Lightweight Caching and Load Balancing for Efficient Content Delivery in Information-Centric IoT

by

Jakob Pfender

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Engineering.

Victoria University of Wellington
2020

Abstract

In recent years, Information-Centric Networking (ICN) has emerged as a promising candidate for a future Internet architecture. While originally designed with the traditional Internet in mind, it has also been identified as a potential replacement for current Internet of Things (IoT) networking solutions. However, applications in the IoT face a number of unique challenges due to the constrained nature of the hardware. One of these challenges is that available memory is often extremely limited.

This thesis aims to evaluate the feasibility of using ICN in-network caching on IoT devices in order to achieve efficient content delivery. It evaluates the performance of existing approaches on constrained hardware and explores how the technology can be improved and tailored towards that environment. Existing strategies are found to be lacking in key aspects, particularly the fact that the effects of network topology are often not considered when making caching decisions. It is shown that approaches based on network centrality are promising, but existing implementations are not suited for constrained hardware. Therefore, a lightweight in-network caching strategy called Approximate Betweenness Centrality (ABC) is proposed that takes the specific requirements of IoT into consideration and allows for efficient cache placement regardless of network topology. Then, a modular solution for load balancing through off-path caching is presented to address potential shortcomings of the centrality-based caching approach. It allows the network to make more efficient use of available caching resources without introducing additional overhead. Furthermore, solutions for ensuring Quality of Service (QoS) are discussed. The expanded role of caching strategies under such QoS constraints is explored and their performance is evaluated.

This thesis shows that it is possible to design and deploy lightweight, low-overhead solutions on constrained hardware. Using a realistic deployment of physical IoT devices, it is demonstrated that these approaches can reach satisfactory levels of performance.

Acknowledgments

This thesis would not exist without the indispensable support of my supervisors, Prof. Winston Seah and Dr. Alvin Valera. Their expert guidance during every step of my journey has been invaluable. I also wish to thank my colleagues Cenk Gündoğan, Thomas Schmidt, and Matthias Wählisch, for expanding my scientific horizons and enabling fruitful collaboration. Particularly helpful to me over the last years were my fellow students Aleja, Deepak, Duncan, Hang, Liang, and Muru, who always provided enlightening discussion and feedback. I would like to recognise the valuable assistance I received from ECS's incredibly supportive staff, particularly Diana, Monoa, and Siyun. Finally, I wish to thank my parents for their support and encouragement throughout my study, and Victoria, for all her love and patience.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ABC	Approximate Betweenness Centrality
ACL	Access Control List
AM	Authorisation Manager
AS	Autonomous System
CCN	Content-Centric Networking
CDN	Content Delivery Network
CEE	Cache Everything Everywhere
CRR	Cache Retention Ratio
CS	Content Store
DD	Directed Diffusion
DM	Diversity Metric
DODAG	Destination Oriented Directed Acyclic Graph
FIB	Forwarding Information Base
FIFO	First In First Out
HVAC	Heating, Ventilation, and Air Conditioning
ICN	Information-Centric Networking
IoT	Internet of Things
IRTF	Internet Research Task Force
ISP	Internet Service Provider
LCC	Lifetime-Based Cooperative Caching
LCD	Leave Copy Down

LCE Leave Copy Everywhere
LFU Least Frequently Used
LRU Least Recently Used
LSR Link State Routing
MCD Move Copy Down
MDMR Max Diversity Most Recent
MTU Maximum Transmission Unit
NCP Near-ICN Cache Placement
NDN Named Data Networking
NDNoT Named Data Network of Things
NFaaS Named Function as a Service
NFN Named Function Networking
NHT Next Hop Table
NSF National Science Foundation
OPC Off-Path Caching
OSPF Open Shortest Path First
PCS Periodic Caching Strategy
PIT Pending Interest Table
QoS Quality of Service
RAM Random-Access Memory
RFI Remote Function Invocation
RMI Remote Method Invocation
RNG Random Number Generation
ROM Read-Only Memory
RPC Remote Procedure Call
RR Random Replacement

SDN Software Defined Networking
SNC Selective Neighbour Caching

TO	Targeted Offloading
TSB	Time Since Birth
TSI	Time Since Inception
VANET	Vehicular Ad-Hoc Network
WSN	Wireless Sensor Network

CHAPTER 1

Introduction

The **Internet of Things (IoT)** and its associated technologies are an active research field with many unanswered questions. By its nature, the IoT connects many heterogeneous "things" together, all of which have vastly different purposes and requirements as well as different processing capabilities, energy sources, and memory capacities. A major concern in IoT research is to find the most effective way to utilise the heterogeneous hardware in order to achieve the goals of a deployment.

IoT deployments can have a wide range of possible goals. They may be gathering data for the purpose of environmental, industrial, or structural monitoring; they may also have a tangible effect on their environment by acting as actuators controlling systems such as Heating, Ventilation, and Air Conditioning (HVAC) or security infrastructure. Furthermore, in any given IoT deployment, the devices themselves may differ significantly in terms of resources and capabilities. Devices that fall under the IoT banner include mobile phones, tablets, and other "smart" personal devices, components of building automation and smart-home systems, as well as a wide range of sensors and actuators. As of 2019, it is estimated [93] that the number of connected IoT devices will exceed 40 billion by 2025, with almost 80 zettabytes of data generated.

Table 1.1.: Classes of constrained devices [29]

Class	ROM	RAM
0	$<<$ 100 kB	$<<$ 10 kB
1	$\sim$ 100 kB	$\sim$ 10 kB
2	$\sim$ 250 kB	$\sim$ 50 kB

The unifying features of the IoT devices discussed in the context of this thesis are the ability to communicate wirelessly (typically using IEEE 802.15.4 [1]) and a small form factor. In order to be able to scale networks up to include hundreds or even thousands of individual nodes, the devices themselves are necessarily severely constrained so as to be economically and physically viable. Constraints typically take the form of limited processing power and memory as well as running on batteries that can be difficult to replace, especially in outdoor environments. These physical constraints in turn lead to constraints on the software running on the device. Limited Read-Only Memory (ROM) implies limited code complexity, limited Random-Access Memory (RAM) implies limited program state/memory, limited processing power implies a limit on computational complexity, and limited battery life implies a need for energy-efficient solutions.

To quantify these constraints, some classification is needed. Bormann *et al.* [29] divide constrained devices into three classes according to their ROM and RAM sizes, which are reproduced in Table 1.1. This thesis is primarily concerned with class 2 devices, which are most common in typical environmental, agricultural, and industrial monitoring applications. Their comparatively large memory means that they are generally capable of supporting the same protocol stacks (e.g. TCP/IP) found on larger devices, albeit sometimes as compressed variants (e.g. 6LoWPAN [82]). Any constraints considered in the remainder of this thesis become even more significant if applied to lower class devices.

The unique limitations of IoT devices have inspired dedicated research communities to work on solutions. The limited processing power of single devices is nowadays mitigated by using distributed solutions to complex problems and having neighbouring nodes cooperate with one another. Short battery lifetimes are counteracted by placing a focus on energy efficiency as well as more advanced solutions such as energy harvesting [128], whereby devices are enabled to capture and store energy from various external sources. Finally, the problem of limited memory is addressed by optimising cache placement within the network to grant all network participants efficient ways to store and retrieve relevant data, as well as introducing filtering mechanisms to reduce the total amount of data being transferred from node to node.

Another challenge in the field of IoT is that of reliable and efficient content delivery. Since most IoT communication uses wireless communication channels, delivery is not always guaranteed. Especially in time-critical applications where unlimited retransmissions are not an option, this can pose serious problems. Compounding this issue is the fact that there is not much that can be done on the software side to improve the reliability of individual transmissions — wireless communication is inherently unreliable, and any application that relies on it needs to be able to work around this fact. However, even if individual transmissions cannot be made more reliable, the robustness of the overall data delivery can still be increased by decreasing the impact of failed transmissions and dropped packets. One prominent approach to this is the use of in-network caching. As content traverses a wireless multi-hop path from sender to receiver, intermediate nodes store a copy of the data, so that in the case of a dropped transmission, it can be retransmitted from the last node that successfully received it rather than re-initiating the entire transfer from the original sender. This also increases the efficiency of content delivery, as access times are reduced for content that is cached closer to the nodes who request it.

Although promising, in-network caching does come with an important caveat. As described above, the context of IoT generally implies devices with severely constrained memory, which limits the amount of data that can be cached by individual devices. Therefore, any solution that implements in-network caching for constrained devices needs to prioritise being lightweight.

