Easy as ABC: A Lightweight Centrality-Based Caching Strategy for Information-Centric IoT

Jakob Pfender
Sch of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
jakob.pfender@ecs.vuw.ac.nz

Alvin Valera
Sch of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
alvin.valera@ecs.vuw.ac.nz

Winston K.G. Seah
Sch of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
winston.seah@ecs.vuw.ac.nz

ABSTRACT

In Information-Centric Networking (ICN), the ability to cache content at multiple points in the network is one of the most important factors in the speed and reliability of content delivery. However, in the constrained environment of the Internet of Things (IoT), memory is often a scarce resource, which means that particular focus needs to be placed on how to use the available memory for caching. Previous research has shown that caching heuristics that take network topology into account have great promise, but are often not feasible for use in the IoT as they typically incur high overheads or require extensive knowledge of the topology. We introduce a simple content caching strategy called Approximate Betweenness Centrality (ABC), which makes use of the topology-based heuristics of existing strategies, but requires no knowledge of the network and incurs no communications overhead. We compare this new strategy to several existing ICN caching strategies and evaluate its effectiveness using real IoT devices in a large physical testbed. We show that our lightweight approach can deliver results that are comparable to those of more expensive strategies while incurring almost no additional costs.

CCS CONCEPTS

• **Networks → Network protocol design**; **Network experimentation**; **Network performance analysis**.

KEYWORDS

Information-centric networking, IoT, caching, network topology

ACM Reference Format:
Jakob Pfender, Alvin Valera, and Winston K.G. Seah. 2019. Easy as ABC: A Lightweight Centrality-Based Caching Strategy for Information-Centric IoT. In *6th ACM Conference on Information-Centric Networking (ICN '19), September 24–26, 2019, Macao, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3357150.3357405

1 INTRODUCTION

Information-Centric Networking (ICN) is a promising future network architecture for the Internet of Things (IoT), as its content-centric nature is uniquely suited to the nature of typical IoT applications and its slim network stack [23] means a smaller memory footprint for the often resource-constrained devices used in IoT [1, 31, 46].

However, unlike traditional ICN, where network participants typically have sufficient access to resources such as memory or processing power and battery life is not an issue, most devices used in IoT have to contend with severe limitations in this regard. Therefore, the indiscriminate caching of all incoming content, as is the default in ICN, is not feasible in this domain. Instead, a number of strategies have been proposed that aim to decide what content to cache at which nodes based on various heuristics. While a number of effective caching strategies for generalised ICN deployments have been proposed, most still assume a level of resource availability that is simply unsuitable for the IoT [3–5, 25, 26]. They operate on the assumption that nodes will have enough memory available to cache indiscriminately and that there is always sufficient bandwidth available, neither of which is guaranteed in the IoT. Therefore, finding caching strategies that take the idiosyncrasies of the IoT such as constrained devices and limited bandwidth into account is an ongoing endeavour in information-centric IoT research.

There are several aspects of network performance that can be improved by employing a suitable caching strategy. Storing cached copies of popular content at multiple locations in the network can decrease network load by alleviating the strain on individual content producers and reducing the number of hops required to reach a cache hit, and increased redundancy improves network stability in the case of node failure, unreliable links, or network partition. However, in this study, we will be mainly focusing on one specific performance measure: *content delivery latency*. Many time-critical IoT applications rely on content being delivered as fast as possible, and the way in which content is cached throughout the network can have a significant impact on how quickly relevant information can be disseminated to where it is required. An effective caching strategy in this regard is one that minimises the distance between consumer and producer, and the aim of this study is to develop a strategy that can achieve this under the constraints imposed by IoT hardware.

In this study, we will be exploring the effects of the network topology on what constitutes the ideal caching location (Section 2), and to that end discuss a pair of caching strategies that aim to take advantage of topological effects to maximise the benefits of in-network caching (Section 3). We then introduce ABC, a novel caching algorithm that builds on the same centrality concept, but is specifically geared towards deployment in the IoT and thus has significantly reduced overhead (Section 4). We evaluate the proposed strategy in comparison with established caching strategies in an experiment using physical hardware (Section 5), discuss related work (Section 6), and finally present our conclusions and discuss potential future work (Section 7).

2 TOPOLOGY EFFECTS

In the real world of IoT deployments, there are as many different network topologies as there are networks. However, to demonstrate the effects of topology on caching performance, we will be focusing on two generalised cases — the *core topology* and the *edge topology* — that are on opposite ends of a spectrum of topology types. There is no formal definition for either of these topology types, and not every topology can be clearly classified into one or the other category. Rather, we use the terms *core topology* and *edge topology* to describe extreme cases (elaborated below). IoT topologies, while falling anywhere on the spectrum between these extremes, tend to resemble one type more strongly than the other. We focus on the extremes here because if a strategy performs well in both cases, it is reasonable to assume that it is topology-independent.

Figure 1 shows an example of a *core topology.* A core topology is defined by the paths between the producer and the consumers intersecting closer to the producer (the "core"); each path from the core outward has only one consumer attached to it at the edge. In such a topology, the ideal caching location would be close to the producer, as this would alleviate strain on the producer while serving the maximum number of consumers with cached copies of the data (Wang *et al.* call caching policies that achieve this effect *Type III* policies [44]). Conversely, caching close to the consumer would decrease the content delivery latency for that consumer, but no other consumer would gain any benefit from the cached copy. Note that the topology shown here is not the most extreme core topology possible; that would be a full star topology in which all paths from the core connect to exactly one edge node. However, in such a topology, there would be almost no benefit from caching outside of the producer.

Figure 2 shows an example of an *edge topology.* In such a topology, paths intersect near the consumers (the "edge") rather than the producer. In other words, multiple leaf nodes tend to share a direct parent node. This parent node would be an ideal caching location to serve all leaf nodes connected to it, as this reduces the need for requests from the edge to be routed all the way to the core (a *Type II* policy [44]). Conversely, caching closer to the core would just alleviate the strain on the producer, with minimal latency improvements.

Looking at these two cases, it is easy to see that the network topology has a significant effect on where content should be cached if latency is to be minimised. For this reason, as we have shown previously [35], caching strategies that emphasise network topology have the potential to be more effective than strategies that ignore the caching nodes' relative positions in the network.

3 CENTRALITY-BASED CACHING

In this section, we will be discussing a class of ICN caching strategies that use the caching node's *centrality* (more specifically, its *betweenness centrality*) to decide whether to cache content. Betweenness centrality [45] describes the number of times a given node lies on one of the paths between all pairs of nodes in the network. In general, the betweenness centrality $C_B(v)$ of a node $v \in V$, where V is the set of all nodes in the network, is defined as follows:

$$C_B(v) = \sum_{i \neq v \neq j \in V} \frac{\sigma_{i,j}(v)}{\sigma_{i,j}}, \tag{1}$$

where $\sigma_{i,j}$ is the total number of paths between two nodes i and j ($i \neq v \neq j$) and $\sigma_{i,j}(v)$ is the number of paths between i and j that pass through v. This definition accounts for the possibility that there are multiple paths between i and j. However, in ICN, without loss of generality we assume that the shortest path between i and j is always used as the content delivery path. Therefore, we can simplify the definition as follows:

$$C_B(v) = \sum_{i \neq v \neq j \in V} \sigma'_{i,j}(v), \tag{2}$$

where

$$\sigma'_{i,j}(v) = \begin{cases} 1, & \text{if } v \text{ on path } (i,j) \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Betweenness centrality has been found to be a useful indicator of node importance in a network [43]. We can apply this to ICN caching by arguing that caching at more "important" (i.e.: central) nodes will be beneficial for caching performance as it increases reachability of content and thus should increase cache hits and reduce content delivery latency. This is the motivation for the work of Chai *et al.* [15], which we will introduce now.

