborhood) by caching just one copy and therefore limiting the redundancy at best within the given network node conglomerate. We propose to efficiently pool, manage, assign and control the available caching storage at each router conglomerate as a distributed pooled storage resource. To this end, we may perform a clustering of the network nodes, or designate the nodes within the different conglomerate/clusters or simply rely on the underlying network topology. In any case, no cache redundancy happens within the cluster and a higher number of different contents could be cached within the network, hence achieving a higher caching diversity. The question naturally arises as to how to manage and control the distributed caching at each router conglomerate. We shall propose a simple hashing technique that maps the requested content name to a node within the router conglomerate. As a result, both content retrieval and bandwidth consumption are reduced. The Forwarding Information Base (FIB) entries should be updated accordingly as in the basic NDN architecture. We shall propose a simple way to perform this updating of the various FIBs. In addition, a certain path stretch would result depending on the degree of connectivity of the nodes within the conglomerate. Note that the proposed caching strategy employs the caching only at the edge of the network and completely avoids caching contents in routers of the core of the network which is then left as a named data packet switching fabric. In addition, the proposed caching strategy implicitly takes into account and freely takes advantage of any content popularity without any effort.

Few solutions have been proposed in the literature that focused on managing and controlling the caches of local nodes in a distributed manner to reduce caching redundancy and increase content diversity within the entire network. However, almost all of them rely on the potentially large sharing of information between neighboring nodes, for an orchestrated caching decision. This work advances the state of the art by offering a distributed simple and efficient caching policy that simultaneously accounts for the content popularity, higher content diversity, caching at the edge of the network, greater efficiency and fully compliant with NDN pillars.

The main contributions of this article can be summarized as follows:

1) the design of PoolCache a novel and fully NDN compliant caching policy for IoT contents. The proposed PoolCahe solution treats local caches at the network edge as a pooled, fully shareable and distributed storage resource. Caching decisions are taken autonomously by nodes within the same network edge neighborhood according to a simple hashing function.
2) The updating of the FIB entries in a very simple manner to enforce the NDN name based routing within the edge neighborhood as well as toward producers.
3) The use of a large Transit Stub topology and a large number of consumers and producers to properly account for different IoT workload setups.
4) the evaluation of the performance of the proposed policy and comparison against other representative caching schemes using the CCN simulator [12]. In addition to the common valuable metrics (i.e., cache hit ratio, content retrieval latency, hop distance, eviction ratio), we investigated the ability of HachCache to reduce content caching redundancy, to alleviate the core network from any caching burden, and to implicitly profit from any potential content popularity.

The remainder of the paper is organized as follows: section 2 introduces the necessary background and reviews the related work. Section 3 describes the proposed Pooling and Cache management technique termed PoolCache, and provides illustrative examples of its operation. Section 4 presents the performance evaluation of PoolCache. It details the various caching evaluation metrics, the used topology and the simulation scenarios. A detailed comparison with some well known caching strategies highlights the efficiency of the proposed PoolCache. Section 5 presents the gain in the efficiency of PoolCache when increasing the popularity index. Finally, Section 6 concludes the paper.

## II. BACKGROUND AND RELATED WORK

### A. NDN STRUCTURE AND OPERATION

NDN is a receiver-driven communication architecture that includes just two types of packets: Interest and Data packets as shown on Figure 1. Each packet has a unique content name. Routers use the content name to forward Interest packets toward the producers. Data packets are transmitted along the reverse path towards the requesting consumers. NDN routers use three fundamental data structures; namely the Pending Interest Tables (PIT), the forwarding information base (FIB), and the Content Store (CS). Users (i.e., consumers) issue Interest packets to request contents. An Interest is routed by name from router to the next according to the FIB until either reaching a router having a cached copy of the requested content in its CS or ultimately reaching the provider (i.e., producer) of the requested content. Upon the arrival of an Interest, a router first performs a lookup of its CS. In case the requested content is not within the cache (i.e., a cache miss), the Interest packet is forwarded to the next hop router according to the FIB, and the interface of this incoming Interest is appended into the PIT. In case of a cache hit, the targeted content is sent back to each of the recorded interfaces stored in the PIT that have requested the content. Then the corresponding entries in the PIT are deleted.

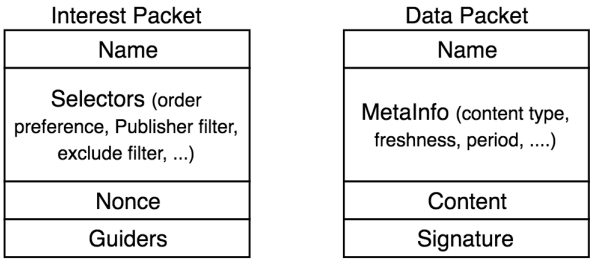



**FIGURE 1.** NDN packet types.

### B. SUITABILITY OF NDN FOR IoT

Current IoT systems and platforms on the Internet rely essentially on the cumbersome CoAP/RPL/6LowPan/802.15 protocol stack. Information Centric Networking (ICN) replaces this stack in a very easy way. ICN in general and NDN in particular provides by design the necessary ingredients and capabilities to be a very suitable and efficient networking platform for IoT traffic and applications. These capabilities concern essentially: location-independent naming, in-network caching, name-based routing, multicast, anycast, consumer mobility and self-secured contents. The authors in [6] conducted a qualitative comparative study of most of the proposed ICN architectures, and showed that Named Data Networking stands out as the most suitable ICN proposal for IoT. Several other work already proposed NDN for IoT. In [13], the authors proposed the design of a high-level NDN architecture that meets IoT challenges. They discussed the benefits of NDN and the way NDN architecture may easily support IoT main requirements. The authors in [14] suggested NDN architecture to be a very effective architecture for IoT scenarios. They also showed that NDN satisfies IoT requirements. Other studies focused rather on IoT requirements that ICN architectures can inherently support without discussing which one is the more suitable for IoT scenarios. The authors in [15] discussed the major requirements for IoT and challenges of ICN architecture to realize a unified framework. The authors in [5] presented IoT requirements and discussed several possible motivations for the introduction of ICN in the context of IoT.

### C. RELATED WORK AND IN-NETWORK CACHING POLICIES

In Named Data Networking, each router has a cache (a.k.a. the CS Store) to store and provide frequently requested contents. An efficient cache management scheme is essential for efficient content distribution, high cache utilization, and permitting a high diversity level. Many caching policies have been proposed for ICN [4], [16]. Leave Copy Everywhere (LCE) [17] is an approach that leaves a copy of the requested data content in each router along the path towards the consumer. This tacitly amounts to a huge redundancy which reduces the caching hit rate and increases the eviction rate to accommodate newly arriving contents. On the contrary, our proposed PoolCache technique caches the new content at just one router, and consequently will necessarily amount to a much better content diversity and availability, much lower redundancy and eviction rate, and a superior cache hit rate and efficiency.

Authors in [18] proposed the Leave Copy Down (LCD) strategy which keeps storing a copy of the requested data content in a node placed one level down in the reverse path. For popular contents, LCD tends to behave as LCE; namely all nodes on the reverse path towards the consumer would eventually cache identical copies. LCD may then have the same shortcomings of LCE especially when requested contents are very popular.

Authors in [19] introduced Probabilistic cache (ProCache) which privileges caching close to consumers. The caching process is executed with a varying probability inversely proportional to the distance between the consumer and the producer. ProCache presents an unequal resource allocation among nodes, a high computational overhead, and requires fine tuning of its parameters.