In-network caching is an essential part of Information-Centric Networking (ICN), a completely new networking paradigm that has been gaining traction in the networking community [157]. Although not developed specifically for the IoT, its data-centric nature makes it a promising candidate for IoT applications [120]. However, as introduced above, the question remains whether the benefits offered by ICN caching strategies can be achieved in the constrained context of IoT.

Ensuring efficient content delivery in IoT devices in a lightweight fashion is clearly a major challenge. While performance is affected by all of the factors — unstable links, limited processing power, finite battery life — described above, this thesis will specifically concentrate on the impact of constrained memory. This leads to an exploration of the new networking paradigms of ICN, specifically the benefits derived from their caching, for their applicability in constrained-memory environments.

1.1 Motivation

In the future envisioned by the IoT paradigm, billions of objects will be interconnected in very large networks and seamlessly embedded in the surrounding environment, creating a completely new family of applications and services. The fundamental building blocks of this vision are smart objects with the ability to sense physical phenomena and trigger actions that have tangible effects in the physical world.

IoT communications are increasingly focused around the retrieval of large amounts of actuation, measurement, or monitoring information from large numbers of sources, while the actual, physical location of these data sources is less important. In other words, traffic patterns in a typical IoT application will take the form of consumers requesting a certain piece of data (such as the current occupancy of a room or the average humidity in a forest prone to wildfires) without knowing or caring which specific device will provide this information. Furthermore, in complex monitoring scenarios, the same data may be requested by multiple consumers at the same time.

In traditional IoT networking, this would mean that large amounts of redundant data would clog up the network and cause congestion problems. However, the shift in traffic patterns from connection-centric to content-centric communications has led to novel ways of thinking about networking, most prominent among them the family of networking paradigms known collectively as Information-Centric Networking (ICN). A notable advantage of these new approaches, apart from improved support for dynamic network topologies, is the ability to provide ubiquitous and transparent in-network caching of data.

In-network caching is attractive for IoT, since it not only reduces latency in accessing content, but can also reduce network traffic, balance load, improve resource utilisation, and increase the stability of the network as a whole. Because of these benefits, caching in IoT has become an active area of research, especially as part of ICN approaches [14, 66, 68, 69, 88, 94, 120, 121, 141, 166]. In addition to its built-in caching, ICN also has the potential to improve network flexibility and adaptability thanks to its native support for mobility and multicast.

However, as opposed to the traditional Internet, IoT data are typically small in size, transient, and frequently superseded by newer versions. This means that caching decisions must be based on dynamic factors such as application requirements and the time ranges for which the content is projected to remain relevant. At the same time, unlike routers in the traditional Internet, IoT devices are often resource-constrained, with limitations in terms of processing capabilities, memory, and energy. For these reasons, caching algorithms designed for intransient Internet traffic are generally regarded as ill-suited for IoT applications [68, 69, 121, 141].

Interestingly, existing research [21] suggests that even when caching is completely disabled, IoT applications can benefit from ICN, simply because the network stack provided by popular implementations such as CCN-Lite is significantly slimmer than that of e.g. 6LoWPAN / IPv6 / RPL, which is currently the de-facto standard in most IoT applications [175–177]. Nevertheless, it would be desirable to have the ability to utilise a full-fledged ICN implementation in the IoT, including caching, in order to use the technology to its fullest potential.

1.2 Problem Statement

Distributed IoT applications often produce large amounts of content, a lot of it redundant. If there is so much traffic that the network as a whole begins to suffer, e.g. from congestion, memory constraints, or exhaustion of processing power, it is necessary to find a solution that effectively reduces the amount of traffic generated by the application without reducing the quality of the results.

Traditionally, this problem has been approached using techniques such as duplicate data detection and filtering, often through explicit detection approaches such as classifiers. However, these solutions often come with their own computation and communication overheads and it can be difficult to balance the benefits and drawbacks of complex filtering approaches. An ideal solution would be one that performs data aggregation inherently as one of its core communication primitives. This leads us naturally to the new paradigm of ICN, which solves the filtering problem by making each piece of data idempotent, meaning that data objects have value beyond point-to-point connections and can be easily reused without any need for duplication (the theory behind this is discussed in more detail in Section 2.1).

If ICN paradigms are to be brought to the IoT, however, there are unique challenges that need to be addressed:

- As noted in Section 1.1, IoT contents are on the whole rather different from "normal" Internet contents. While modern Internet traffic is more and more being dominated by large media files, IoT content is generally small and transient. This means that in addition to considering the differences in hardware, caching strategies for information-centric IoT should ideally also take into account differences in the content they handle.
- IoT deployments have a wide range of possible topologies. Topology and caching are closely intertwined; the topology has a direct influence on which nodes in the network are the best candidates for caching content, as it determines where in the network content flows are most likely to intersect. Some caching strategies tend to keep content close to the network's core, while others prefer to push it out towards the edge. A caching strategy that does one or the other may be extremely efficient in one topology

type and utterly useless in another. Furthermore, if the network topology is dynamic, the ideal caching location may change over time. Therefore, an optimal caching strategy should aspire to work equally well in all topology types and have the ability to adapt to dynamic topologies.

- Selecting specific nodes to cache on, particularly if dictated by topology, can result in a load balancing problem, where certain nodes end up caching a much larger proportion of the network's content, making them more important, which means that more requests will be routed towards them, which in turn may overtax their resources, leading to a decrease in overall performance. To counter this, it would be desirable to make caching more intelligent and flexible, specifically by enabling caching nodes to offload some of their cached content to their neighbours in order to more evenly balance caching load in the network.

In short, the idiosyncrasies of IoT and their interaction with ICN approaches need to be fully explored and understood before viable approaches can be found. Any proposed solution will have to take into account the unique traffic shapes and topology effects found in the IoT while being lightweight and able to be deployed on severely limited hardware.

1.3 Research Questions

The primary aim of this thesis is to find ways to ensure lightweight and efficient content delivery in IoT environments. This overarching goal will be approached by addressing the following research questions:

1. How to bring the full benefits of Information-Centric Networking (ICN), including its caching mechanism, to the resource-constrained environment of IoT?

To satisfyingly answer this question, the following ancillary questions need to be answered:

a. **What are the characteristics of ICN caching when applied to the IoT?**

What metrics can be used to characterise caching efficiency in IoT? What is the relationship between caching decision strategy and cache replacement policy? How can we get the most out of the limited caching resources of a typical IoT device? All of these questions need to be answered before attempting to design an ICN caching solution for the IoT.

b. **How does network topology affect caching effectiveness and how can sufficient caching performance be achieved regardless of topology type?**

The relationship between topology and caching is intricate and worth exploring. Identifying which caching strategies work best with which topology type can provide useful information if the topology type of a given network is known and unchanging, while developing a caching strategy that is effective regardless of topology would be of great benefit to deployments with unknown and/or dynamic network topologies.

2. **How to achieve efficient caching in distributed IoT applications while minimising memory requirements?**

Using what was learnt from the previous research questions, how do we go about designing a caching strategy for information-centric IoT that acknowledges the specific characteristics of that domain and takes optimal advantage of the idiosyncrasies of IoT? How do we leverage topology effects to determine optimal caching locations in the network? Furthermore, how could Quality of Service (QoS) considerations for differently prioritised traffic flows be integrated into such a system?

3. **How to balance caching duties in the network so that available resources are evenly utilised without sacrificing performance?**

Devices that happen to occupy optimal caching locations are likely to experience stronger load than their neighbours. How can we pass some of that load on to underutilised devices without drastically increasing content delivery latency? The challenge is to find a non-intrusive way for neighbours to communicate their respective caching loads and to transfer cached data between each other.

1.4 Research Goals

The goal of this thesis is to explore ways to ensure reliable and efficient data delivery in IoT applications using ICN.

The specific research objectives of this thesis are as follows:

- **Objective 1: Characterise the behaviour of ICN caching in the IoT.**

  The aim of this objective is to produce a comprehensive overview of the factors affecting caching performance in information-centric IoT. This includes analysing existing caching strategies and how they could be deployed to IoT devices, defining suitable metrics for quantifying caching performance as well as designing an experiment setup suitable to compare and contrast different approaches to caching using the aforementioned metrics. Furthermore, effects unique to the IoT environment, such as the wireless network topology, need to be discussed and characterised and their impact on caching performance quantified.