The basic concept of the centrality caching strategies proposed by Chai *et al.* is that when a content chunk is sent from node i to node j, it shall be cached at the node v with the highest centrality value $C_B(v)$ among all nodes on the path (i, j). This is achieved in practice by extending all ICN packets to include a field for the centrality value. Interest packets will then use this field to record the highest centrality value they encounter en route to their destination. This value is then recorded in the Data packet that is returned, and a node on the return path caches the Data if and only if its own centrality is equal to or higher than the centrality value recorded in the Data packet. This mechanism is illustrated in Algorithm 1. The functions `handle_interest()` and `handle_data()` define the behaviour when a node receives an Interest or Data packet, respectively. If the incoming Interest can be satisfied locally, `canSatisfy()` returns `true`, otherwise `false`. Content chunks are retrieved from the local Content Store (CS) using `getData()`; and

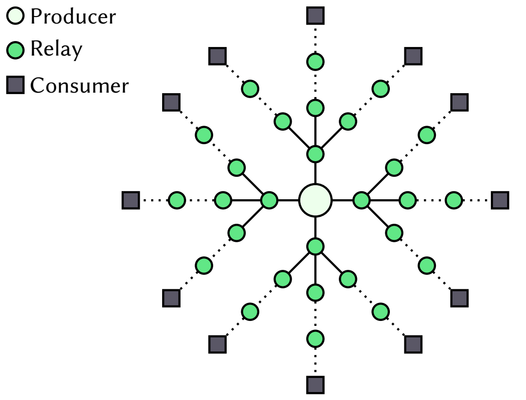


Figure 1: Core topology

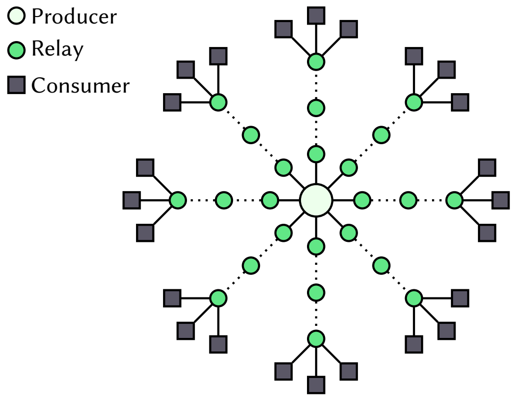


Figure 2: Edge topology

Algorithm 1 Betw/EgoBetw caching decision

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          Data.Centrality ← Interest.Centrality
5:          reply(Data)
6:      else
7:          if myCentrality > Interest.Centrality then
8:              Interest.Centrality ← myCentrality
9:          end if
10:         forward(Interest)
11:     end if
12: end function
13:
14: function HANDLE_DATA(Data)
15:     if myCentrality ≥ Data.Centrality then
16:         cache(Data)
17:     end if
18:     forward(Data)
19: end function
```

`reply()`, `forward()`, and `cache()` correspond to the ICN primitives for replying to an Interest with a Data packet, forwarding packets to the next hop, and caching content.

All centrality caching strategies discussed in this paper use the same caching mechanism as described above. The difference is in how they calculate the value for the betweenness centrality that is used in the caching decision. Chai *et al.* propose two variants: *Betw* and *EgoBetw* [15].

Betw is a straightforward implementation of the betweenness centrality measure as described by Wasserman and Faust [45]. Before nodes begin exchanging Interests and Data, there is a setup phase in which all nodes are assigned a centrality value using Eq. (2). The authors do not specify how exactly this setup phase is realised; in a fully static topology where delivery paths never change, it may be feasible to simply manually assign the correct centrality value to every node *a priori*. However, it is more likely that the nodes themselves will have to exchange neighbour information in such a way that every node in the network has full information about every other node in the network. This implies a significant overhead, in terms of communications (exchanging all of the neighbour lists), memory (storing neighbour information for the whole network), computational (converting the neighbour information into a centrality value), and time (waiting until every node has full knowledge of the network).

As the complexity of *Betw* is high along multiple dimensions, Chai *et al.* propose a more lightweight alternative called *EgoBetw*. Instead of using global knowledge, this method calculates an approximation of a node's centrality value by having it exchange connectivity information only with its one-hop neighbours. This approach is based on the concept of ego network betweenness [20]. A node's ego network is defined as that node, its one-hop neighbours, and all links between any of those nodes. A node's ego betweenness centrality is thus the number of times it lies on one of the paths between all pairs of nodes in its ego network. The calculation is the same as in Eq. (2), except that V denotes the node's ego network instead of the whole network. Chai *et al.* show that a node's ego betweenness is a reasonable approximation of the real betweenness measure.

For both *Betw* and *EgoBetw*, if we assume that the centrality value is to be calculated entirely on the nodes without any *a priori* knowledge, each node needs to flood its own Forwarding Information Base (FIB) entries to all other nodes in the (ego) network, which equates to a baseline of at least n broadcast messages, where n is the number of nodes in the entire network. If we use *Betw* in a multi-hop environment, the initial broadcast will not reach every other node in the network, necessitating further transmissions. In the worst case, up to n further retransmissions are necessary, thus placing *Betw*'s messaging overhead in $O\left(n^2\right)$. In *EgoBetw*, nodes only need to exchange and store the neighbour information of their immediate neighbours. This means that each node only needs to send one broadcast message; the messaging overhead of *EgoBetw* is thus in $O\left(n\right)$, where n is the number of nodes in the network.

Since every node requires full knowledge about all pairs of nodes in the (ego) network to calculate its centrality, the memory overhead per node is in $O\left(n^2\right)$ for *Betw* and in $O\left(d^2\right)$ for *EgoBetw*, where d is that node's degree ($d \leq n - 1$).

In order to calculate its centrality value, each node has to check whether it is on the path between each pair of nodes, thus placing the computational complexity in $O(n^2)$ for *Betw* and in $O(d^2)$ for *EgoBetw*.

Further complexity arises if the topology is dynamic, either because of unstable links resulting in variable delivery paths or because of mobile participants. In this case, the exchange of neighbour information needs to be repeated at a frequency that matches the frequency of changes to the topology, thus incurring further overhead.

Given the severe limitations of IoT deployments in terms of memory space and link stability, it is highly questionable whether an implementation of *Betw* that carries out the centrality calculations on the nodes themselves can be realistically considered. *EgoBetw* is more feasible thanks to reduced knowledge requirements and complexity; however, the overhead induced by the exchange of neighbour information, although only link-local, is still significant, and especially in a dynamic topology may result in unacceptable contention of the wireless medium.

4 APPROXIMATE BETWEENNESS CENTRALITY

Given the fact that, as shown above, *Betw* and *EgoBetw* are difficult or even impossible to implement on IoT hardware, our aim in this work is to develop a method of determining node centrality that approximates the results of the existing strategies while subject to the constraint that it must be feasible to implement and run on typical IoT hardware with extremely limited memory, bandwidth, and processing power.

To that end, we now present our new contribution to centrality-based ICN caching: Approximate Betweenness Centrality (ABC). ABC handles the caching decision in the same way as *Betw* and *EgoBetw* do (i.e., content is always cached at the nodes with the highest centrality on the return path) but the centrality calculation does not incur the communications, storage, and computational overhead inherent in the other strategies.

4.1 Operation

Instead of relying on a costly setup phase, ABC has each node approximate its own centrality during runtime using information carried in the packets they receive. We assume that a prefix owned by a producer contains some information that uniquely ties it to that producer. This can take the form of a unique ID, a physical or logical address, a location identifier, or similar. Furthermore, we extend Interest packets to carry the unique identifier of the original requesting node as metadata. The question of whether the assumption of unique identifiers and the extension of Interest packets in this fashion are reasonable in information-centric IoT will be addressed in Section 4.2.

Embedding producer and consumer identifiers in Interest packets enables each node that handles an Interest to ascertain that it is on the path between the consumer and the producer of that Interest. This is equivalent to the knowledge a node in *Betw* would have about whether it is on the path between a given pair of nodes. Every Interest from a new producer and/or to a new consumer would thereby increase the node's knowledge about which delivery paths it is on. Thus, by keeping track of which pairs of nodes it serves, each node can over time approximate a centrality value for itself, which will eventually converge to the value that would have been calculated by *Betw*/*EgoBetw* in the setup phase. Of course, in terms of pure performance, this convergence time represents a disadvantage as *Betw*/*EgoBetw* can make use of fully accurate centrality values from the start; however, these require an *a priori* setup phase that is completely eliminated by ABC.

Algorithm 2 Centrality Approximation