The Betweenness-Centrality (Btw) strategy was proposed in [20], where data contents are cached just once on the reverse path at the node having the highest betweeness-centrality. The Betweenness Centrality (Btw) strategy is based on measuring the number of times a node belongs to a path between all pairs of nodes in a network topology. As such, identifying the value of a router's betweenness centrality is a challenging task not viable for resource constrained nodes.

The authors in [21] introduced edge caching, a graph-based caching strategy. Edge-caching strategy is proposed for tree topology and caches contents at the topology leaves. In this

respect, it may result in a high degree of duplication among neighboring leaves. However, like our proposed PoolCache technique it alleviates the core network from any caching burden and overhead.

Some or all of these aforementioned schemes are the standard commonly used benchmark for virtually all Information Centric Caching proposals. They played an important role to ascertain the efficiency and judge the relevance of any proposed more advanced caching scheme.

In [11], [22], the authors proposed the Consumer caching strategy which caches data contents on routers directly connected to the consumers. Consumer cache behaves then just like the edge caching technique for networks having a tree topology. However, it behaves like LCE in the other extreme case where there are consumers connected to each router on the reverse path. The Consumer caching strategy performances depend on the network topology, the distribution of the consumers within the network, and the popularity of the different contents.

In [23], [24], the authors proposed a collaborative in-network caching scheme with Content-space Partitioning and Hash-Routing named CPHR. They partitioned the content space and then assigned partitions to caches. They formulated the problem of assigning partitions to caches as an optimization problem that maximizes the overall hit ratio and proposed a heuristic to solve it. They also formulated the partitioning as a min-max linear optimization problem that balances the cache loads. Using the general purpose simulator ns3 and adopting the Least Recently Used LRU technique for content replacement, they showed that CPHR can double the overall hit ratio but at the same time increases the average propagation latency. It is worth to note that the paper does not contain any comparison with other known caching schemes, and that their collective caching requires extra content tables to maintain caching states and costly mechanisms to maintain table consistency. The complexity of partitioning the content space and the periodic maintenance of the various data structures put a large burden to be viable for IoT traffic. Furthermore, the implicit assumption of a fixed content space and stationary network is hardly viable for IoT scenarios and setups.

In [25], the authors also proposed a cluster-based in-networking caching mechanism to reduce caching redundancy for Content Centric Networking. They partitioned the complete network into a certain number of clusters using an improved K-medoids cluster algorithm. They used a Virtual Distributed Hash Table (VDHT) to efficiently control and manage the resources stored in each cluster. In addition, they proposed different policies for intra and inter cluster routing to forward the requests. They showed by simulation using the specialized ccnsim simulator that their proposed caching technique outperforms both LCE and ProCache in terms of cahe hit ratio and link load as a function of the content popularity index (skewness). It is worth noting that in addition to the overhead of the k-Medoid clustering algorithm, the caching overhead significantly increases as clusters grow in size. Their technique does not differentiate between edge nodes and core routers, all behaves and performs the same tasks. The performance comparison is restricted to only LCE and ProCache using just two performance metrics.

In [26], the authors proposed an efficient hash-based cache distribution and search schemes in CCN. The updating of the FIBs is done by flooding which induces a huge overhead. The performance evaluation is performed by simulation using ns3, and they used a very simple grid network topology. Besides no comparison is conducted with any other caching scheme.

The authors in [27] proposed the hierarchical cluster-based caching (HCC) strategy; a two-layer hierarchical cluster based caching solution. The clustering of the whole network is done using the known Weighted-based Clustering Algorithm (WCA). A centralized cluster head selection is used for each cluster to make caching decisions based on a centralized probability matrix. The proposed solution generates a high communication overhead, as it periodically exchanges information related to content caching. The unavailability of a clusterhead breaks down the operation of the complete cluster. HCC still restricts content request forwarding and reduces cache utilization. The implicit assumption of a stationary network may not be viable for IoT scenarios.

In [28], the authors proposed the sharing of cache summaries among neighboring routers to increase the diversity of cached contents in NDN. They proposed a scheme to define a summary packet using a bloom filter and a method to share the summary. At the arrival of a Data packet, a router decides whether or not to save it depending on the cache summaries of neighboring routers. Upon the arrival of an Interest, a router can forward the Interest to a neighboring router that has the requested content. Unlike our proposed PoolCahe, their proposed technique necessitates an involved cooperation and requires persistent querying and the maintenance of summaries.

In [29], the authors proposed DANTE a diversity-improved caching scheme for vehicular Named Data Networking. They leveraged the broadcast wireless support for nearby nodes to cache different content and thus improve caching diversity. The objective is to give a higher caching probability to more popular contents with a longer lifetime, which are not already cached by a nearby node within the wireless neighborhood. the intrinsic broadcast nature of the wireless medium allows each vehicle to infer what contents are available in its neighborhood. Diversifying the caching of different contents among nearby vehicles limits the caching redundancy in the wireless neighborhood, and translates into better network performance. DANTE necessitates a wireless network and is devoted to vehicular NDN. PoolCache, on the opposite, may be applied to any NDN regardless the nature of the underlying medium communication technology.

In all preceding work, both the clustering of the network or the partitioning of the content space induce a high complexity and add a non negligible computing and communication overhead. Besides, the tacit assumption of stationary producers and consumers is far from being the practical case

of IoT environments. Most of these research work performs the clustering of the whole network and therefore require a non-negligible computing and communication overhead, yet they tacitly do not alleviate the core routers from the caching burden. In addition, they keep track of heavy data structures that may require periodic heavy maintenance. Most of these work used simplistic network topology, and provided a rather limited performance comparison with other known caching techniques.

### D. TRANSIT-STUB TOPOLOGY

In simulations, the used network topology plays an active role to be able to appropriately represent real-world situations and scenarios. Research in [30] showed that Internet domain structures have four to five hierarchical tiers. The tiers are logical routing domains. The top tier has massive Autonomous Systems (AS) connectivity, and ASs in the lower tiers have lower edge degrees. Additionally, the bottom-tier ASs depend on the top tier to route destination packets. Stub domains comprise the bottom tier, and transit domains the top tier. As mentioned in [31], every domain in the real Internet is either considered a transit or a stub domain. Transit domains provide transit connectivity to different domains and send packets to external sources. Stub domains send packets that are within a domain. Most traffic comes from stub domains, which typically represent Internet endpoints. Previous work show no particular topology that can evaluate all NDN aspects. However, we take the direction followed by most researchers that the TS topology reflects the current Internet hierarchical structure and represents the wide area IoT. As such, We shall consider, for the performance evaluation of PoolCache, a large three level transit-stub topology.

## III. PoolCache: POOLING AND COLLABORATIVE CACHE MANAGEMENT TECHNIQUE FOR NDN/IoT NETWORKS

PoolCache is a collaborative caching technique that considers the various storage resources for caching within a given neighborhood (a given subset of nodes) to be managed as one global caching storage. Namely, PoolCache controls one global Content Store which is partitioned among the different nodes of the given neighborhood or cluster. Essentially, PoolCache acts at the edge of the network and never considers caching contents in core routers. Within a cluster of nodes, PoolCache caches a content in just one node and consequently a zero redundancy is achieved within any cluster. The caching node is designated according to a hashing function on the content name. The hash function should evenly distribute/cache contents among the different nodes of the cluster. Our hash function maps content names to nodeIDs. Neighborhoods or node clusters could be defined in different ways. We may repose on the underlying topology of the network to compose the different clusters. We may also resort to a clustering or partitioning algorithm to decide the clusters or the partitions [26], [27], [32]. Though doing so is not only time consuming but also do not preclude network core routers from serving as caching nodes. ISPs can also designate the different neighborhoods in an ad hoc manner according to their needs and the requirements of different applications and working environments (e.g.; Enterprise, video streaming, schools, ...). The designation of nodes within the same cluster may not depend on the topology being flat or hierarchical as long as these nodes are interconnected and within the same geographical neighborhood. It is worth noting that no need for a complete clustering or partitioning of the whole network. In fact, we may just designate some clusters and many nodes may remain unclustered. For the simulation purpose in this article, we shall rely on the underlying topology of the network which is considered to be a hierarchical Transit Stub (TS) topology. Consequently, the stubs may tacitly define the different clusters. Such a TS topology represents adequately the current Internet topology [33]–[35].