- **Objective 2: Design a lightweight topology-independent caching strategy for information-centric IoT.**

  Using the knowledge gained from Objective 1, the aim of this objective is to provide a general in-network caching solution for IoT that can offer certain performance guarantees even in deployments where the topology is unknown or dynamic.

- **Objective 3: Design a lightweight and efficient off-path caching strategy for information-centric IoT.**

  The aim of this objective is to reduce the risk of overtaxing individual devices identified as ideal caching candidates, as well as to better utilise all caching resources available in the network while keeping the communications overhead to a minimum.

1.5 Publications

- Jakob Pfender and Winston K.G. Seah. Leveraging Localisation Techniques for In-Network Duplicate Event Data Detection and Filtering. In *Proceedings of the 42nd IEEE Conference on Local Computer Networks (LCN)*, October 9–12, 2017, Singapore.
- Jakob Pfender, Alvin Valera, and Winston K.G. Seah. Performance Comparison of Caching Strategies for Information-Centric IoT. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, September 21–23, 2018, Boston, MA, USA.
- Jakob Pfender, Alvin Valera, and Winston K.G. Seah. Easy as ABC: A Lightweight Centrality-Based Caching Strategy for Information-Centric IoT. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, September 24–26, 2019, Macao, China.
- Cenk Gündoğan, Jakob Pfender, Michael Frey, Thomas C. Schmidt, Felix Shzu-Juraschek, Matthias Wählisch. Gain More for Less: The Surprising Benefits of QoS Management in Constrained NDN Networks. In *Proceedings of the 6th ACM Conference on Information-Centric Networking*, September 24–26, 2019, Macao, China.
- Cenk Gündoğan, Jakob Pfender, Peter Kietzmann, Thomas C. Schmidt, Matthias Wählisch. On the Impact of QoS Management in an Information-Centric Internet of Things. In *Computer Communications*, Volume 154, March 15, 2020, Pages 160–172.

1.6 Organisation of Thesis

The rest of this thesis is organised as follows: Chapter 2 provides an overview of relevant research in Information-Centric Networking and the opportunities and challenges it brings to IoT. It also provides a comprehensive overview of the most prominent caching decision strategies and cache replacement policies and their applicability to the IoT context. Chapter 3 presents experimental evaluations and performance comparisons of several different caching approaches, with a particular focus on content delivery latency and how it is affected by network topology. In Chapter 4, these findings are applied to the design of a novel,

topology-independent caching strategy that is specifically designed for deployment in IoT and accounts for the constraints of the domain. In Chapter 5, the caching strategy proposed in the previous chapter is further refined to include an off-path caching component that enables it to make better use of available caching resources in the network while reducing load imbalances and keeping overheads to a minimum. Chapter 6 presents a further extension to existing caching strategies that allows traffic of different Quality of Service (QoS) levels to be treated appropriately without sacrificing performance or increasing complexity. This contribution was developed in cooperation with researchers from other institutions and as such is slightly removed from the rest of the research presented in this thesis; nevertheless, it provides novel insights into the role of caching under QoS constraints. Finally, Chapter 7 presents conclusions and an outlook of open questions and potential future work.

CHAPTER 2

Literature Review

This chapter will give an overview of the existing body of research relevant to this thesis. It will discuss the family of Information-Centric Networking (ICN) approaches and their associated challenges, with a particular focus on Named Data Networking (NDN) and its potential for the IoT. Existing research on the question of caching in the Information-Centric IoT is discussed in detail, as the opportunities and challenges it brings are foundational motivations for this thesis.

2.1 Information-Centric Networking

Content-Centric Networking (CCN), was first proposed by Jacobson *et al.* in 2009 [73]. CCN and its successors are often grouped together with similar approaches under the umbrella term *Information-Centric Networking (ICN)*. Their main contribution constitutes a complete overhaul of the existing approach to networking, replacing the host-based addressing system of IP with a new system that treats named content objects as first class network entities. CCN decouples location from identity by basing its routing logic entirely on the unique names of the routed content objects instead of unique addresses of hosts. This allows network participants to be agnostic about where their requested content actually resides in the network, as well as enabling transparent systems for in-network caching and replication of data,

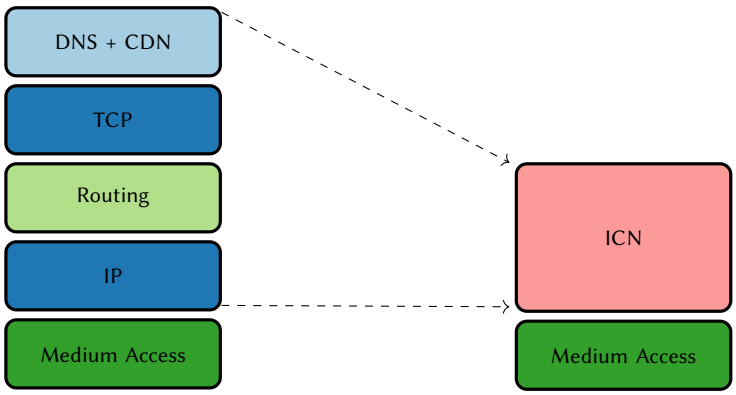


Figure 2.1.: Comparison of TCP/IP and ICN protocol stacks

thus increasing availability and performance. It also leads to a simplification of the network stack: DNS becomes obsolete, routing becomes optional, and the network functionality of TCP/IP and application-layer functionality such as Content Delivery Networks (CDNs) are compressed into the ICN layer (see Figure 2.1).

The original design of the Internet's architecture was based on point-to-point connections between individual hosts. The development of CCN and its ICN relatives was driven by the observation that this does not reflect the way the Internet of today is primarily used: as a network for the distribution of information. ICN allows consumers to focus on the actual data they need without having to define specific, physical locations from which to retrieve this content.

Furthermore, decoupling content from its physical location also makes it possible for identical copies of this content to be stored at multiple locations in the network, thus making it more easily accessible for consumers that have high latency to the original producer. This is similar to the concept of Content Delivery Networks (CDNs), but here it is integrated directly into the network layer instead of being a closed ecosystem on the application layer.

2.1.1 ICN Architectures

While CCN and its successor NDN (see below) are the best-known representatives of the ICN family and will be the focus of this thesis, there are several other architectures that fall under the same umbrella. The most noteworthy ones are briefly described here. *PSIRP / PURSUIT* [56, 136] is a blank-slate approach that replaces the IP protocol stack with a publish-subscribe architecture in which the topology (i.e., the routes between nodes) is actively managed based on the subscriptions that exist. Unlike CCN, data delivery does not necessarily take the same path as the corresponding request, making on-path caching less viable. *NetInf* [46], the architecture of the *SAIL* project, is based on a hierarchy of caches of different sizes and capabilities, with content placement determined by popularity. This is a heavyweight approach and likely unsuitable for IoT deployment. *COMET* [58] provides an ICN-like architecture that nevertheless explicitly allows for location-aware services, such as retrieving information from a specific subset of nodes. The CONVERGENCE project's *CONET* [47] architecture is a transitional approach that reuses parts of the existing IP architecture to provision a content-aware architecture that can run either natively or as an overlay. It is otherwise fairly close to CCN in design, but routing information for data packets is stored in the interest packet instead of at the content routers, and routing as a whole is more tightly integrated into the architecture. Finally, the *MobilityFirst* [129] project is an ICN architecture specifically designed to treat mobile devices as first-class citizens. As such, names are even more strongly decoupled from network entities than they are in CCN, with entities having globally routable IDs to ensure reachability.

The ICN family also includes some technologies that are older than CCN, such as *DONA* [77], which already incorporated some of the concepts that would later come to define CCN. However, CCN was the first approach to fully define most of the concepts now considered integral to ICN.

Xylomenos *et al.* [157] provide a comprehensive survey on the past, present, and future of ICN.

2.1.2 Named Data Networking

Named Data Networking (NDN) [163], which has emerged as one of the most popular ICN solutions in recent years, is a comprehensive implementation and extension of the ideas brought forward by CCN. While it is not the only ICN implementation under active development, research suggests that it is the most suited for IoT applications thanks to its scalable naming technique and flexible approach to caching [14]. Therefore, it will be discussed here as a representative of ICN as a whole, since most of its aspects are generalisable to that larger field.

Like CCN, NDN represents a shift from the host-centric architecture of IP towards a content-centric architecture, where the primary function of the network is not delivering packets to specific addresses, but fetching data identified by unique names. Current NDN research is funded by the United States National Science Foundation (NSF) as part of its Future Internet Architecture Program.