```
1: function UPDATE_CENTRALITY(Source, Destination)
2:     if (Source,Destination) $\notin$ myPaths then
3:         myPaths $\leftarrow$ myPaths + (Source,Destination)
4:         myCentrality $\leftarrow$ myCentrality +1
5:     end if
6: end function
```

Algorithm 2 shows how ABC approximates a node's centrality. The `update_centrality()` function is called at the start of the `handle_interest()` function; the rest of the caching logic is identical to Algorithm 1.

4.2 Analysis

As all information required by ABC is piggy-backed onto the Interest packets that are being sent anyway, there is no need for any additional broadcast messages or any other exchange of information. Thus, ABC effectively eliminates the messaging overhead by reducing it to $O(1)$.

Nodes no longer require knowledge about all pairs of nodes in the (ego) network; instead, they only need to count the absolute number of paths they are on. In the worst case (if all paths in the network were to pass through a given node) this equates to a memory overhead of $O(p)$, where p is the number of paths in the network[1] ($p \leq n(n-1)$). In a realistic topology, a node will only ever be on a subset of paths in the network, and the required memory is bounded by the number of paths it is on. This means that edge nodes will use close to no additional memory, while central nodes may use more. The actual memory overhead observed in our experiments will be discussed in Section 5.

In ABC, there is no need to compute nodes' centrality values by checking their presence or absence on every path in the network. Instead, nodes simply increment their centrality values whenever they see a new path in an incoming Interest. The computational complexity of ABC is therefore $O(1)$.

For ease of comparison, the overheads of *Betw*, *EgoBetw*, and ABC are shown in Table 1.

Section 3 mentions as a compounding problem the issue of dynamic topologies. Both *Betw* and *EgoBetw* rely on an exchange of information and subsequent calculation of centralities that is separate from regular ICN operations, likely in the form of an *a priori* setup phase, and has a static result. This means that this step, along with the communications and computational overhead it incurs, would need to be repeated whenever there is a change in topology (for *Betw*, the entire network needs to be re-evaluated, while in *EgoBetw* this is limited to the ego networks directly affected by the topology change). Depending on the deployment scenario, changes

[1] We assume that a path (i, j) is not necessarily identical to the corresponding path (j, i) if $i \neq j$, since FIB entries are generated independently from one another and there is no guarantee that they will be symmetrical.

Strategy	Messaging overhead	Memory overhead	Computational overhead
Betw	$O(n^2)$	$O(n^2)$	$O(n^2)$
EgoBetw	$O(n)$	$O(d^2)$	$O(d^2)$
ABC	$O(1)$	$O(p)$	$O(1)$

Table 1: Messaging, memory, and computational overheads of the centrality-based caching strategies, where n is the number of nodes in the network, d is the degree of the caching node, and p is the number of paths in the network

in topology may be frequent. This means that the already significant overheads of these strategies will grow even further. ABC, on the other hand, can easily accommodate dynamic topologies by using time-outs for the information stored on recorded paths. These time-outs could even be adjusted dynamically according to the frequency of topology changes. Since this mechanism still only uses information from existing Interest packets, the overheads described above are not affected by this change.

A further advantage of ABC is ease of implementation. Requiring only a simple extension of Interest and Data packets by one field and an additional code block in the Interest handler to count node pairs, it is uncomplicated to extend an existing ICN deployment to use ABC. *Betw* and *EgoBetw*, on the other hand, require provisioning for the setup phase (which can not rely on ICN infrastructure) and, in case computations can not be handled on the nodes due to hardware constraints, a way to offload the determination of betweenness values to a central controller.

In Section 4.1, we mentioned that we rely on Interests that clearly identify both their producer and their consumer. This carries both a strong assumption (that a singular source exists for each prefix) and a break in ICN principles (carrying consumer information in Interest packets), which need to be addressed.

In the domain of information-centric IoT, the assumption of a single node source for each prefix is not universally true, but also not unrealistic as nodes in typical IoT deployments usually have either clearly defined roles (such as being associated with a uniquely identified sensor/actuator or room) or a defined physical or logical location identifier. There may be cases in which a prefix is jointly owned by a group of nodes (e.g. in environmental monitoring where several nodes may be tied to the same region); however, we would argue that the operation of ABC would not be significantly hindered by this as we could simply treat this group of nodes as one producer for its path counting purposes.

The break with ICN principles would present a problem if we wanted to deploy ABC in the wider Internet; however, in the domain of IoT, we generally assume that our deployment is siloed behind a gateway, meaning that protocol breaks that are confined to the deployment are less of an issue. It must however be noted that our extension would not be able to interoperate with services relying on anonymity.

Ultimately, ABC's contribution to centrality-based caching is simple, comprising only of a way to approximate centrality values, but it is precisely this simplicity that makes it so promising. It reduces complexity across several dimensions, including the cost of implementation, as it simply leverages information from existing traffic during runtime.

5 EVALUATION

This section presents a comparison of the ABC caching strategy with a number of other ICN caching strategies. We focus on hop reduction and latency as our performance metrics, as content delivery latency is typically the most important factor in time-critical IoT applications. We also discuss the cache hit rate as this is an important measure for any caching strategy.

It is important to note that for all metrics evaluated in this section, ABC is not expected to outperform *Betw/EgoBetw* directly. In fact, this would be rather surprising as ABC relies on an inherently less accurate centrality measure for its caching decision. The goal of these experiments is to explore whether ABC's performance is acceptably close to that of *Betw/EgoBetw*, which coupled with its significantly reduced complexity would make it a very promising candidate for deployment on constrained devices.

5.1 Experiment Setup

In order to compare ABC to the other strategies, we ran a series of experiments on the **FIT IoT-LAB** [2] open testbed. As our IoT hardware, we used IoT-LAB's specially developed **M3 node**[2], which is a class 2 [9] device that has an STM32 (ARM Cortex M3) microcontroller and an Atmel AT86RF231 2.4 GHz transceiver [6]. The M3 node has 64 kB RAM and 512 kB ROM. As firmware for the nodes, we use a simple **RIOT-OS** [7] application using **CCN-lite**[3] as the ICN implementation, modified to support the different caching strategies.

The experiments were conducted on the *Grenoble* site[4] of the IoT-LAB testbed. The site features about 380 M3 nodes, which are distributed across the rooms and corridors of one floor of an office building. This means that nodes are subject to realistic conditions found in indoor IoT deployments, such as multipath effects, reflection, and absorption caused by walls, doors, and windows made of various materials, as well as unpredictable interference by other wireless signals and people moving around the building. These conditions mean that data gathered will be very close to what might be expected in a real-world deployment.