In NDN, the size of the capacity of the Content Store at each node is very limited relatively to the huge number of contents. In [36], the authors showed that the ratio of the cache size over the catalogue size (the number of named contents) should be within the interval $\left[10^{-5}; 10^{-1}\right]$. PoolCache by treating the CSs of the nodes within the cluster as a unique storage is then a novel way to have a much large CS. This will tacitly amounts to better performances as will be investigated in the next section.

The system design of PoolCache requires the addition of just one additional name within the packet format (for both the Interest and the Data packets) of NDN. We shall hereafter refer to Packet.name1 and Packet.name2 for the two names within the Packet being either Ineterest or Data. No additional field is required in the FIB; namely the forwarding is done exactly the way it is specified by the NDN Network Forwarding Deamon (NFD). The underlying name routing is that of the NDN (a.k.a. the Named Data Link State Routing Protocol NLSR). The packet now carries two names: the content name and the caching node name. The forwarding is always done using the name prescribed first in the packet; namely Packet.name1. As illustrated on Figures 2 and 3, we consider a cluster formed by nodes $n1$, $n2$, $n3$, $n4$, $n5$ and $n6$. When a request (Interest) comes from a consumer as indicated by the red arrow (1), edge router $n1$ hashes the requested content name (/Home/Room1/Tmp) to one of the nodes in its cluster. This indicates $n3$ as the caching node, that is the unique node that could have a cached copy of the requested content within the cluster. Edge router $n1$ first takes the requested content name prescribed in the received Interest.name1 and puts it in Interest.name2. Then it places $n3$ in Interest.name1. This way and as indicated by the FIB of node $n1$ on both Figures 2 and 3, the Interest is now to be forwarded to node $n2$. This is also indicated by the red arrow (2) on both figures. In turn, $n2$ forwards this coming Interest to $n3$ as indicated by the red arrow (3) and according to its FIB as shown in both figures. The designated caching node $n3$ upon receiving the Interest packet, inspects its CS to see whether there is a cached copy of the requested content. We distinguish here the two following cases.

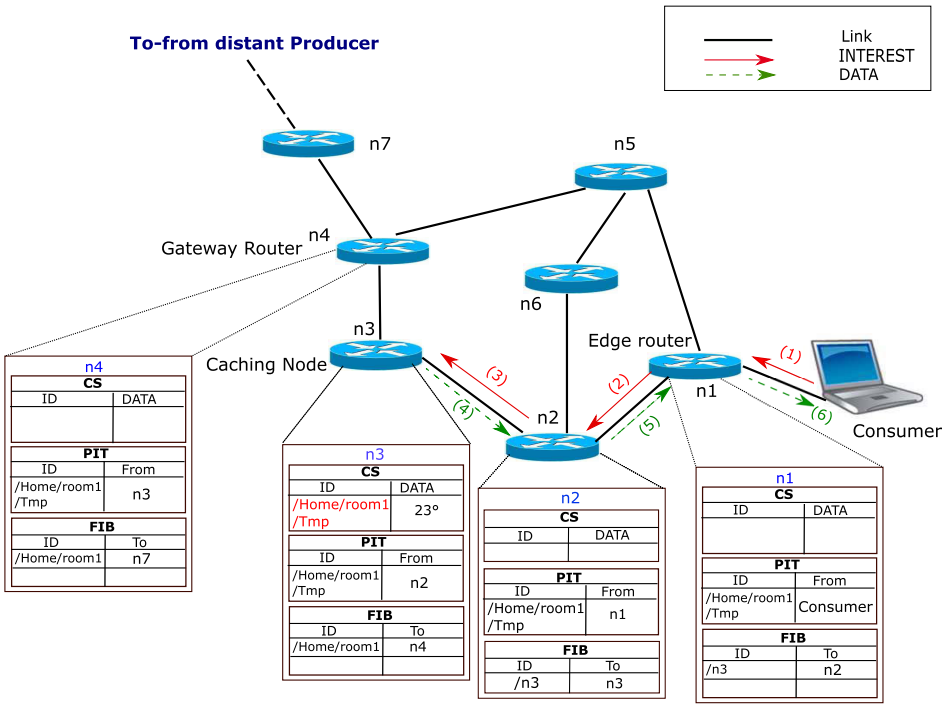



**FIGURE 2.** PoolCache: Case of existence of a cached copy within the defined neighborhood.

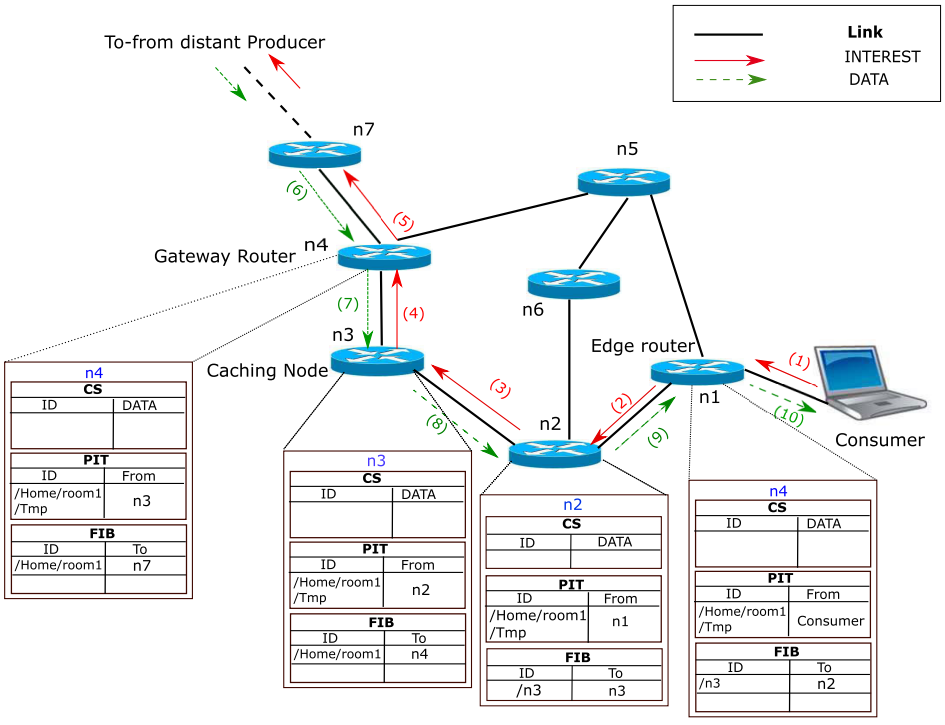



**FIGURE 3.** PoolCache: Case of non-existence of a cached copy within the defined neighborhood.

The first case is when there is indeed a cached copy of the requested content in the caching node CS. This is illustrated on Figure 2: the requested content /Home/Room1/Tmp is indicated in red in the CS of $n3$. Node $n3$ responds with its cached copy which contains the content name in Data.name1. The Data packet follows the reverse path as indicated by the PITs of $n3$, $n2$, and $n1$. This is illustrated on Figure 2 by green arrows (4), (5) and (6) successively.