NDN defines two fundamental types of packets used for communication: *Interest* and *Data*. When a consumer wants to request content, it puts the unique name of that content into an Interest packet and sends it to the network. Routers in the network then forward the Interest towards the named data's producer. Once the Interest reaches a network node that has the named data, a Data packet carrying the data is returned to the consumer.

This mechanism is made possible by the router architecture specified by NDN. An NDN router defines three NDN-specific structures: A *Content Store (CS)*, a *Pending Interest Table (PIT)*, and a *Forwarding Information Base (FIB)*, as illustrated in Figure 2.2. When an NDN router receives an Interest packet, it first checks whether it has data in its CS that matches the Interest. If it does, it can immediately satisfy the Interest by replying with a Data packet containing the named data. If the router does not find the named data in its CS and does not have a PIT entry corresponding to the named data, it consults its FIB to decide on which interface to forward the Interest. The FIB is a routing table that maps names to interfaces. The router uses its FIB to forward Interests it cannot satisfy on one or more interfaces. Routers may use an arbitrarily complex forwarding strategy to choose the best interface for each name. Once the router has forwarded the Interest, it creates an entry in its PIT that names

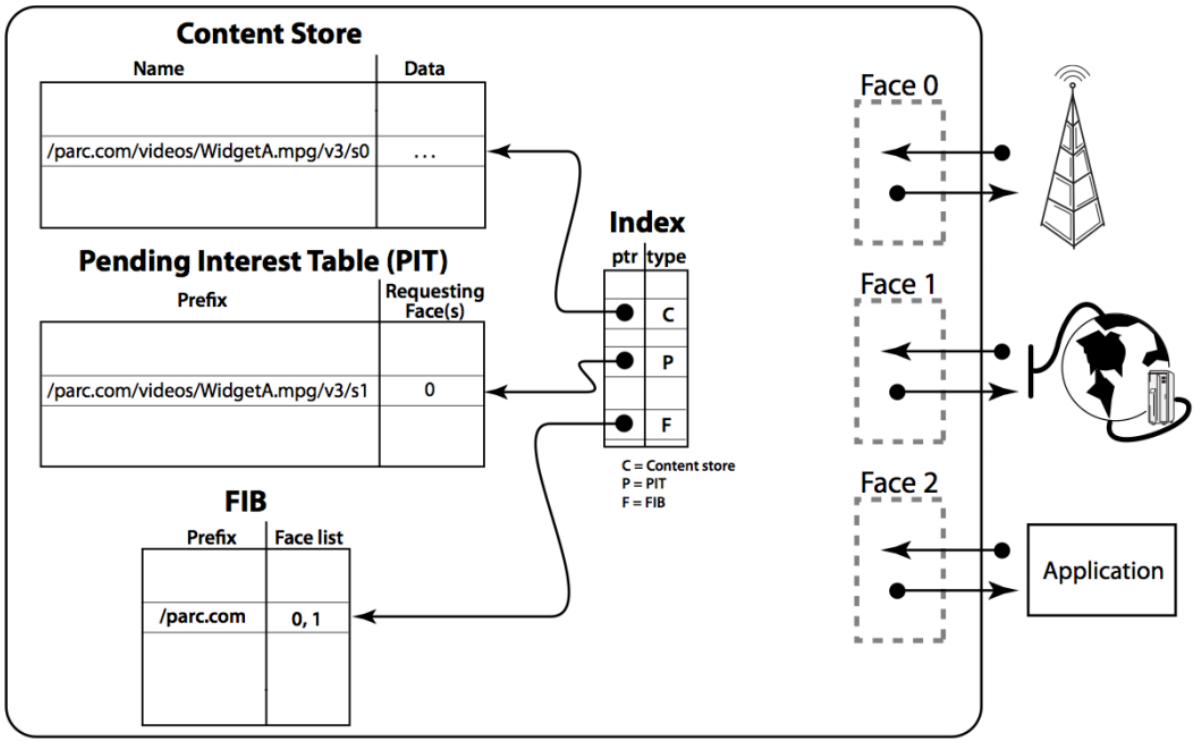


Figure 2.2.: NDN router components [73]

the requested data along with the interface on which the Interest was received. Each received Interest is first checked against the PIT before it is forwarded. If a corresponding PIT entry is found, this means that the router has already forwarded an Interest for the same data, and instead of forwarding the Interest, it simply adds the new interface to the PIT entry. When the router receives a Data packet, it checks the name of the contained data against its PIT, forwards the Data to all interfaces listed in the corresponding entry, and, if possible, saves the Data in its own CS. The forwarding logic described above is illustrated in Figure 2.3 and broken down into the components of a single NDN router in Figure 2.4.

The mechanisms described above result in a hop-by-hop forwarding paradigm that ensures that multiple requests for the same piece of data are consolidated at each hop and that content is duplicated and cached at various points in the network. Due to the way Interests are recorded at the routers, a Data packet always travels the reverse of the path taken by the Interest(s) that requested it. Thanks to the built-in caching mechanism, popular data objects are automatically replicated across the network, making them more accessible and reducing congestion. NDN also supports native multicast by allowing routers to forward Interests on more than one interface. This also leads to greater stability of the network, as information retrieval is not dependent on links to individual hosts.

Extensive work has gone into extending and refining all aspects of ICN. One of the major changes introduced by NDN concerns the role of the forwarding plane. In traditional IP routing, the forwarding plane is stateless and simply follows the rules set by the routing plane. In NDN, on the other hand, since the routers keep track of incoming Interests, the forwarding plane is stateful. Furthermore, since NDN has the built-in capability to forward Interests on multiple interfaces, its forwarding mechanism is described as *adaptive forwarding*. The research conducted by Yi *et al.* [159, 160] represents an in-depth examination of the capabilities and limitations of NDN's forwarding plane. Yi also proposes several new forwarding plane mechanisms for fast failure detection and congestion control [160].

A major question of CCN and NDN that was left unanswered by the original proposals was how producers can announce the names of their data to the rest of the network and how these names are propagated through the network — in other words, how routers construct their routing tables. Early implementations of CCN relied entirely on statically constructed