Of the 380 available nodes, each experiment run is conducted on an arbitrary subset of 50 nodes (chosen randomly each time), each of which act as producers, consumers, and relays at the same time. The transmission range of individual nodes is not enough to reach all other nodes in the building, so communication will be predominantly multihop. The mean path length is between 3 and 4 hops and the maximum is 6 hops. This kind of multihop setup is commonly found in the industrial monitoring domain.

[2]https://github.com/iot-lab/iot-lab/wiki/Hardware_M3-node
[3]https://github.com/cn-uofbasel/ccn-lite
[4]https://www.iot-lab.info/deployment/grenoble/

Cache sizes are kept intentionally small; each node's cache can store up to 5 unique content chunks (all content chunks have the same size). This small cache size was chosen for two reasons. For one, RAM is extremely limited in IoT devices. The M3 nodes used in this experiment have 64 kB of RAM. A constant fraction of this RAM is occupied by the operating system (4.4 kB) and the CCN-lite network stack (8.7 kB) [23], leaving about 50 kB that have to be shared between the CCN-lite heap (comprising CS, FIB, and Pending Interest Table (PIT)), and the actual application running on top of the network stack. However, these numbers are at the upper end of typical RAM sizes for class 2 devices. We also need to consider class 1 devices with RAM in the order of 10 kB [9]. In these devices, the OS and network stack already need to be pruned for features, and the remaining CCN-lite heap size will be at most 1 kB [8]. Depending on the nature of the data transmitted by the application, available cache space may thus be severely limited. This motivates our decision to limit the number of CS entries in this way in order to be able to assess expected performance under these conditions.

The secondary motivation for limiting the number of CS items to 5 is that many adverse effects of ICN content availability could simply be countered by over-provisioning, i.e. providing more cache space (if the available RAM allows), thus ensuring content distribution. This means that performance differences between caching strategies become less pronounced as cache size increases. Therefore, it is more interesting to look at performance under limited cache sizes, since this is where differences will be most noticeable.

The experiment is managed by a control script using the IoT-LAB API, which has access to all node caches, outputs, and inputs. The API can execute shell commands on individual nodes, which is used to provide the *a priori* topology knowledge required by the *Betw*/*EgoBetw* algorithms. This allows us to circumvent the issues mentioned in Section 3, where we established that the multiple overheads implied by the need to exchange node neighbour lists, storing global information about the network in every node, computing the centrality value at every node, and waiting until all centrality values have converged would make these approaches entirely unfeasible for the IoT. Thanks to IoT-LAB, however, we have access to a controller that has perfect knowledge of the entire network, allowing us to offload the entire process to more powerful, centralised hardware. Of course, this would not be possible in a real deployment, but the following evaluation will show that even under these idealised circumstances, the proposed ABC strategy, which *is* fully distributed and implemented exclusively on the nodes themselves, can compete with the algorithms that offload their calculations.

5.2 Experiment Topologies

We perform our evaluations on two different network topologies: The *core* and *edge* topologies, as introduced in Section 2.

5.2.1 Core Topology. An experiment using the core topology begins with a brief (30 seconds) setup phase, during which every node advertises its own content prefix (dictated by its address), which is then propagated through the rest of the network using **HoPP**'s [24] routing algorithm. HoPP is primarily a publish-and-subscribe scheme for information-centric IoT, but also includes a prefix advertisement mechanism based on the **Trickle** [29] algorithm. The fact that the routing algorithm is based on Trickle also means that nodes' FIBs can be kept up to date during runtime.

The resultant network topology is a direct result of the FIB contents, which in turn are a direct result of the routing algorithm. In the HoPP/Trickle routing algorithm, prefix advertisements are propagated in a tree-like fashion. A producer will advertise its own prefixes with a rank of 0, which is then increased by each node that forwards the advertisement. CCN-lite's forwarding plane is configured in such a way that for any matching prefix, the FIB entry with the lowest rank is always preferred. This means that any multi-hop forwarding path will always minimise the number of hops to reach the producer. It also means that forwarding paths are more likely to converge closer to the producer, as the lowest-ranked nodes will be found there. This means that the resultant topology is a core topology.

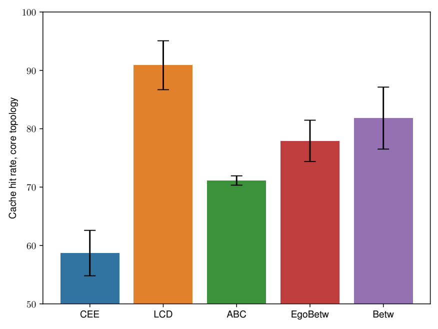
5.2.2 Edge Topology. In the setup phase of an experiment using the edge topology, each node advertises its own presence to its neighbours via broadcast. Nodes record every neighbour they can hear and pass this information on to the IoT-LAB control script. Because the control script has access to nodes' physical locations through the IoT-LAB API, it can use the nodes' neighbour information to construct an edge topology in which the FIB entries for each content prefix are connected in such a way that delivery paths run directly from the producer to the most distant consumers, with all other nodes connected to the most distant connected node they can hear. Thus, instead of connecting to the neighbour that is closest to the producer, nodes will tend to connect to the neighbour that is furthest toward the edge. This means that forwarding paths are more likely to converge at the edge, where the outermost nodes are found. Thus, the resultant topology resembles an edge topology.

5.3 Experiment Description

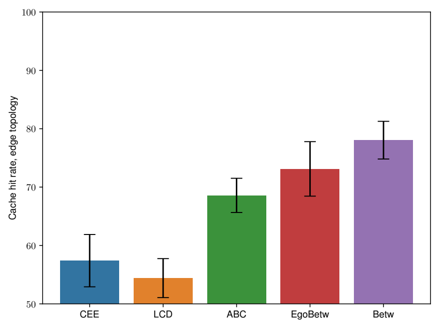
After topology setup is complete, every node will start requesting content chunks with random IDs in $\{0, \ldots, 49\}$ from each of the prefixes in its FIB (with 50 producers, there are thus 2500 distinct objects that can be requested) every 300 to 900 ms. Requested content IDs follow a uniformly random distribution to model the typical request distribution found in IoT applications [33, 38]. Interest and Data packets are handled as specified by the Named Data Networking (NDN) standard. The first time a node receives an Interest for a content chunk it owns, it produces that content chunk (the actual payload is irrelevant for our experiment) and sends it back towards the consumer. Caching of content chunks at intermediate nodes is dictated by the caching strategy selected for the experiment.

The control script takes periodic snapshots of cache contents and FIBs and logs statistics such as latency and hop counts. We use this information to evaluate the caching strategies in the rest of this section.

In addition to the proposed ABC strategy and the *Betw* and *EgoBetw* strategies introduced in Section 3, we also included two more strategies in our evaluation: Cache Everything Everywhere (CEE) [27] and Leave Copy Down (LCD) [28]. CEE is the default caching strategy assumed in most ICN implementations. We include it here to show how much there is to gain from employing a caching heuristic rather than simply caching all content. LCD



(a) Core topology



(b) Edge topology

Figure 3: Mean cache hit rate