The second case is when there is no cached copy of the requested content in the CS of the caching node $n3$. This is illustrated on Figure 3. $n3$ first prescribes the content name in Interest.name1 and its own name $n3$ in Interest.name2 and then forwards normally the Interest according to its FIB. The prescription of the caching node in Interest.name2 is meant to avoid the caching of the content on its way back from the producer to the consumer. The only node permitted to cache the content is the caching node $n3$. As indicated by the FIB of $n3$, the Interest is then forwarded to node $n4$ (the red arrow (4)). Node $n4$, in its turn, forwards the incoming interest to $n7$ as indicated by its FIB and illustrated by red arrow (5). The Interest will be forwarded until it reaches the producer which responds with the requested content. The producer first puts the content name in Data.name1 and the caching node name $n3$, which is received in Interest.name2, in Data.name2. The Data packet will follow the reverse path as prescribed by the PITs until it reaches the consumer. Along this route, no intermediate node but $n3$ performs the caching of the content.

Algorithm 1 illustrates the PoolCache strategy at a given node. It is the algorithm to be executed at the arrival of an Interest packet. The node receiving an Interest packet, first checks whether the Interest comes directly from an attached consumer. In the affirmative, it uses the hashing function H to map the content name (Interest.name1) to its Caching Node. Then it inserts the received content name in Interest.name2 and the Caching node in Interest.name1. Then it forwards the Interest normally according to its FIB. Since Interest.name1 contains the Caching node name, the Interest will then be forwarded toward the Caching node. Otherwise, that is if the incoming Interest is not from an attached consumer, we need to check whether the receiving node is the Caching node to execute either of the above described cases. Note that the last else in Algorithm 1 represents the case of a node that is neither attached to a consumer nor the Caching node. This type of node just forwards the Interest according to its FIB. This insures that all nodes forward Interest packets exactly according to the specification of the Network Forwarding Daemon of the NDN.

Algorithm 2 is the Data forwarding and cache decision algorithm of PoolCache that is executed by a node when it receives a Data packet. The Data packet contains the two names: Content name in Data.name1 and Caching node in Data.name2. These two names should have been inserted by the producer before sending the Data packet. The producer gets the Caching node from Interest.name2 of the Interest it had received. The Data packet travels following the reverse path until reaching the Caching node where it is going to be cached and then travels towards the consumer. It is here worthy to note that all nodes but the Caching node just forward the Data packet according to their PITs. The unique node that caches the Data packet is the Caching node that is prescribed in Data.name2 of the received Data packet. This Firstly insures a zero redundancy within the designated neighborhood as well as a high diversity within the network, and secondly caching is only performed at the edge of the network and never at nodes of the internal levels which are left for the sole purpose of named packet switching.

**Algorithm 1** Interest Forwarding Algorithm

*H*: Used hashing function
*FIB*: Forwarding Information Base
*PIT*: Pending Interest Table
*CS*: Content Store

**Input:** Interest Packet
**Output:** Interest Forward Decision

```
if Received from consumer then
    Caching node := H(Interest.name1)
    Interest.name2 := Interest.name1
    Interest.name1 := Caching node
    Forward the Interest packet according to FIB
else if Receiving Node Id = Interest.name1 then
    if Cached copy in CS then
        Return data packet to consumer according to PIT
    else
        Itemp := Interest.name1
        Interest.name1 := Interest.name2
        Interest.name2 := Itemp
        Forward the interest packet according to FIB
    end if
else
    Forward the Interest packet according to FIB
end if
```

**Algorithm 2** Data Forwarding and Caching Decision Algorithm

*PIT*: Pending Interest Table
*CS*: Content Store

**Input:** Data packet with Data.name1 = content name and Data.name2 = Caching node
**Output:** Forwarding and Caching decision

```
if Receiving Node Id = Data.name2 then
    Cache the incoming Data packet in CS
    Forward Data packet according to PIT
else
    Forward Data packet according to PIT
end if
```

## IV. PERFORMANCE EVALUATION

In this section, we detail the performance evaluation of our proposed caching technique PoolCache, and we perform an extensive comparison with known caching approaches such as Leave Copy Everywhere LCE [17], Leave Copy Down LCD [18], Probability Caching ProbCache [19], Betweenness-Centrality Btw [20], Edge-caching [21], and the Consumer-cache [22].The performance evaluation of the proposed work as well as the aforementioned schemes is conducted by simulation using the ccnSim simulator [12] which

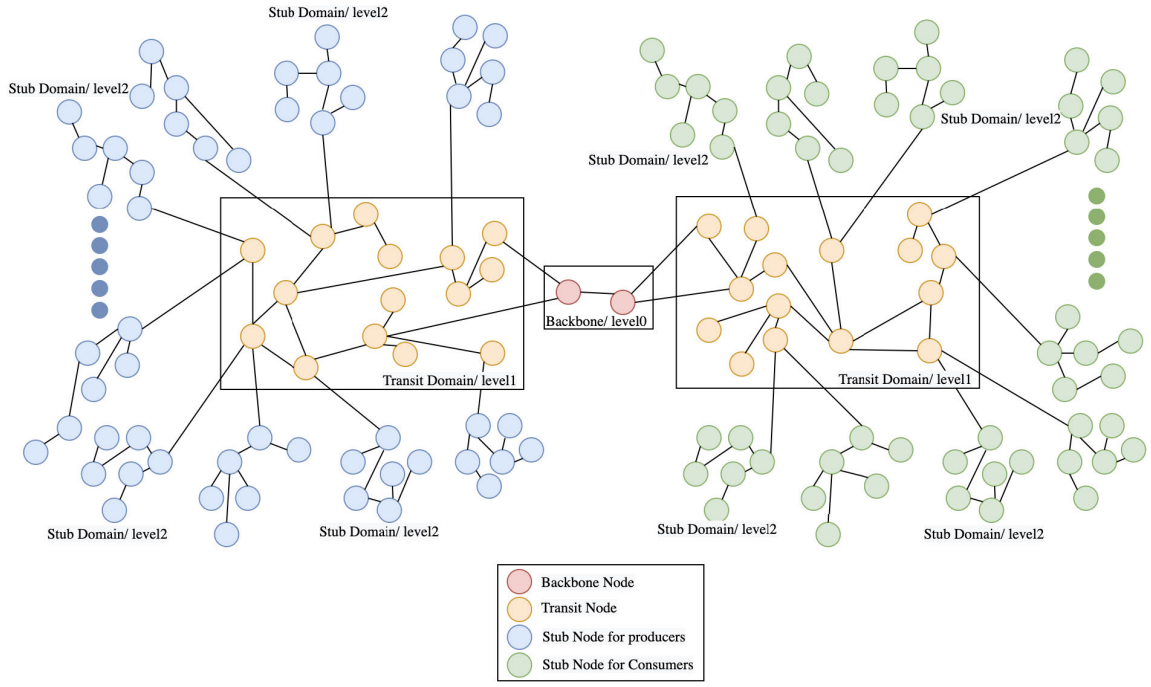



**FIGURE 4.** Transit-Stub topology used in the simulation.

is a C++ framework under the OMNeT++ to simulate NDN architectures.