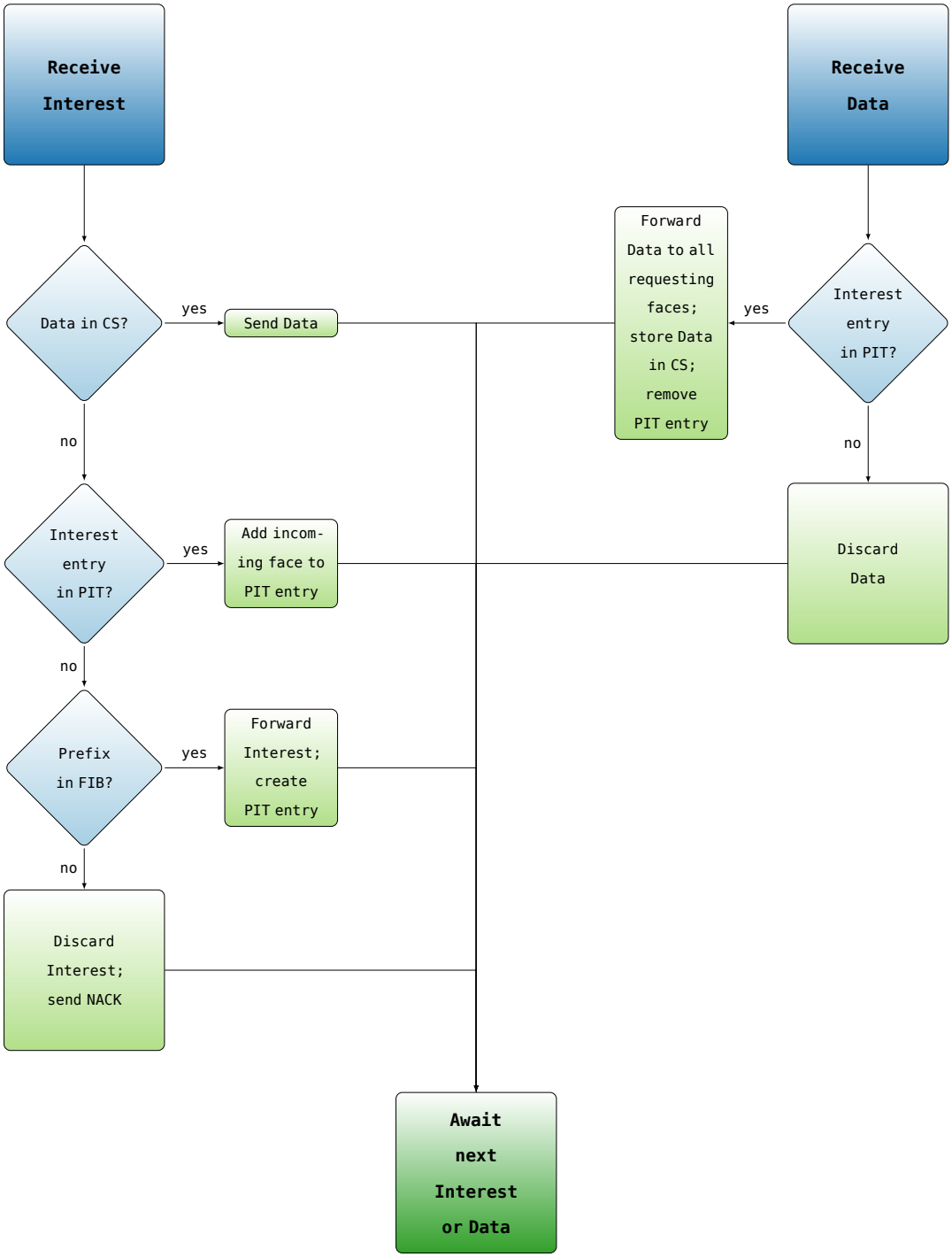


Figure 2.3.: Core operation loop of an NDN forwarder, triggered for each received Interest or Data

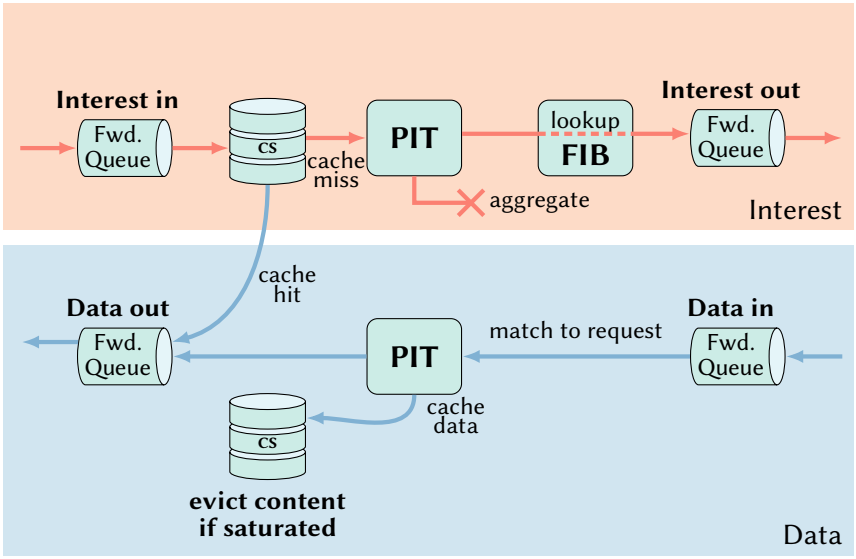


Figure 2.4.: Flow description for Interest and Data messages in an NDN forwarder (adapted from Gündoğan *et al.* [63])

FIBs. However, this approach is not scalable; dynamic solutions are required. Because name announcement in NDN has very similar requirements to prefix announcements in traditional IP routing, it is possible to develop appropriate solutions based on existing, well-understood routing protocols. A first step towards solving the problem was *OSPFN* [149], based on Open Shortest Path First (OSPF), which still used IP addresses to identify routers and could only compute a single best hop for any given name. The next iteration of NDN routing protocols was *NLSR* [70], which adapts the Link State Routing (LSR) approach of OSPFN to use only NDN infrastructure. NLSR uses NDN's Interest and Data packets to disseminate routing information and is able to rank the available forwarding options for each data name to enable adaptive forwarding.

Since NDN has neither end-to-end virtual channels nor ports, IP's socket API obviously cannot be used for communication. Moiseenko and Zhang [99] propose a producer-consumer API for NDN that is designed to replace the socket API. It introduces *producer* and *consumer contexts*, which associate NDN names with various functions related to their production and processing, respectively. The API includes functions for content verification, packet framing, caching, content-based security, and namespace registration.

Afanasyev *et al.* [5] discuss the question of fragmentation in NDN. End-host or end-to-end fragmentation, as common in IP, is not feasible for NDN due to its hop-by-hop nature. Interest packets need to be complete at each hop for the NDN router to be able to decide how to answer them, and Data packets need to be complete in order to be matched against the PIT. In addition, IP's pre-fragmentation of data based on path Maximum Transmission Unit (MTU) is not possible in traditional NDN because of the reusability of Data packets. Therefore, NDN uses a *hop-by-hop fragmentation and reassembly strategy (HBH F/R)*, in which packets are fragmented at each hop according to the next hop's MTU and then reassembled there. The reference implementation of the HBH F/R protocol is *NDNLP* [133]. The impossibility of pre-fragmentation in NDN has since been called into question by newer strategies such as *STNDN* [119], which allows for independent routing of fragmented content chunks, with aggregation only occurring at selected nodes along the transmission path.

2.1.3 Security and Stability

Security and stability play an important role in any networking technology. Since content in ICN is allowed and even encouraged to be replicated at multiple points in the network, steps need to be taken to ensure integrity and provenance of data objects. NDN solves this problem by requiring data producers to cryptographically sign every Data packet they produce [73]. The question of whether to actually trust a given producer is left to the consumer. Requiring content to be signed can also help mitigate *prefix hijacking* [159]. Since public keys can be distributed via NDN, it is also possible to encrypt data objects.

Burke *et al.* [33] propose a full-fledged security paradigm for NDN, demonstrated using a lighting control system. Their approach makes use of well-known public-key cryptography methods to sign not only Data, but Interest packets bearing commands in order to provide trust in actuation commands. The public-key infrastructure requires some out-of-band bootstrapping in order to provide network participants with the root public key of an Authorisation Manager (AM), but once trust in the AM is established, the AM can delegate trust management of arbitrary namespaces to selected network participants. Commands issued to actuators then take the form of an Interest with the name containing the intended receiver,

the issuing application, the command itself, and an authentication nonce computed using the application's private key. The receiver can then verify the command using the application's public key (and optional Access Control Lists (ACLs) to ensure access rights in addition to provenance).

In terms of stability, NDN has the benefit of being immune to traditional DDoS attacks, since direct addressing of individual hosts is not supported. However, there is a similar attack vector called *Interest Flooding*, in which attackers oversaturate the network with bogus requests for (non-existent) data, leading to congestion and choking out legitimate Interest packets. Afanasyev *et al.* [4] propose an effective approach, taking advantage of NDN's stateful forwarding plane, to combat this type of attack by introducing a satisfaction-based pushback mechanism that limits data rates to consumers based on the rate at which their Interests can be satisfied.

Wählisch *et al.* [143] provide a comprehensive overview of the different attack vectors that are theoretically feasible in NDN.

2.1.4 Quality of Service

Because it comprises more diverse resources (forwarding queue, PIT, and CS) than IP, ICN lends itself more naturally to Quality of Service (QoS) considerations than traditional networking. Several approaches have been proposed within the Internet Research Task Force (IRTF)'s ICN research group [64, 74, 98, 108, 109]. Potential solutions include encoding service classes into content names [74, 98] or extending ICN message types with reliable versions of Interest and Data packets [139]. However, most existing solutions may lead to inflation of names or PIT entries as they allow for content with the same name to have different service classes. This runs counter to the idea of aggregating Interests and Data at the PIT and CS based on their names. This could prove infeasible for solutions intended for constrained devices.

2.1.5 Applications

Several purely NDN-based applications have been developed during the development of NDN itself, such as *Chronos* [174], a serverless multi-user chat service.

NDN seems to be particularly promising for the building automation domain, which presents unique challenges in terms of sensing and actuation. Prominent examples of building management systems using NDN are the lighting control system proposed by Burke *et al.* [33] (which can be generalised to describe any sensing-actuation system), *NDN-BMS* [130], a building management system deployed at UCLA, and a disaster management system for smart campuses proposed by Ali *et al.* [7]. The latter also has significant overlap with ICN-related research in the IoT (see Section 2.1.8).

NDN support is also being brought to browsers and the client side of the web thanks to projects such as *NDN.js* [131], which for example allows easy browser-side interaction with NDN content using the `ndn://` URI scheme.