works by copying a requested content object exactly one hop further every time a cache hit occurs, which means that it tends to keep content very close to the core, and only very popular content will be moved to the edge of the network. It is included in the evaluation to showcase a strategy that has very good performance in one topology type and very poor performance in another, as a contrast to the centrality-based strategies, whose performance is expected to be strong regardless of topology.

To showcase the effects of the topology on the caching strategies, we have split the visualisation of the results into the two topology types.

5.4 Cache Hit Rate

Although content delivery latency will be our main focus of analysis for this paper, a basic performance metric that cannot be overlooked is that of the *cache hit rate*. Cache hit rate describes the ratio of content objects that are retrieved as a cached copy from another node in the network as opposed to being retrieved from the original producer. In general, a higher cache hit rate is desirable, as it means that (i) content delivery times are reduced as content requests are being fulfilled without having to traverse the full path to the producer and (ii) strain on individual producers is reduced as the number of requests routed to them goes down, thus increasing battery life and reducing the probability of dropped packets due to saturated buffers.

Figure 3 shows the mean cache hit rate for the different strategies in the two topology types. As has been shown in previous literature [10, 36, 37], CEE's cache hit rate barely exceeds 50%, meaning that almost half of all content requests need to be routed to the original producer to be fulfilled. As an extreme contrast, LCD can reach a cache hit rate of over 90% in the core topology. In the edge topology, on the other hand, it performs even worse than CEE. This is to be expected: Since LCD keeps content close to the core by definition, it is much better suited to core topologies than it is to edge topologies.

The centrality-based strategies perform well across both topologies, only being outperformed by LCD in the core topology. It is evident that there is a clear link between the accuracy of the centrality measure and the performance of the caching strategy (since this is the only difference between the three strategies): The more information is available to the betweenness calculation, the better the estimate, which in turn results in better performance. However, we can also see that even with the rough centrality approximation provided by ABC, our cache hit rate is only about 10% worse than that of *Betw* and still significantly better than LCD in the edge topology and CEE in both topology types.

The centrality-based strategies perform slightly worse in the edge topology (by about 5% on average) compared to the core topology. The reason for this is that a core topology will generally have a larger number of central, well-connected nodes that make good candidates for caching data, whereas even well-connected nodes near the edge can only provide tangible benefits for their corner of the network. However, the centrality strategies still clearly outperform CEE in both topology types. The relative performance between the three strategies stays consistent across topology types.

5.5 Hop Count Reduction

For each Interest, the *distance to source* is the number of hops between its origin and the owner of the requested content prefix. Put simply, it is the number of hops that would be needed to deliver the content if there were no caches between the producer and the consumer. We compare this to the *hops to hit*, which is the actual number of hops taken by the Interest packet before a cache hit. The closer a cached copy exists to the consumer, the lower the *hops to hit*. The more effective a caching strategy is at storing content, the more content will be available closer to the consumer. The difference between the *distance to source* and the *hops to hit* is called the *hop count reduction*.

The mean hop count reduction for the different strategies is shown in Figs. 4a and 5a for the core and edge topologies respectively. For both topology types and all caching strategies, there is a measurable reduction in hops to hit that becomes more pronounced as the distance to source increases. In other words, the bigger the distance between the consumer and the content prefix owner, the more likely it is that the requested content will be found in a cache in an intermediate node, thus reducing the hops to hit.

As with the cache hit rate as shown in Section 5.4, we can see that the only strategy that is significantly affected by the topology type is LCD, which once again is the best-performing strategy in the core topology, but performs the poorest in the edge topology.

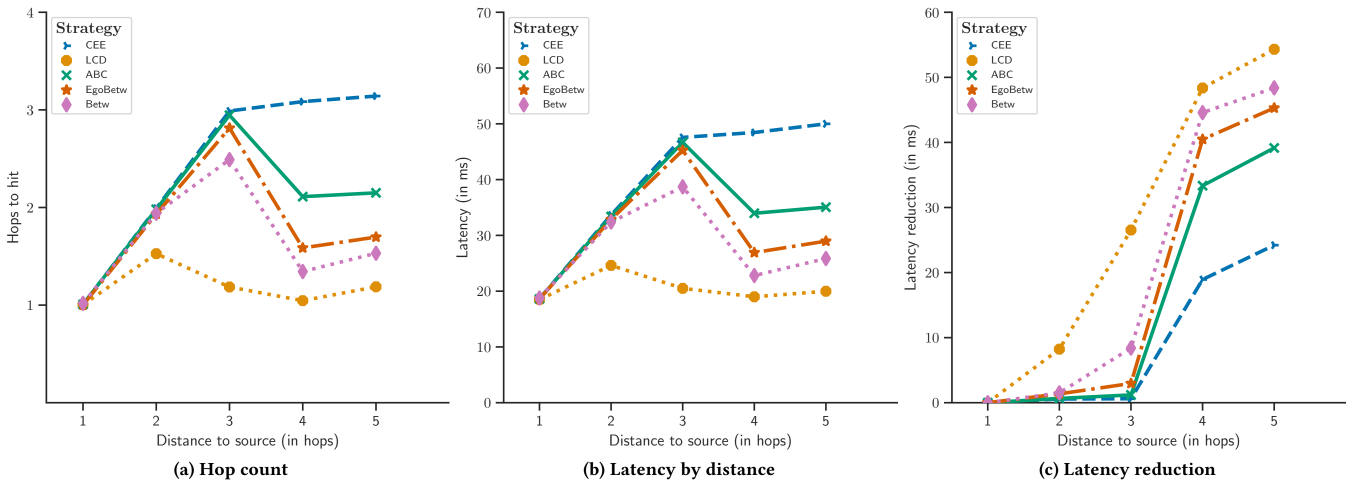


Figure 4: Performance comparison using core topology

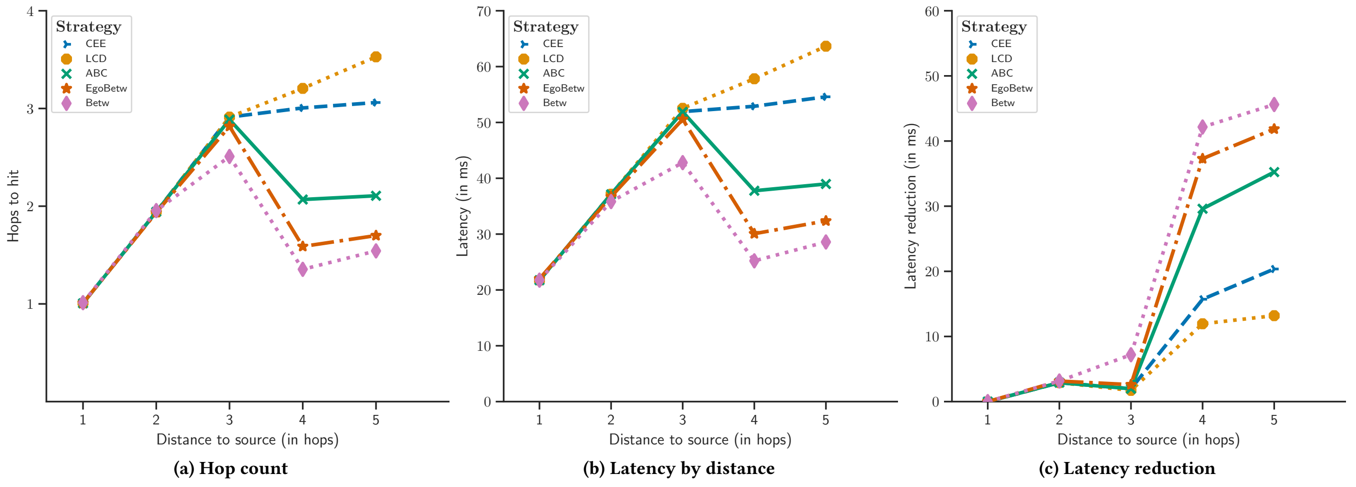


Figure 5: Performance comparison using edge topology

Clearly, LCD is a strategy that can be highly effective if employed in the topology it is designed for, but it can not be a universal solution to the caching problem.

CEE performs adequately; up to a distance to source of 3 hops, there is no significant hop count reduction, but at larger distances, the hops to hit even out, meaning that it can generally satisfy content requests within a reasonable number of hops. This shows the value of ICN caching even in its most basic state, as the hop count reduction for larger networks will still be noticeable.

The largest performance gains, however, are achieved by the centrality-based schemes. Interestingly enough, their hop count reduction is even stronger at a distance of 4 hops to the producer than it is at a distance of 3 hops. We attribute this observation to the fact that these strategies exploit centrality when deciding cache placement: At a longer distance to source, there are more potential caching nodes on the path to choose from, and thus a more optimal cache distribution can be reached. All centrality-based schemes follow this overall pattern, with variations in how much they actually reduce the hop count. We can see that in the core topology, the pattern followed by LCD resembles that of the centrality approaches — as the caching decisions reached by LCD in a core topology are very similar to those reached by centrality strategies — whereas in the edge topology, LCD's pattern more closely resembles that of CEE, as it is entirely divorced from this topology type.

Within the centrality-based schemes, we once again see that *Betw* does indeed boast the strongest performance, closely followed by *EgoBetw*, and ABC slightly behind the two. This follows our observations in Section 5.4 and is to be expected; after all, *Betw* has knowledge of the entire network topology when making its decision, and both *Betw* and *EgoBetw* can rely on intensive communications between neighbouring nodes to inform their strategy. It is encouraging, then, that ABC, which does not have global knowledge of the network and requires no additional communication between neighbours, can still achieve results that are comparable to its more complex relatives and outclass those of CEE and — in the edge topology — LCD.

5.6 Latency and Latency Reduction

Figures 4b and 5b show the mean content delivery latencies in relation to the distance to source. It is immediately obvious that the latencies for the different strategies and topology types follow the same pattern as the hop counts, which is intuitive as a reduction in hops to hit should result in a proportional reduction in latency. However, it is possible for a specific caching strategy to introduce additional delay through computational overhead, meaning that a direct correlation between hop count and latency is not guaranteed and that there may be latency differences between strategies that would not be evident from the hop count alone. Our results, however, show that there is no significant delay introduced by the different caching strategies. Of course, it needs to be noted here that as stated in Section 5.1, *Betw* and *EgoBetw* were implemented in such a way that the actual computation of the betweenness measure is offloaded to the IoT-LAB control script, which runs on conventional server hardware and thus has vastly more resources at its disposal than the IoT hardware. If these calculations were to run on the nodes themselves, it is possible that they might introduce a significant latency to the forwarding process. In order to compare the strategies in the most favourable terms, however, we decided to circumvent this possible source of latency.