### A. SIMULATION SCENARIO

For our simulation, we opted for the Transit-Stub (TS) topology whose properties closely imitate the wide area IoT topology. The TS is a 3-level hierarchical topology composed of interconnected stubs and transit domains. While stubs only carry either originating or terminating traffic, transit domains are intended to efficiently interconnect stub domains. As such, it will be judicious to preserve nodes of the transit domains from any caching burden. This is exactly the way our proposed caching technique behaves.

Figure 4 illustrates our TS topology that is generated using the GT-ITM [1] (Georgia Tech Internetwork Topology Models). GT-ITM is a collection of software tools for generating and analyzing graph models of network topologies using the NED language in OMNET++.

The TS topology used is a 3-level hierarchy composed of a core domain containing 2 level 0 backbone routers, 2 transit domains each containing 15 level 1 backbone routers, and 30 stubs per transit domain each containing on the average 6 routers. The total number of routers in our TS topology is then 392 nodes (we interchangeably use routers or nodes). Producers and consumers can only connect to stub nodes. Producers and consumers are connected to nodes in different transit domains. 59 producers are spread randomly across 30 stubs in the left transit domain of the network on Figure 4 (blue color nodes), and 48 consumers are attached to nodes in 8 out of the 30 stubs on the right side transit domain of Figure 4 (green color nodes). Transmission delays are set by GT-ITM so that transmissions in the inner level 0 are much faster than those in transit level which in turn are much faster than those in the stub level. Values are set in the range of $[2; 78]\,ms$.

We suppose that Interests are generated at each consumer following the Independent Request Model (IRM). However, the content requested in each generated Interest follows the Zipf probability distribution with parameter $\alpha = 1.2$. All nodes have the same constant cache size that takes into consideration the storage interval designated by Rossi and Rossini in [36] as described earlier. The replacement strategy used for all considered caching techniques is the Least Recently Used (LRU). We could have easily used other replacement techniques [37], but here we are solely concentrating on the design of an efficient caching technique. Table 1 summarizes the system parameters used in our simulations.

### B. EVALUATION METRICS

In order to conduct the performance evaluation of our proposed strategy and the comparison with the described caching schemes, we consider the cache hit ratio, the server hit ratio, the average retrieval latency, the hop reduction ratio, and the number of eviction metrics.

A cache hit occurs when an Interest is answered by an intermediate router along the path to the server. The cache hit ratio measures the reduction of the rate of access to the server; namely the server load reduction rate. Equation (1) gives this metric, where $N$ is the number of consumers, $serverHit_i$ is

[1] http://www.cc.gatech.edu/projects/gtitm/

**TABLE 1. The key simulation parameters.**

| Parameter | Value |
|---|---|
| Network topology | Transit-Stub (TS) Topology |
| Number of nodes | 392 *nodes* |
| Number of Contents | 10000 |
| Consumers (Clients) | 48 |
| Producers (Servers) | 59 |
| Cache size over catalog size ratio | $10^{-3}$ |
| Cache Size | 10 chunks |
| Content Size | 1 chunk |
| Replicas | 1 |
| Content request model | IRM |
| Arrival rate | $\lambda = 1$ |
| Replacement Policy | LRU |
| Content Distribution Model | Zipf ($\alpha = 1.2$) |
| Transmission Delay | [2; 78]*ms* |
| Runtime | 300*s* |

the number of requests sent by consumer $i$ and satisfied by the server (a.k.a. producer) and $localHit_i$ is the number of requests sent by consumer $i$ and satisfied by a cache of a router along the path to the producer. The cache hit ratio is probably the most fundamental metric for evaluating the performance of caching in NDN.

$$CacheHitRatio = \frac{\sum_{i=1}^{N} localHit_i}{\sum_{i=1}^{N}(localHit_i + serverHit_i)} \quad (1)$$

Conversely, an Interest that travels through all the nodes until reaching the server is counted as a server hit. The server hit ratio, given by Equation (2), is just the one complement of the cache hit ratio.

$$ServerHitRatio = \frac{\sum_{i=1}^{N} serverHit_i}{\sum_{i=1}^{N}(localHit_i + serverHit_i)} \quad (2)$$

The Content retrieval latency is the average time spent to receive the requested content. The content may be sent by an intermediate node or ultimately by its producer. The content retrieval time is calculated by Equation (3) where $R_i$ denotes the number of Interests generated by node $i$, $T_{ir}$ represents the retrieval latency taken by Interest number $r$ generated by consumer $i$, and $N$ is the number of consumers

$$AverageRetrievalLatency = \frac{\sum_{i=1}^{N} \frac{\sum_{r=1}^{R_i} T_{ir}}{R_i}}{N} \quad (3)$$

The hop reduction ratio is another important performance metric that measures the reduction of the number of hops traversed to satisfy a request compared to the number of hops needed to retrieve the content from its server. The Hop reduction ratio is calculated using Equation (4), where $h_{ir}$ denotes the number of hops traversed by Interest number $r$ generated from Consumer $i$ until being answered either by a cache or its producer, $H_{ir}$ denotes the number of hops to be traversed by Interest $r$ from consumer $i$ to the producer of the requested content, and $R_i$ denotes the number of Interests generated by node $i$.

$$HopReductionRatio = 1 - \frac{\sum_{i=1}^{N} \frac{\sum_{r=1}^{R_i} \frac{h_{ir}}{H_{ir}}}{R_i}}{N} \quad (4)$$

The cache eviction represents the total number of evicted contents from all network nodes. A content eviction happens when the content of an arriving Data packet has to be cached and the cache is full. In this case, a given content has to be evicted from the cache for the newly arriving to be cached.

### C. SIMULATION RESULTS

We here present the simulation results of the defined network scenario to show the performance of the proposed caching technique and how it compares to those of six of the well-known in-network caching strategies; namely LCE, LCD, ProCache, Btw, Edge and consumer-cache. All simulations are replicated 12 times; the 95% confidence intervals are shown on all plots.

#### 1) CACHE HIT RATIO

Cache hits occur when contents are served by intermediate caches rather than the content servers (the producers). Figure 5 illustrates the cache hit ratio for our proposed PoolCache as well as for the 6 different defined caching policies. PoolCache clearly outperforms all the other caching strategies. This accomplishment is essentially due to the zero redundancy insured by PoolCache and the pooling of all caching storage in a unique distributed managed storage space. This allows to have more storage for caching within each defined neighborhood. Consumer-cache performs the worst as it considers only caching at the node directly connected to the consumer. It is worthy to note that a cache hit is accounted for when the Interest is answered by any node along the path toward the producer. As a result, we observe that all the other caching techniques perform better than the consumer-cache. LCE fills the network caches faster, roughly increasing content replacement and reducing cache efficiency. LCD and ProbCache shows roughly the same cache performance. In the same manner as consumer-cache, edge-cache caches contents at the leaves of the topology

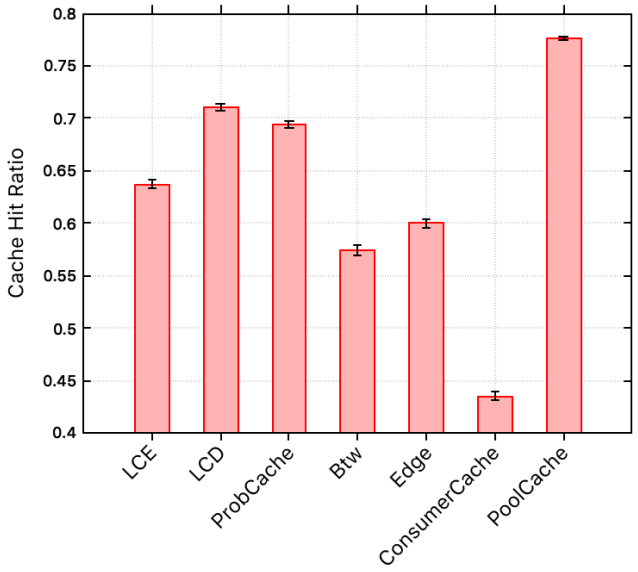



**FIGURE 5. Cache hit ratio.**

and consequently amounts to a lot of caching replacements, resulting in low cache hits. In the Btw caching strategy, cache nodes are positioned halfway along the request path and possibly closer to the cluster. The Btw strategy has a value that is always centered around the network, which yields a low cache hit ratio.