2.1.6 Publish-Subscribe Patterns and the Push Primitive

One key area in which the NDN approach has been found to be lacking is publish-subscribe based architectures and applications that rely on push operations. At its core, NDN is designed with consumer-initiated content distribution in mind, i.e. individual pieces of content are only transmitted from producer to consumer if explicitly requested by the consumer. In publish-subscribe architectures, on the other hand, consumers express a general, open-ended interest in any (future) content a specific producer offers, which will then be delivered to them continually without explicit prompts. This type of content delivery is not explicitly provided for in NDN.

While some of the other ICN technologies described in Section 2.1.1 offer native support for publish-subscribe, such as COMET [58] and PSIRP [136], these architectures typically come with much more significant overheads than NDN does. Consider for example PSIRP's reliance on explicit topology managers and use of Dynamic Hash Tables (DHTs) and Bloom filters. These represent a layer of complexity that is, in all likelihood, not a worthwhile tradeoff in exchange for publish-subscribe functionality, particularly in the IoT space. Instead, the research trend within NDN is oriented towards supporting the paradigm using existing primitives.

A naïve approach to implementing publish-subscribe functionality using NDN could be to have consumers periodically send Interests for content they want as a way of simulating subscriptions. However, this content may or may not actually exist, which could result in high messaging overheads. Moreover, consumers would not be able to control the update frequency if consumers decide how frequently requests are sent. An alternative approach would be to introduce special long-lived Interests that are not discarded upon being satisfied, instead remaining at intermediate nodes' PITs until they expire or are cancelled by the consumer. However, this would introduce further complexity to Interest handling, particularly if combined with regular (not long-lived) Interests.

Carzaniga *et al.* [40] suggest exploiting the fact that there exists a symmetry between how producers in NDN register prefixes and how consumers in publish-subscribe architectures register interests. Both work by registering prefixes in forwarding tables, which are then used to aggregate and forward Interests to producers (in NDN) and update notifications to consumers (in publish-subscribe). They argue that when viewed this way, the only difference between NDN and publish-subscribe in terms of forwarding is the source of the forwarding information: in NDN, the producer, and in publish-subscribe, the consumer. However, this symmetry is broken on the opposing side. An Interest packet is expected to result in a Data response, while a push message satisfying a subscription is one-way. In addition, NDN Interests are not necessarily forwarded all the way to the producer who announced the prefix if they can be satisfied along the way, whereas a push message must always reach its intended recipient. In order to combine the two paradigms, a network must thus be able to differentiate between three types of messages: one-way messages, messages that expect a reply, and replies.

In the IoT space, Gündoğan *et al.* have developed *HoPP* [61], a publish-subscribe scheme for information-centric IoT networks that utilises link-local unicast advertisements to inform subscribers of the availability of new content. Subscribers can then request this new content using the standard NDN paradigms of sending an Interest and receiving the Data in return. The authors find that HoPP is robust and resilient and that its performance in the IoT is even better than comparable non-ICN approaches such as MQTT [60].

2.1.7 Remote Procedure Calls

Remote Procedure Call (RPC) [104], also known as Remote Method Invocation (RMI), Remote Function Invocation (RFI), or Distributed Object Communication, is a set of principles governing the communication between distributed objects in a network. It allows network participants to call functions implemented by other network participants without having to explicitly account for the fact that the callee is a different machine. While ICN was not explicitly designed with RPC in mind, its communication primitives lend themselves naturally to its implementation. An Interest could easily represent the function call, while the corresponding Data represents the return value. This fact was exploited early on in the development of NDN and has resulted in the proposal of *Named Function Networking (NFN)* [138] as well as more recent iterations such as *Named Function as a Service (NFaaS)* [80].

These solutions extend the named data model of ICN to include remote functions in the set of requestable objects. While NFN is a purely functional approach, the later NFaaS generalises the concept to create a more dynamic solution that allows nodes in the network to download (i.e. cache) functions and use them locally. This way, other ICN primitives, such as caching and forwarding strategies, can be applied to optimise the distribution of functions in the network.

A contribution to this field that recently garnered a lot of positive attention is *RICE* [78], which addresses previously unsolved challenges such as authentication and authorisation, parameter passing, and support for non-trivial operations. This idea has since been further developed into an approach called *Compute First Networking* [79].

2.1.8 ICN and the IoT

Research on how to apply ICN paradigms to Wireless Sensor Networks (WSNs) and the IoT is still a comparatively young field. However, researchers are laying the foundations of what is sometimes called the *Named Data Network of Things (NDNoT)* — in particular, current research is focused on how to adapt and optimise existing ICN strategies to the unique environment of IoT, with its unreliable links and devices that are constrained in both memory and processing power.

Amadeo *et al.* [10] discuss the relative merits of NDN for the IoT as opposed to other data-centric approaches such as *Directed Diffusion (DD)* [72] — which is also centred around a hop-by-hop forwarding paradigm — and propose a modified NDN scheme called *dd-NDN* that integrates aspects of DD into NDN, such as a Next Hop Table (NHT) that can be used to express path preferences for Interest and Data packets.

Baccelli *et al.* [21] discuss the potential benefits ICN can bring to IoT environments. They ported the NDN implementation **CCN-lite**[1] to **RIOT-OS** [19, 20], an operating system for the IoT, and conducted real-world experiments using an existing testbed consisting of 60 nodes distributed across three university buildings. They tested two different approaches to routing. The first is a simple Interest flooding mechanism that does not require any FIB entries, fitting the memory constraints of IoT devices but placing excessive load on the network. The second is a reactive scheme that, after initially flooding Interests, constructs adaptive FIB entries based on received Data packets. The authors also evaluated the impact of in-network caching on network congestion and compared NDN's performance and memory footprint (using the reactive routing scheme) to that of the 6LoWPAN/IPv6/RPL network stack. They found that NDN's RAM and ROM usage is less than a quarter of RPL+6LoWPAN's, and the load it places on the network (in terms of sent packets), even with in-network caching disabled, is less than half. However, they note that NDN packet processing, especially related to name prefixes, is a CPU bottleneck, which is especially problematic for constrained devices with limited processing power. Long names in particular are an issue, as the small MTU typically found in IoT environments can lead to a single Interest packet being split into several transmissions.

[1] https://github.com/cn-uofbasel/ccn-lite

The authors suggest losing the soft requirement of NDN names being human-readable, as a majority of communications in typical IoT applications is machine-to-machine and using binary representations for names would greatly reduce the amount of packet space taken up by the name. It must however be noted that this would preclude the ability to use cryptographically generated names. The authors also propose several IoT-specific enhancements to the ICN paradigm, such as forwarding content requested by multiple neighbours with a single multicast message and optimistically caching unsolicited content chunks received from broadcast messages.

Melvix *et al.* [97] propose a context-aware forwarding strategy for the NDNoT that takes into account the relevant application's tolerance for inaccuracy when making forwarding decisions, such as whether to respond to Interests by reusing cached data or data from similar queries. Interests from different domains (i.e. different applications) may be mapped to the same producer, but the applications might have different requirements regarding accuracy and freshness, which may necessitate different forwarding decisions. This is solved by inserting an additional context classification step — the *Domain Resolution Engine* — between the normal name resolution and forwarding decision steps taken when an Interest is received. This way, Interests from an application that requires accurate and fresh data can be forwarded directly to the producer without wasting time checking the CS, while less critical Interests may be satisfied by using cached data.

A comprehensive and up to date overview of the state of the art of information-centric IoT research is presented by Djama *et al.* [49]. Their contribution is a survey and meta-survey that introduces a taxonomy of existing research and discusses and compares current research and identifies open research questions in the areas of caching, QoS, security, in-network processing, publish-subscribe patterns, and more.

2.1.9 Summary of Information-Centric Networking

ICN is a highly active research area with significant potential in IoT, but also many unanswered questions. It remains to be seen how faithfully its fundamental concepts can be translated to the context of IoT and what new issues may arise from this transition. As the field is still young and growing rapidly, any snapshot of existing research will not reflect the state of the art for very long.

2.2 Caching Strategies in Information-Centric IoT

ICN has been shown to have a lot of promise for IoT applications, thanks to its content-centric nature and slim network stack [2, 13, 16, 71, 91, 106, 141, 145, 158]. However, one of the core tenets of traditional ICN is the automatic and indiscriminate caching of any and all received content. This does not translate well to the IoT, where resources such as memory are often severely limited. In traditional networks, it is conceivable to have dedicated caches with virtually unlimited storage space, or even to have each participating node cache all content it receives. In IoT, however, this is neither possible nor productive. Instead, consideration needs to be given to the question of what content and how much of it should be cached, how long it should be cached, and what content should be replaced once the cache inevitably fills up. While traditional ICN has attempted to answer these questions, most of the proposed solutions still rely heavily on large caches. If this resource is unavailable, their performance suffers.