The most interesting measure when considering novel caching heuristics is not the latency itself, but the *reduction* in latency, i.e. the expected gain in performance when employing the given caching strategy. We calculate the latency reduction by comparing the mean latency per distance to source of each strategy to the mean latency for that number of hops without caching (i.e. the expected latency without in-network caches). The result, as shown in Figs. 4c and 5c, is the *latency reduction* of each caching strategy. Following directly from the results in Figs. 4a, 4b, 5a and 5b, we can see that the biggest gains in performance can be seen at a distance to source of 3 to 4 hops. As shown in Section 5.5, LCD, being uniquely suited for core topologies, already exhibits significant latency reduction at the lowest hop counts and maintains the strongest reduction overall in the core topology, whereas in the edge topology, it performs the same as the other strategies at lower hop counts and is quickly overtaken by them as hop count grows, achieving only minimal improvements over latencies without caching. The other strategies all follow roughly the same pattern, with improvements in latency being slightly smaller overall in the edge topology compared to the core topology. The hierarchy between the centrality-based strategies is consistent, but the difference in latency reduction between the lightweight ABC and the complex *Betw* is in the range of 10 *ms*, a very encouraging result.

5.7 Memory Use of ABC

In Section 4.2 we mentioned that the memory required by nodes in ABC to store the paths they are on depends on their centrality. In fact, the number of paths stored in a node is exactly equal to its centrality. In our experiments, we found that there were only a few nodes with high centrality values (core nodes in the core topology and edge nodes in the edge topology), with the upper bound being around 20 on average for the core topology and 5 on average for the edge topology, while non-central nodes averaged below 5.

5.8 Convergence Time of ABC

As mentioned in Section 4, ABC eliminates the need for *a priori* topology knowledge by approximating nodes' centralities during runtime. However, this also means that there is a period of time after ABC is initialised during which the centrality values will not be correct. In fact, since all nodes have an initial centrality of 0, ABC will perform identically to CEE until the node centrality values start to diverge. It is important to examine how long it takes for the node centrality values to reach a suitable level of differentiation, and also how long it takes for the values to converge to a sufficient approximation of the "real" centrality values that would be calculated by *Betw*.

In our experiments, it only took an average of between 3 and 5 Interests from each node to form a rough distinction in node centralities, such that there were only one or two nodes with the highest centrality value on any given path. On average, it took approximately 50 *s* for all nodes in the network to reach a sufficient approximation C_{app} of their actual centrality value C_{real} such that $|C_{real} - C_{app}| \leq 2$.

5.9 Summary of Results

Upon first inspection, the results shown here may not seem particularly compelling as ABC is never able to outperform the existing centrality strategies. However, the actual, tangible advantage of ABC lies in the fact that its complexity, as shown in Section 4.2, is significantly lower than that of *Betw/EgoBetw*. This means that in contrast to those strategies, it is actually a viable candidate for implementation on constrained devices, and the fact that its results are not significantly worse than the more complex strategies as well as being consistent across different topology types provide a strong motivation for its use.

6 RELATED WORK

The ICN caching strategies that are most relevant to this study — i.e., those based on the centrality measure — have already been introduced in Section 3. The following section will give a brief overview of a number of other caching strategies that have been developed for ICN, as well as discuss previous comparative studies.

6.1 Caching Strategies

Since Cache Everything Everywhere (CEE), the default caching strategy that Content-Centric Networking (CCN) was originally intended to employ, was found to be suboptimal [10, 36, 37], a number of alternative strategies have been proposed. Since there are various performance metrics that can be improved by employing different caching strategies, these strategies are very diverse, and there is no one strategy that is clearly superior to all others in all aspects of performance.

If the objective of the caching heuristic is to improve cache diversity in order to be able to store more content copies across the network, the family of *probabilistic* caching strategies may present solutions. The strategies within this family range from simple, static-probability strategies [4, 26, 41, 50] to dynamic strategies that compute a caching probability based on factors such as path length [36], content freshness [18], content popularity [16] battery level, cache occupancy, neighbouring cache contents [49], or a combination of some or all of these factors [25]. However, none of these policies take the topology of the network into account.

We have already introduced a number of policies that utilise topology in Section 3. Other approaches make use of the concept of topology potential [40, 51], which describes the effect that a node exerts on other nodes in the network, which like gravity is affected by its distance to other nodes. However, like the centrality approaches, they suffer from their global knowledge requirements and their inflexibility in the face of dynamic topologies.

The middle ground between purely local strategies and those that require global knowledge is occupied by the family of *cooperative caching* strategies. This cooperation can be *explicit* [32], requiring some communication overhead between neighbouring nodes, or *implicit* [30, 47], using information inherent to the nodes or the content to achieve cooperation. It could be argued that centrality strategies use a form of implicit cooperation due to the fact that the caching decision is based on information provided with the content. As will be elaborated further in Section 7, it may be feasible to extend ABC with an explicit cooperation component to enable off-path caching.

Providing some valuable context for our work, Wang *et al.* [44] performed an in-depth analysis of the interactions between network topology and caching metrics. They show that caching policies are at their most optimal when content popularity and topology (i.e., node betweenness), are strongly correlated. They also provide a categorisation for caching policies according to where content is cached (see Section 2).

6.2 Comparative Studies

In the domain of traditional ICN, there are a number of surveys [21, 48, 50] and comparative studies [41, 50] on caching strategies. Studies that focus specifically on the IoT, however, are not as numerous.

In traditional ICN, the tangible benefits of pervasive caching have been called into question in favour of simple edge caching [22]. However, IoT differs significantly from the traditional Internet both in topology as well as traffic patterns (uniform rather than Zipfian), so these findings are not applicable to the scenarios we studied.