### 2) SERVER HIT RATIO

A Server (producer) hit occurs when an Interest could not be satisfied by any intermediate node along the path to the producer. That is when no intermediate node has a cached copy of the requested content. Figure 6 portrays the server hit ratios of PoolCache and the different considered caching techniques. Note that the server hit ratio is just the one complement of the cache hit ratio. Clearly, PoolCache has the lowest server ratio. It is also worth noting that PoolCache alleviates the internal nodes (a.k.a. the level 0 and level 1 routers) from the burden of caching all together. This is not the case of the LCE, LCD and betweeness strategies.

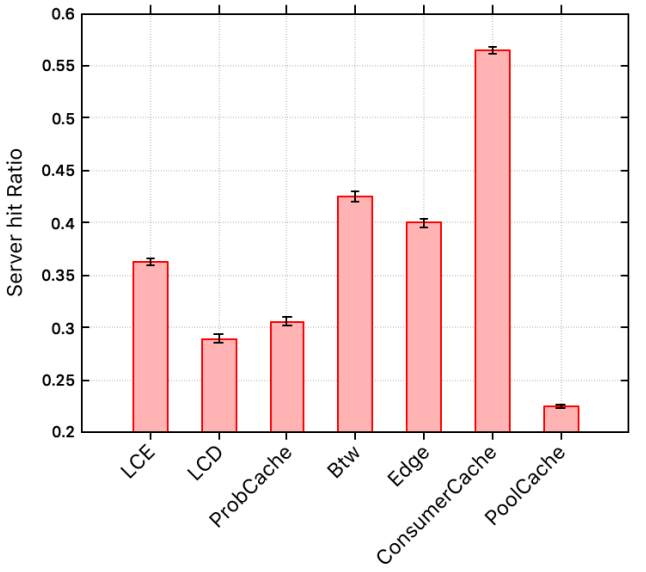



**FIGURE 6.** Server hit ratio.

### 3) RETRIEVAL LATENCY

Figure 7 represents the content retrieval latency for PoolCache and the other considered caching schemes. Clearly, PoolCache yields the lowest average response time to retrieve a content as it has a big chance to have been cached within the local cluster. Recall that PoolCache has a high cache hit ratio; meaning that a copy of the requested content has a probability equal to the cache hit ratio of being available locally from the pooled store. On the other hand, we observe that consumer-cache (and also the same for edge-cache) yields the highest retrieval time as its cache hit ratio is the lowest which in turn necessitates most of the time to bring contents from their corresponding producers. The other caching strategies have rather high retrieval times as the caching is usually done at remote nodes and not close to the consumers.

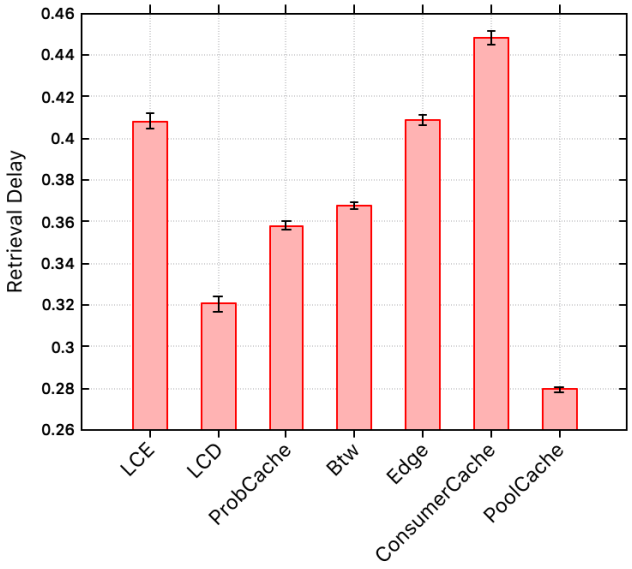



**FIGURE 7.** Response latency.

### 4) NUMBER OF EVICTION

A content eviction from the Content Store occurs when a new content is to be cached and the cache is full. This new content will replace the Least Recently Used (LRU) content. The LRU replacement strategy is here used for PoolCache and all the considered caching techniques. Figure 8 portrays the number of evictions encountered during the simulations of each caching technique. PoolCache yields the lowest number of eviction as it uses a pooled store. LCE yields the highest eviction rate since the same contents are cached everywhere. Consumer-cache and Edge-cache show high eviction rate since they cache at the consumers which yields high cache misses. The other techniques show rather low eviction rate but still much higher than that of PoolCache.