Although transferring ICN caching to the IoT comes with these limitations, it is still a promising avenue of research. Especially in multihop scenarios, where packets have to traverse multiple lossy wireless links, on-path caching capabilities can help reduce network load caused by retransmissions. If a data transmission fails due to a lossy link but the content was cached on an intermediate node, subsequent retransmissions will only need to fetch the data from the caching node instead of the original producer, thus reducing the total volume of traffic

caused by retransmissions while increasing the likelihood of successful transmission. Gündoğan *et al.* [60] show that in a multi-hop scenario, ICN with on-path caching can outperform IP-based protocols in terms of flow balance and goodput thanks to this feature. Thus, it is worthwhile to pursue the feasibility of caching strategies in information-centric IoT.

In recent years, considerable amounts of research have been conducted into adapting traditional ICN caching strategies to the specific needs of IoT. Several different approaches were developed to counteract the limited resources as well as take advantage of the wireless, ad-hoc nature of IoT networks. Broadly speaking, a caching strategy can be applied on two different aspects: it can affect the *caching decision*, i.e. whether or not an incoming content chunk is to be stored in the cache, or it can affect the *cache replacement decision*, i.e. which piece of cached content to replace if a new content chunk is to be placed in a cache that has reached its capacity. Most proposed caching strategies only affect either the caching decision or the cache replacement decision, with a few holistic approaches that take both into consideration.

Ever since Jacobson *et al.* proposed CCN [73], the *Cache Everything Everywhere (CEE)/Least Recently Used (LRU)* approach introduced there has been questioned in regard to its suitability for CCN, NDN [163], and related ICN implementations [34, 123]. The research community has since reached the consensus that *CEE/LRU* is not the most optimal caching strategy [35, 116, 117]. This resulted in a wealth of new caching approaches being developed.

2.2.1 Existing Studies of ICN Caching in IoT

The usefulness of ICN paradigms in the domain of WSNs and IoT has been widely agreed upon [8–10, 21, 57, 90, 145, 168, 169]. However, the idiosyncrasies of IoT — wireless communication, limited resources, transient contents — call into question whether the approaches that have been shown to be successful in traditional ICN can simply be transferred over to this new domain or whether new solutions are necessary. While caching is not the only aspect of ICN affected by this, it is one of the most prominent since limited memory is one of the most obvious hurdles for any IoT application. Other IoT-specific challenges include mobility, dynamic topologies, unreliable communications, and energy efficiency [8, 11, 67, 168, 169].

Zhang *et al.* [162], Fang *et al.* [54], and Zhang *et al.* [165] provide comprehensive surveys and taxonomies for the major classes of caching strategies in traditional ICN, as well as presenting challenges and potential future research directions. Tarnoi *et al.* [137], Zhang *et al.* [165], Din *et al.* [48], Priscilla and Charulatha [115], and Naeem *et al.* [101] present comparative evaluations of several different caching approaches for traditional ICN. While these contributions are extremely valuable, it is not a given that the results can be applied directly to IoT environments. Likewise, Carofiglio *et al.* have produced a body of work [36–39] focusing on latency effects in traditional ICN caching, but their solutions also do not address the idiosyncrasies of IoT environments. The most comprehensive surveys of caching schemes for information-centric IoT were presented by Arshad *et al.* [14, 15] and Gupta *et al.* [65]. These papers, however, are pure surveys, with no experimental evaluation or comparison of the presented strategies.

So far the only comparative studies on ICN caching strategies specifically in the IoT were carried out by Hail *et al.* [69] and Meddeb *et al.* [94], both of whom use simulated environments for their evaluations. Hail *et al.* compared all four permutations of the *CEE* and probabilistic (with $p = 0.5$) cache decision strategies and the *Random Replacement (RR)* and *LRU* cache replacement policies (all of the strategies listed here will be described in detail in Sections 2.2.2 and 2.2.3). The performance metrics measured in their experiments were cache hit ratio, data retrieval delay, and Interest retransmissions. Meddeb *et al.* compared *CEE*, *LCD*, *ProbCache*, *Betw* [41], *Edge Caching* [55], and their own *Consumer-Cache* strategy in combination with the *RR*, *LRU*, *Least Frequently Used (LFU)*, and *First In First Out (FIFO)* cache replacement strategies and evaluated them in regard to cache hit ratio, number of evictions, hop reduction ratio, and data retrieval delay. At the time of writing, evaluations performed on physical IoT hardware operating in realistic conditions were sorely missing from the existing body of literature.

Table 2.1.: Families of caching strategies with selected representatives (non-exhaustive)

Layer	Family	Representative Strategies
Caching decision	Traditional	CEE/LCE [73, 163]
		LCD [84, 162]
	Static probability	$Prob(p)$ [68, 165]
	Dynamic probability	pCASTING [68]
	Topology-based	ProbCache [116]
	Centrality-based	Betw/EgoBetw [41]
	Implicit coordination	Li & Simon [89]
		Zeng & Hong [161]
	Explicit coordination	Liu *et al.* [92]
		PCS [100]
		EM3C [76]
	Edge-only	Edge Caching [55]
		Consumer-Cache [95]
		ELC [17]
	Freshness-based	Quevedo *et al.* [121]
		LCC [170]
		FDC [12]
	Popularity-based	Vural *et al.* [142]
		TCCN [134]
	Off-path	Dräxler & Karl [51]
		Saha *et al.* [126]
		NCP [107]
Cache replacement	Traditional	Random Replacement (RR)
		Least Recently Used (LRU)
		LFU [137]
	Hybrid	Modified-LRU [81]
		LRFU [118]
	Content-based	MDMR [66]
	Centrality-based	NICE [75]

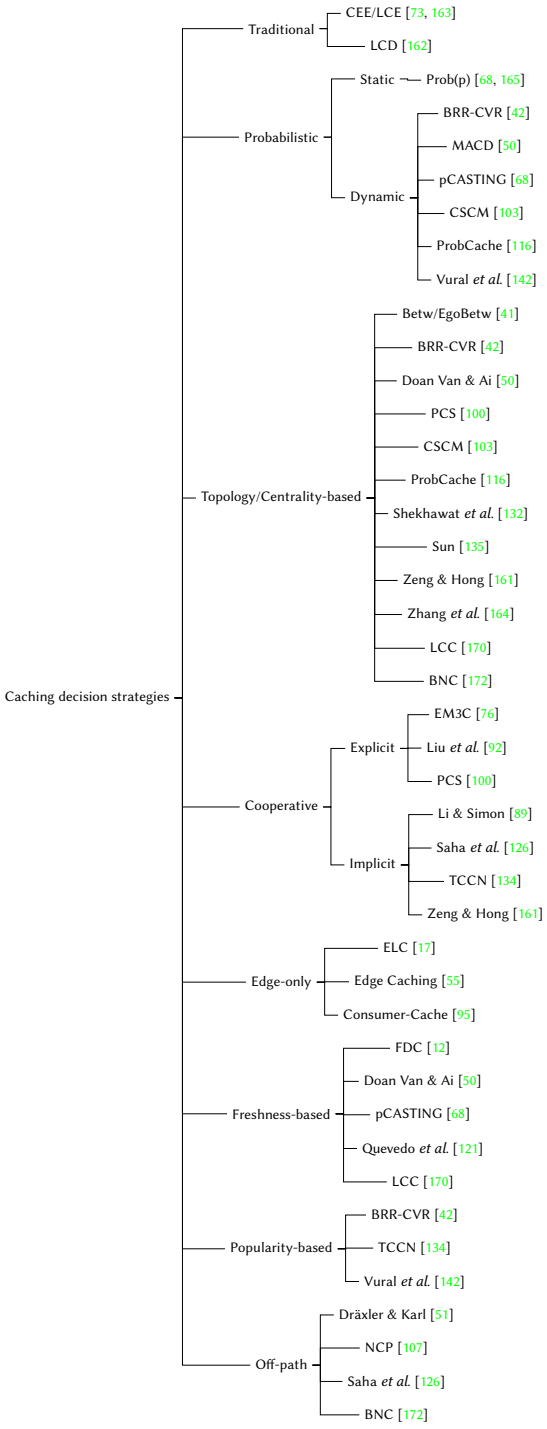


Figure 2.5.: Taxonomy of modern caching decision strategies

The caching strategy can affect two distinct aspects of the caching process: the *caching decision* and the *cache replacement decision*[2]. Since most existing approaches focus on only one of these aspects, each aspect will be discussed individually in the following sections, along with some strategies that change the behaviour of the respective aspect. Table 2.1 shows an overview of the different families of strategies for caching decision and cache replacement, with representatives for each.