Arshad *et al.* [4, 5] have contributed a comprehensive overview of ICN caching schemes for the IoT, but provide no experimental evaluation. The first experimental studies in this area were conducted by Hail *et al.* [26] and Meddeb *et al.* [33], who used simulated hardware for their evaluations. The first study to use physical hardware in a realistic environment was conducted by us [34].

Content delivery latency as a performance metric has been explored in most of the studies named above and was the focus of several publications by Carofiglio *et al.* [11–14]. However, these solutions are once again not intended to address IoT-specific challenges. The question of ICN caching latency in the IoT was studied in detail in our previous work [35].

7 CONCLUSIONS AND FUTURE WORK

We have presented ABC, a simple lightweight caching scheme for information-centric IoT that uses approximate centrality information to cache data in the most convenient location regardless of topology. We have demonstrated that while this approach does not outperform existing strategies that make use of more precise centrality measures, it can provide similar reductions in content delivery latency without requiring any setup, global knowledge, or communications overhead.

We have shown that if the network topology is well-known, a caching strategy specifically designed for that topology may be the optimal choice for reducing content delivery latency, whereas centrality-based caching strategies can achieve strong results in both edge cases examined here, making them a strong choice if the topology is unknown or mutable.

One potential issue that was not addressed in the main text relates to the fact that caching strategies in the centrality-based family, including ABC, inherently place a higher strain on certain, well-connected nodes, because those are the nodes with the highest centrality values and thus the likeliest candidates for caching. This can potentially cause problems, as well-connected nodes may already have to contend with above-average load due to the very fact that they are more central, meaning that more traffic is routed through them compared to edge nodes. Choosing them as the preferred caching locations on top of this may exacerbate this effect, potentially leading to dropped packets as buffers become saturated or, in the worst case, node failure as batteries drain faster than those of less-taxed neighbour nodes. How much of an obstacle this presents in reality will depend on application- and deployment-specific factors, such as the traffic rate and whether the nodes have access to a constant power source or easily replaceable batteries. Although we did not see immediate evidence for such performance degradation in our experiments so far, a more detailed study of the relative load placed on different nodes when using centrality-based caching may shed more light on this issue and perhaps suggest reasonable upper bounds for how much central nodes should be preferred before the risks outweigh the benefits.

There is an entire class of ICN caching strategies that could benefit from the findings in this study: the off-path caching strategies [17, 19, 39, 42]. Off-path caching describes a general caching philosophy in which content can not only be stored at nodes on its delivery path, but also offloaded onto neighbouring nodes to achieve more optimal content distribution. It is conceivable that combining this approach with the centrality metrics employed here, we could achieve even better caching performance by offloading content onto off-path nodes with stronger centrality. Similarly, the disproportionate load problem described above may be mitigated by using off-path caching to distribute load more evenly.

In addition to addressing the above issues, future work will aim to extend the generalisability of the findings presented in this paper. We will evaluate ABC's performance under more varied scenarios, particularly using traffic patterns and topologies from real-world IoT applications. Furthermore, we plan to run experiments using larger cache sizes in order to examine how sensitive ABC's benefits are to external limitations.

REFERENCES

[1] S. S. Adhatarao, M. Arumaithurai, D. Kutscher, and X. Fu. 2018. ISI: Integrate Sensor Networks to Internet With ICN. *IEEE Internet of Things Journal* 5, 2 (April 2018), 491–499. https://doi.org/10.1109/JIOT.2017.2741923

[2] Cédric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frédéric Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and others. 2015. FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed. In *Proceedings of the 2nd IEEE World Forum on Internet of Things (WF-IoT)*. Milan, Italy.

[3] Marica Amadeo, Claudia Campolo, Jose Quevedo, Daniel Corujo, Antonella Molinaro, Antonio Iera, Rui L. Aguiar, and Athanasios V. Vasilakos. 2016. Information-Centric Networking for the Internet of Things: Challenges and Opportunities. *IEEE Network* 30, 2 (2016), 92–100.

[4] Sobia Arshad, Muhammad Awais Azam, Mubashir Husain Rehmani, and Jonathan Loo. 2017. Information-Centric Networking Based Caching and Naming Schemes for Internet of Things: A Survey and Future Research Directions. *arXiv preprint arXiv:1710.03473* (2017).

[5] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo. 2018. Recent Advances in Information-Centric Networking based Internet of Things (ICN-IoT). *IEEE Internet of Things Journal* (2018), 1–1. https://doi.org/10.1109/JIOT.2018.2873343

[6] Atmel. 2009. AT86RF231 Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications. http://www.atmel.com/images/doc8111.pdf

[7] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch. 2018. RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. *IEEE Internet of Things Journal* 5, 6 (Dec. 2018), 4428–4440. https://doi.org/10.1109/JIOT.2018.2815038

[8] Emmanuel Baccelli, Christian Mehlis, Oliver Hahm, Thomas C. Schmidt, and Matthias Wählisch. 2014. Information Centric Networking in the IoT: Experiments with NDN in the Wild. In *Proceedings of the 1st International Conference on Information-Centric Networking (ICN)*. Paris, France, 77–86. http://dl.acm.org/citation.cfm?id=2660144

[9] Carsten Bormann, Mehmet Ersue, and Ari Keranen. 2014. *Terminology for Constrained-Node Networks*. Technical Report.

[10] Giovanna Carofiglio, Vinicius Gehlen, and Diego Perino. 2011. Experimental Evaluation of Memory Management in Content-Centric Networking. In *Proceedings of the IEEE International Conference on Communications (ICC)*. Kyoto, Japan, 1–6.

[11] G. Carofiglio, L. Mekinda, and L. Muscariello. 2015. FOCAL: Forwarding and Caching with Latency Awareness in Information-Centric Networking. In *2015 IEEE Globecom Workshops (GC Wkshps)*. 1–7. https://doi.org/10.1109/GLOCOMW.2015.7413972

[12] G. Carofiglio, L. Mekinda, and L. Muscariello. 2015. LAC: Introducing Latency-Aware Caching in Information-Centric Networks. In *Proceedings of the IEEE 40th Conference on Local Computer Networks (LCN)*. 422–425. https://doi.org/10.1109/LCN.2015.7366343

[13] Giovanna Carofiglio, Leonce Mekinda, and Luca Muscariello. 2016. Analysis of Latency-Aware Caching Strategies in Information-Centric Networking. In *Proceedings of the 1st Workshop on Content Caching and Delivery in Wireless Networks (CCDWN '16)*. ACM, New York, NY, USA, 5:1–5:7. https://doi.org/10.1145/2836183.2836188 event-place: Heidelberg, Germany.

[14] Giovanna Carofiglio, Léonce Mekinda, and Luca Muscariello. 2016. Joint Forwarding and Caching with Latency Awareness in Information-Centric Networking. *Computer Networks* 110 (Dec. 2016), 133–153. https://doi.org/10.1016/j.comnet.2016.09.019

[15] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. 2013. Cache "Less for More" in Information-Centric Networks (Extended Version). *Computer Communications* 36, 7 (2013), 758–770.

[16] Bo Chen, Liang Liu, Zhao Zhang, Wenbo Yang, and Huadong Ma. 2016. BRR-CVR: A Collaborative Caching Strategy for Information-Centric Wireless Sensor Networks. In *12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*. IEEE, 31–38.