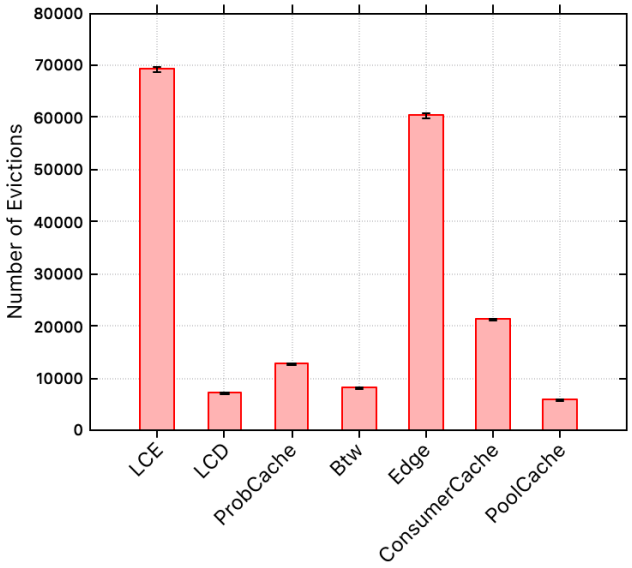



**FIGURE 8.** Number of eviction.

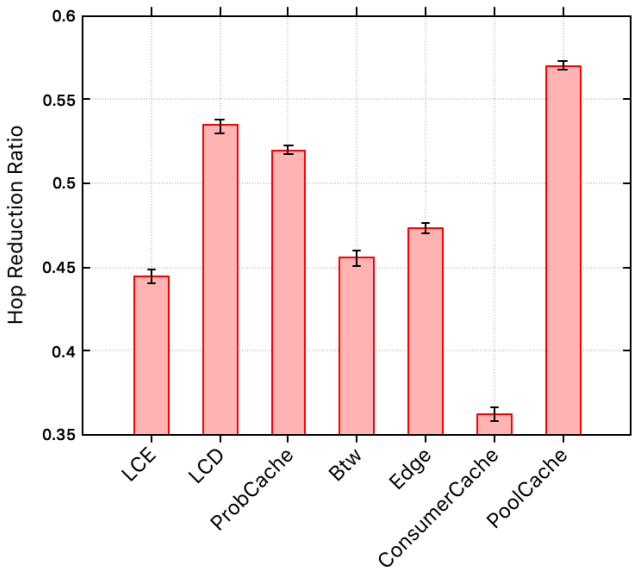



**FIGURE 9.** Hop reduction ratio.

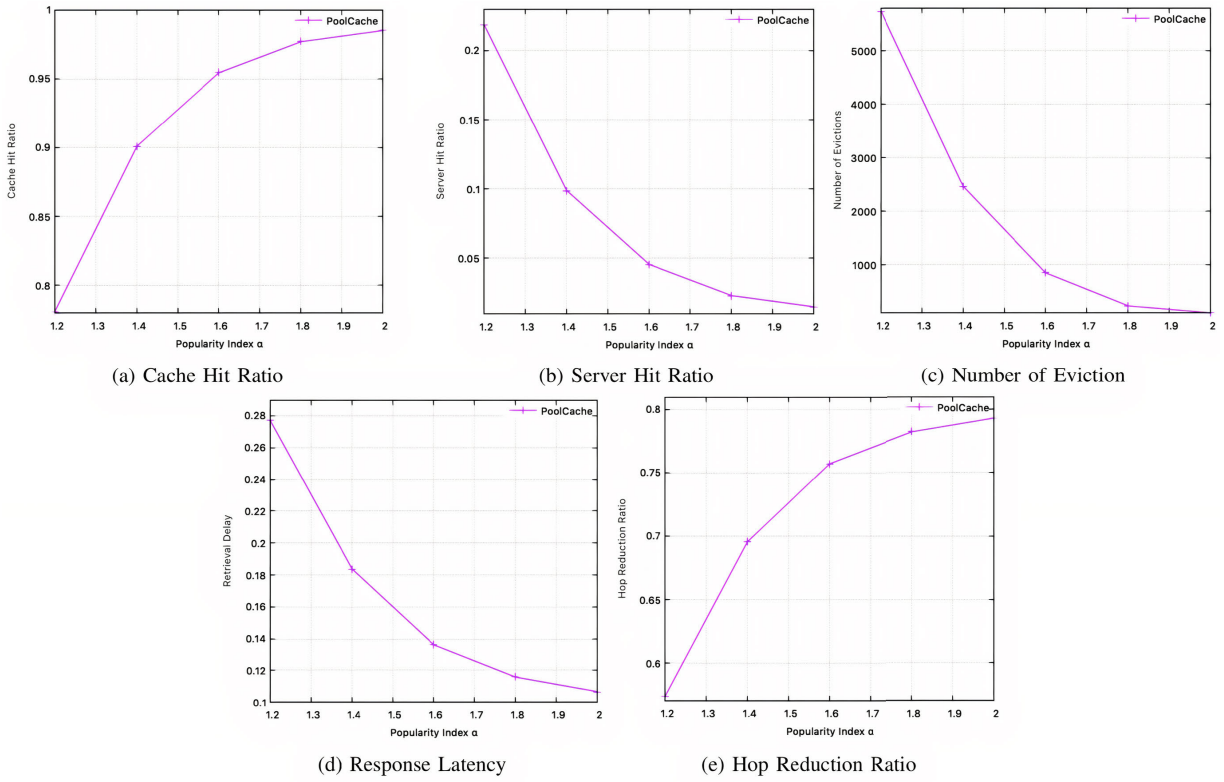
### 5) HOP REDUCTION RATIO

The hop reduction ratio is inversely proportional to the server hit ratio. For a larger server hit ratio, we normally obtain a larger hop count and therefore a smaller hop reduction ratio. The hop reduction ratios for the different caching techniques are represented on Figure 9. PoolCache outperforms all the considered techniques; as a matter of fact its server hit ratio is the smallest (see Figure 6). Consumer-cache yields the highest hop reduction ratio since generated Interest are mostly answered by their corresponding producers amounting to the highest server hit ratio as shown on Figure 6. ProbCache and LCD roughly perform the same, however better than Btw and Edge for the same reason.

## V. IMPACT OF CONTENT POPULARITY ON PoolCache

The conducted simulations assumed that Interests are generated at each consumer following the Independent Request Model (IRM), and the contents requested in these Interests follow the Zipf probability distribution model with a skewness or popularity parameter $\alpha = 1.2$. This parameter plays a major role as it determines the shape of the cumulative probability function and regulates the deterioration in requests frequencies. Virtually all previous studies on web caching in the Internet considered a value less or equal to 1. Recently however, new research work and evidence showed that popularity has become more important than before, and values of $\alpha > 1$ are becoming more common due to the proliferation of the Internet of Things and the pervasive spread of social networking [38]–[40]. The popularity is then more concentrated than before, and as a result caching becomes a more profitable approach. In [38], the authors analyzed 14 websites and showed that all of them provided a value of $\alpha > 1$.

Figure 10 illustrates the impact of an increasing popularity on the different performance metrics of PoolCache. We clearly observe that the cache hit ratio and the hop reduction ratio increases as the popularity index increases. Conversely, the server hit ratio, the number of evictions and




(a) Cache Hit Ratio (b) Server Hit Ratio (c) Number of Eviction

(d) Response Latency (e) Hop Reduction Ratio

**FIGURE 10.** The impact of content popularity on PoolCache.

the content average retrieval delay decrease as the popularity index increases.

Indeed when the popularity index increases, requests for the most popular contents become very frequent. PoolCache allows then to cache virtually all popular contents in its local pooled store. This tacitly yields much better performances. Unpopular contents are rarely requested and are quickly evicted and replaced by more popular contents.

## VI. CONCLUSION

NDN-IoT provides an enabling technology and a current networking architecture for the proliferation of IoT applications. Despite the many proposed caching techniques, efficient in-network caching with limited resources remains a real challenge. The storage capacity reserved for caching arriving contents at each node is rather very limited relatively to the colossal number of contents. The question naturally arises as to how to increase this storage capacity with no additional storage resource. This is mainly the rationale behind the proposed PoolCache.

Poolcache pools the dedicated caching storage of collections of designated nodes. PoolCache manages the storage pool of each collection in a collaborative manner and insures zero content redundancy with each pooled storage. As such, PoolCache increases the content diversity and content availability within the network as it allows much more different contents to be cached. The nodes belonging to each collections should be geographically within the same neighborhood such as the same stub. There is, however, no necessity or obligation to include or involve all and every node within the same neighborhood. In the very extreme case, each collection or neighborhood contains just a single node. In this latter case, PoolCache behaves as the Edge-Cache strategy. Conducted simulations using the ccnSim show that PoolCache clearly outperforms many know caching decision algorithms. Simulations results also show that PoolCache profits from any eventual content popularity and provides sustained performance.

## REFERENCES


[1] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, "Recent advances in information-centric networking based Internet of Things (ICN-IoT)," 2017, *arXiv:1710.03473*. [Online]. Available: http://arxiv.org/abs/1710.03473

[2] G. M. D. T. Forecast, "Cisco visual networking index: Global mobile data traffic forecast update, 2017–2022," Cisco, San Jose, CA, USA, White Paper, Feb. 2019, p. 2022.

[3] V. Jacobson, J. Burke, D. Estrin, L. Zhang, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, and P. Ohm, "Named data networking (NDN) project 2012–2013 annual report," Named Data Netw. (NDN), Shanghai, China, Tech. Rep., 2013.

[4] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, "Recent advances in information-centric networking-based Internet of Things (ICN-IoT)," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2128–2158, Apr. 2019.

[5] M. Amadeo, C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar, and A. V. Vasilakos, "Information-centric networking for the Internet of Things: Challenges and opportunities," *IEEE Netw.*, vol. 30, no. 2, pp. 92–100, Mar. 2016.

[6] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and S. Al-Ahmadi, "Named data networking: A promising architecture for the Internet of Things (IoT)," *Int. J. Semantic Web Inf. Syst.*, vol. 14, no. 2, pp. 86–112, 2018.

[7] A. Aboodi, T.-C. Wan, and G.-C. Sodhy, "Survey on the incorporation of NDN/CCN in IoT," *IEEE Access*, vol. 7, pp. 71827–71858, 2019.

[8] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, "Producer mobility support in named data Internet of Things network," *Procedia Comput. Sci.*, vol. 109, pp. 1067–1073, Jan. 2017.

[9] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and S. Gannouni, "AFIRM: Adaptive forwarding based link recovery for mobility support in NDN/IoT networks," *Future Gener. Comput. Syst.*, vol. 87, pp. 351–363, Oct. 2018.

[10] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, "How to cache in ICN-based IoT environments?" in *Proc. IEEE/ACS 14th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Oct. 2017, pp. 1117–1124.

[11] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and S. Alahmadi, "Cache freshness in named data networking for the Internet of Things," *Comput. J.*, vol. 61, no. 10, pp. 1496–1511, Oct. 2018.

[12] R. Chiocchetti, D. Rossi, and G. Rossini, "CcnSim: An highly scalable CCN simulator," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 2309–2314.

[13] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, "Named data networking for IoT: An architectural perspective," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2014, pp. 1–5.

[14] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information centric networking in the IoT: Experiments with NDN in the wild," in *Proc. 1st Int. Conf. Inf.-Centric Netw. INC*, 2014, pp. 77–86.

[15] Y. Zhang, D. Raychadhuri, L. Grieco, E. Baccelli, J. Burke, R. Ravindran, and G. Wang, "ICN based architecture for IoT—Requirements and challenges," IETF: Internet Draft, draft-zhang-iot-icn-challenges-02, Fremont, CA, USA, Tech. Rep., 2016.

[16] I. U. Din, S. Hassan, M. K. Khan, M. Guizani, O. Ghazali, and A. Habbal, "Caching in information-centric networking: Strategies, challenges, and future research directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1443–1474, 2nd Quart., 2018.

[17] N. Laoutaiis, S. Syntila, and L. Stavrakakis, "Meta algorithms for hierarchical Web caches," in *Proc. IEEE Int. Conf. Perform., Comput., Commun.*, Apr. 2004, pp. 445–452.

[18] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Perform. Eval.*, vol. 63, no. 7, pp. 609–634, Jul. 2006.

[19] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd Ed. ICN Workshop Inf.-Centric Netw. - ICN*, 2012, pp. 55–60.

[20] W. K. Chai, D. He, H. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks," in *Networking* (Lecture Notes in Computer Science), vol. 7289. Berlin, Germany: Springer, 2012, pp. 27–40.

[21] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. C. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable ICN," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 147–158.

[22] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, "Cache coherence in Machine-to-Machine information centric networks," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 430–433.

[23] J. W. Wang Sen, J. Bi, and A. V. Vasilakos, "CPHR: In-network caching for information-centric networking with partitioning and hash-routing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2742–2755, Oct. 2016.

[24] S. Wang, J. Bi, and J. Wu, "Collaborative caching based on hash-routing for information-centric networking," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 535–536.

[25] C. Li and K. Okamura, "Cluster-based in-networking caching for content-centric networking," *Int. J. Comput. Sci. Netw. Secur.*, vol. 14, no. 11, p. 1, 2014.

[26] Y. Sato, Y. Ito, and H. Koga, "Hash-based cache distribution and search schemes in content-centric networking," *IEICE Trans. Inf. Syst.*, vol. E102.D, no. 5, pp. 998–1001, 2019.

[27] H. Yan, D. Gao, W. Su, C. H. Foh, H. Zhang, and A. V. Vasilakos, "Caching strategy based on hierarchical cluster for named data networking," *IEEE Access*, vol. 5, pp. 8433–8443, 2017.

[28] J. H. Mun and H. Lim, "Cache sharing using Bloom filters in named data networking," *J. Netw. Comput. Appl.*, vol. 90, pp. 74–82, Jul. 2017.

[29] M. Amadeo, G. Ruggeri, C. Campolo, and A. Molinaro, "Diversity-improved caching of popular transient contents in vehicular named data networking," *Comput. Netw.*, vol. 184, pp. 92–100, Oct. 2021.

[30] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the Internet hierarchy from multiple vantage points," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, Jun. 2001, pp. 618–627.

[31] K. Pentikousis, B. Ohlman, E. Davies, S. Spirou, and G. Boggia, *Information-Centric Networking: Evaluation and Security Considerations*, document IETF: RFC 7945, Tech. Rep., 2016.
[32] P. V. Rani and S. M. Shalinie, "Efficient cache distribution using hash-routing schemes and nodal clustering for information centric network," in *Proc. 4th Int. Conf. Signal Process., Commun. Netw. (ICSCN)*, Mar. 2017, pp. 1–6.
[33] B. Donnet and T. Friedman, "Internet topology discovery: A survey," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 4, pp. 2–15, 4th Quart., 2007.
[34] H. Haddadi, S. Uhlig, and A. Moore, "Modeling Internet topology dynamics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 65–68, 2008.
[35] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
[36] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing," Telecom ParisTech, Paris, France, Tech. Rep., 2011.
[37] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, K. Drira, and H. Mathkour, "Least fresh first cache replacement policy for NDN-based IoT networks," *Pervas. Mobile Comput.*, vol. 52, pp. 60–70, Jan. 2019.
[38] M. G. Zotano, J. G. Sanz, and J. Pavón, "Analysis of Web objects distribution," in *Proc. 12th Int. Conf. Distrib. Comput. Artif. Intell.*, vol. 373. Salamanca, Spain: Springer, Jun. 2015, pp. 105–112.
[39] Naeem, Nor, Hassan, and Kim, "Compound popular content caching strategy in named data networking," *Electronics*, vol. 8, no. 7, p. 771, Jul. 2019.
[40] J. Zhi, J. Li, H. Wu, and Y. Ren, "GAC: Gain-aware 2-round cooperative caching approach in information-centric networking," in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2018, pp. 1–8.

**BASHAER ALAHMRI** received the B.Sc. degree in computer science from Princess Nourah Bint Abdulrahman University and the M.Sc. degree from the Department of Computer Science, King Saud University, Riyadh, Saudi Arabia. Her research interests include information-centric networks, next-generation network architecture, and the Internet of Things.

**SAAD AL-AHMADI** (Member, IEEE) is currently an Assistant Professor with the Department of Computer Science, King Saud University, Saudi Arabia. He has published many articles in highly cited journals and worked as a part-time Consultant in several government organizations as well as the private sector. His current research interests include the IoT security, machine learning for healthcare, and future generation networks.

**ABDELFETTAH BELGHITH** (Senior Member, IEEE) received the Master of Science and Ph.D. degrees in computer science from the University of California at Los Angeles (UCLA), in 1982 and 1987, respectively. Since 1992, he has been a Full Professor with the National School of Computer Sciences (ENSI), University of Manouba, Tunisia. He is currently on a sabbatical leave at King Saud University, Saudi Arabia. His research interests include computer networks, wireless networks, multimedia Internet, mobile computing, distributed algorithms, systems and information security, simulation, and performance evaluation. He runs several research projects in cooperation with other universities, research laboratories, and research institutions. He has published more than 350 research papers in highly ranked journals and conference proceedings. He is the Past Chair of the IEEE Tunisia Section, the Chair of the IEEE ComSoc and VTS Tunisia Chapters, and the Director of the HANA Research Laboratory, National School of Computer Sciences.