[17] Raffaele Chiocchetti, Dario Rossi, Giuseppe Rossini, Giovanna Carofiglio, and Diego Perino. 2012. Exploit the Known or Explore the Unknown?: Hamlet-Like Doubts in ICN. In *Proceedings of the Second Edition of the ICN Workshop on Information-Centric Networking*. Helsinki, Finland, 7–12.

[18] Dong Doan Van and Qingsong Ai. 2018. An Efficient in-Network Caching Decision Algorithm for Internet of Things. *International Journal of Communication Systems* 31, 8 (2018), e3521.

[19] Martin Dräxler and Holger Karl. 2012. Efficiency of on-Path and Off-Path Caching Strategies in Information Centric Networks. In *Proceedings of the IEEE International Conference on Green Computing and Communications (GreenCom)*. Besancon, France, 581–587.

[20] Martin Everett and Stephen P. Borgatti. 2005. Ego Network Betweenness. *Social Networks* 27, 1 (2005), 31–38.

[21] Chao Fang, F. Richard Yu, Tao Huang, Jiang Liu, and Yunjie Liu. 2014. A Survey of Energy-Efficient Caching in Information-Centric Networking. *IEEE Communications Magazine* 52, 11 (2014), 122–129.

[22] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, K. C. Ng, Vyas Sekar, and Scott Shenker. 2013. Less Pain, Most of the Gain: Incrementally Deployable ICN. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 147–158.

[23] Cenk Gündoğan, Peter Kietzmann, Martine Lenders, Hauke Petersen, Thomas C. Schmidt, and Matthias Wählisch. 2018. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In *Proceedings of the 5th ACM Conference on Information-Centric Networking (ICN '18)*. ACM, New York, NY, USA, 159–171. https://doi.org/10.1145/3267955.3267967 Boston, Massachusetts.

[24] C. Gündoğan, P. Kietzmann, T. C. Schmidt, and M. Wählisch. 2018. HoPP: Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things. In *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. 331–334. https://doi.org/10.1109/LCN.2018.8638030

[25] Mohamed Ahmed Hail, Marica Amadeo, Antonella Molinaro, and Stefan Fischer. 2015. Caching in Named Data Networking for the Wireless Internet of Things. In *Proceedings of the International Conference on Recent Advances in Internet of Things (RIoT)*. Singapore, 1–6.

[26] Mohamed Ahmed M. Hail, Marica Amadeo, Antonella Molinaro, and Stefan Fischer. 2015. On the Performance of Caching and Forwarding in Information-Centric Networking for the IoT. In *Proceedings of the International Conference on Wired/Wireless Internet Communication (WWIC)*. Springer, Malaga, Spain, 313–326.

[27] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. Rome, Italy, 1–12. http://dl.acm.org/citation.cfm?id=1658941

[28] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. 2006. The LCD Interconnection of LRU Caches and its Analysis. *Performance Evaluation* 63, 7 (July 2006), 609–634. https://doi.org/10.1016/j.peva.2005.05.003

[29] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. 2011. *The Trickle Algorithm*. RFC 6206. RFC Editor. http://www.rfc-editor.org/rfc/rfc6206.txt http://www.rfc-editor.org/rfc/rfc6206.txt.

[30] Zhe Li and Gwendal Simon. 2011. Time-Shifted TV in Content Centric Networks: The Case for Cooperative In-Network Caching. In *Proceedings of the IEEE International Conference on Communications (ICC)*. Kyoto, Japan, 1–6.

[31] B. Liu, T. Jiang, Z. Wang, and Y. Cao. 2017. Object-Oriented Network: A Named-Data Architecture Toward the Future Internet. *IEEE Internet of Things Journal* 4, 4 (Aug. 2017), 957–967. https://doi.org/10.1109/JIOT.2017.2714658

[32] Yinlong Liu, Dali Zhu, and Wei Ma. 2016. A Novel Cooperative Caching Scheme for Content Centric Mobile Ad Hoc Networks. In *IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 824–829.

[33] Maroua Meddeb, Amine Dhraief, Abdelfettah Belghith, Thierry Monteil, and Khalil Drira. 2017. How to Cache in ICN-Based IoT Environments?. In *IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 1117–1124.

[34] Jakob Pfender, Alvin Valera, and Winston K. G. Seah. 2018. Performance Comparison of Caching Strategies for Information-Centric IoT. In *Proceedings of the 5th ACM Conference on Information-Centric Networking (ICN '18)*. ACM, New York, NY, USA, 43–53. https://doi.org/10.1145/3267955.3267966 Boston, Massachusetts.

[35] Jakob Pfender, Alvin Valera, and Winston K. G. Seah. 2019. Content Delivery Latency of Caching Strategies for Information-Centric IoT. *arXiv:1905.01011 [cs]* (May 2019). http://arxiv.org/abs/1905.01011 arXiv: 1905.01011.

[36] Ioannis Psaras, Wei Koong Chai, and George Pavlou. 2012. Probabilistic In-Network Caching for Information-Centric Networks. In *Proceedings of the Second Edition of the ICN Workshop on Information-Centric Networking*. Helsinki, Finland, 55–60.

[37] Ioannis Psaras, Richard G. Clegg, Raul Landa, Wei Koong Chai, and George Pavlou. 2011. Modelling and Evaluation of CCN-caching Trees. In *Proceedings of the 10th International IFIP TC 6 Conference on Networking (NETWORKING)*. Valencia, Spain, 78–91. http://dl.acm.org/citation.cfm?id=2008780.2008789

[38] Akhila Rao, Olov Schelén, and Anders Lindgren. 2016. Performance Implications for IoT Over Information Centric Networks. In *Proceedings of the Eleventh ACM Workshop on Challenged Networks*. ACM, 57–62.

[39] Sumanta Saha, Andrey Lukyanenko, and Antti Ylä-Jääski. 2013. Cooperative Caching Through Routing Control in Information-Centric Networks. In *Proceedings of the IEEE INFOCOM*. Turin, Italy, 100–104.

[40] Yong Sun, Tiankui Zhang, Ruoqi Wang, Weidong Feng, and Pu Chen. 2016. Topology Potential Based Probability Caching Strategy for Content Centric Ad Hoc Networks. *Journal of Residuals Science & Technology* 13, 6 (2016).

[41] Saran Tarnoi, Kalika Suksomboon, Wuttipong Kumwilaisak, and Yusheng Ji. 2014. Performance of Probabilistic Caching and Cache Replacement Policies for Content-Centric Networks. In *Proceedings of the IEEE 39th Conference on Local Computer Networks (LCN)*. Edmonton, AB, Canada, 99–106.

[42] Xenofon Vasilakos, Vasilios A. Siris, George C. Polyzos, and Marios Pomonis. 2012. Proactive Selective Neighbor Caching for Enhancing Mobility Support in Information-Centric Networks. In *Proceedings of the Second Edition of the Icn Workshop on Information-Centric Networking*. Helsinki, Finland, 61–66.

[43] Huijuan Wang, Javier Martin Hernandez, and Piet Van Mieghem. 2008. Betweenness Centrality in a Weighted Network. *Physical Review E* 77, 4 (2008), 046105.

[44] L. Wang, S. Bayhan, and J. Kangasharju. 2014. Effects of Cooperation Policy and Network Topology on Performance of In-Network Caching. *IEEE Communications Letters* 18, 4 (April 2014), 680–683. https://doi.org/10.1109/LCOMM.2014.013114.132647

[45] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis: Methods and Applications*. Vol. 8. Cambridge University Press.

[46] Z. Yan, S. Zeadally, and Y. Park. 2014. A Novel Vehicular Information Network Architecture Based on Named Data Networking (NDN). *IEEE Internet of Things Journal* 1, 6 (Dec. 2014), 525–532. https://doi.org/10.1109/JIOT.2014.2354294

[47] Yuguang Zeng and Xiaoyan Hong. 2011. A Caching Strategy in Mobile Ad Hoc Named Data Network. In *6th International ICST Conference on Communications and Networking in China (CHINACOM)*. IEEE, 805–809.

[48] Guoqiang Zhang, Yang Li, and Tao Lin. 2013. Caching in Information Centric Networking: A Survey. *Computer Networks* 57, 16 (2013), 3128–3141.

[49] Li Zhang, Jiayan Zhao, and Zhenlian Shi. 2015. LF: A Caching Strategy for Named Data Mobile Ad Hoc Networks. In *Proceedings of the 4th International Conference on Computer Engineering and Networks*. Springer, 279–290.

[50] Meng Zhang, Hongbin Luo, and Hongke Zhang. 2015. A Survey of Caching Mechanisms in Information-Centric Networking. *IEEE Communications Surveys & Tutorials* 17, 3 (2015), 1473–1499.

[51] Le Zhou, Tiankui Zhang, Xiaogeng Xu, Zhimin Zeng, and Yinlong Liu. 2015. Broadcasting Based Neighborhood Cooperative Caching for Content Centric Ad Hoc Networks. In *IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE, 1–5.