2.2.2 Caching Decision

The caching decision needs to be made whenever an ICN node receives a content chunk it does not yet have in its Content Store (CS). It determines whether the new content chunk will be stored in the CS or discarded.

This section introduces different approaches to caching strategy with varying complexity. For the most relevant strategies, a pseudocode definition is provided. Every strategy features two functions, HANDLE_INTEREST() and HANDLE_DATA(), which define what the strategy does upon reception of an Interest or Data packet, respectively. There are also functions that are not further defined in the pseudocode, such as `canSatisfy()`, which returns `true` if the incoming Interest can be satisfied locally and `false` otherwise; `getData()`, which retrieves a content chunk from the local CS; and the NDN primitives `reply()` (for replying to an Interest with a Data packet), `forward()` (for forwarding Interest and Data packets to the next hop), and `cache()` (which stores content in the CS if it is not already stored there).

Caching strategies can be broadly categorised into a few different families, depending on what information they use to reach their caching decision. These families, along with representative caching strategies, are shown in Figure 2.5. Note that some strategies may fall into two or more categories; this is reflected in the taxonomy.

[2] A third aspect, called *cache coherency*, is also sometimes identified [15]. It describes the process of checking the validity of cached contents. However, few existing studies focus on this aspect and its impact on performance and content delivery is mostly negligible.

Algorithm 1 Cache Everything Everywhere (CEE)

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          reply(Data)
5:      else
6:          forward(Interest)
7:      end if
8:  end function
9:
10: function HANDLE_DATA(Data)
11:     cache(Data)
12:     forward(Data)
13: end function
```

Cache Everything Everywhere (CEE)

The most straightforward caching decision strategy, *CEE* (also known as *Leave Copy Everywhere (LCE)*) is the strategy that is used in traditional ICN [73, 163]. Nodes will attempt to cache every incoming content chunk that is not already in their CS (see Algorithm 1). In traditional ICN, caches can generally be assumed to be relatively large. Therefore, this strategy is viable and causes little overhead. Its main advantage is that it offers the fastest possible propagation of content through the network — any node that receives content will immediately store it[3], ensuring rapid replication of all available content. However, especially in a strongly connected network, it also leads to high redundancy: *every node caching every piece of content means that every CS in the network will look roughly the same.* This is where the crux of the problem lies when considering a network with severely limited caching capabilities.

It has become consensus [35, 116, 117] that this redundancy leads to suboptimal performance. This can be shown as follows: The totality of the many small caches in the network can be interpreted as one single, stratified cache. However, if every individual cache stores the same content, that means the network's total caching capabilities are not being used to their fullest

[3] Some IoT-specific implementations go so far as to have nodes cache all content they overhear, regardless of whether the transmission was intended for them or not.

extent. Instead, caches tend to hold only the most recent content. This can be particularly problematic with certain traffic patterns, especially if content requests are cyclical, as it can lead to *thrashing* — i.e., content keeps cycling out of the cache before it is requested again, leading to continuous cache misses. It would therefore be desirable to reduce overall redundancy by ensuring that different caches carry different contents, thus increasing the chance that content from any given producer will be replicated in at least one cache in the network. This is how *diversity* will be defined in the remainder of this work (see Section 3.1.5): The ratio of unique content objects in all caches to unique content producers. The more advanced caching strategies discussed below attempt to maximise this metric.

It should be noted that diversity is not necessarily desirable in every IoT scenario. In applications where the content request pattern is very skewed (e.g., using a Zipfian distribution [32] with $\alpha > 1$) maximising diversity will actually hurt performance as cache resources are wasted on less popular content. However, the closer content requests are to the uniform distribution, the more beneficial cache diversity becomes, as a larger percentage of content objects are duplicated in the network. Since uniform request distributions are more common in IoT contexts [94, 122], cache diversity will be emphasised in this thesis.

Leave Copy Down (LCD)

If caching everything at every node is not an option, but the caching process is to remain simple, *LCD* [84] (Algorithm 2) is a viable option. In *LCD*, content is always cached only at the next hop from the node where the cache hit occurred, i.e. initially one hop downstream from the producer, and one hop further downstream with each subsequent request. This is achieved by extending the Data packet with a Time Since Birth (TSB) field, which counts the number of hops since the "birth" (i.e. creation or retrieval from CS) of the Data packet. TSB is initially 1 and is incremented every time the Data packet is forwarded. Nodes only cache Data with a TSB of exactly 1.

LCD is a somewhat "conservative" caching strategy that tends to keep content close to the producer, but still alleviates load on the core. Since every cache hit results in the content being cached one hop closer to the requesting consumer, popular content that is requested with high frequency will gradually move closer to consumers with each step.

Algorithm 2 Leave Copy Down (LCD)

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

There is also a variant of *LCD* called *Move Copy Down (MCD)* in which instead of simply copying a content chunk to the next node whenever a cache hit occurs, that content chunk is explicitly *moved* to the next node — it is deleted from the current cache and only stored in the next cache. This frees up cache space for new content near the core.

Probabilistic Caching, Static Probability

To increase cache diversity and decrease redundancy, the easiest solution is to simply introduce a certain probability that any given content chunk will not be cached. The most straightforward version of this approach, referred to in this thesis as *Prob(p)* (see Algorithm 3), simply sets an *a priori* probability p that a given node will store a given content chunk. Upon receipt of a new content chunk, the node generates a random number between 0 and 1. If the generated number is smaller than p, the content is stored in the cache; otherwise, it is forwarded without being cached. This has the obvious effect that not every content chunk is cached at

Algorithm 3 Prob(p), where p is constant

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          reply(Data)
5:      else
6:          forward(Interest)
7:      end if
8:  end function
9:
10: function HANDLE_DATA(Data)
11:     if rand() < p then
12:         cache(Data)
13:     end if
14:     forward(Data)
15: end function
```

every node, thus effectively increasing cache diversity across the network. An important implication is that more popular content has a higher chance of being stored at more points in the network since it is encountered more often. More generally, if a node receives the same content chunk n times, that chunk's caching probability at that node is $1 - (1 - p)^n$ [165].

It is easy to see that probabilistic caching increases diversity and reduces redundancy across the network's caches. This means that on average, the number of unique content objects that have copies stored in the network is increased. It also inherently favours popular content, thus increasing the cache hit ratio (see Section 3.1.1). However, the complexity in getting the most out of probabilistic caching lies in choosing the best value for the caching probability p. It is intuitively clear that a lower p will result in lower cache redundancy. However, too low a value for p may result in underutilisation of available resources.

In approaches where p is set to a fixed value, the most commonly chosen value is $p = 0.5$ [68]; however, other probabilities, such as 0.7, 0.3, 0.1, and even 0.01 have also been considered [137]. It should be noted that *CEE* can be treated as a special case of probabilistic caching with $p = 1$. In previous studies, a lower p has been found to correlate with better performance [14, 68, 69, 116, 137, 165]. A definitive lower bound for p (i.e. a value for p at which the caching probability is too low to result in effective caching) has yet to be conclusively determined.

Probabilistic Caching, Dynamic Probability

Instead of defining an *a priori* caching probability that is the same for every caching decision at every node, a technique can be designed that dynamically computes a caching probability for each individual node or even for each content chunk, based on available information, in order to adapt the caching behaviour to the state of the network. The strategies can be based purely on node-local information, such as the current contents of the cache or the node's battery levels; they can be based on properties of the incoming content chunk, such as its age, type, or producer; or they can be based on information from the wider network, such as the position of the caching node in the network topology or the cache contents of neighbouring nodes.

Hail *et al.* propose *pCASTING* [68], which uses local information for its probability calculation. It considers the freshness (age) of the content chunk as well as the node's battery level and its cache occupancy when calculating p. The calculated probability is directly proportional to the battery level, inversely proportional to the cache occupancy, and directly proportional to the content object's *residual freshness*.

The residual freshness FR of a received content item is defined as:

$$FR = 1 - \frac{currentTime - t_s}{f}, \tag{2.1}$$

Algorithm 4 pCASTING

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          reply(Data)
5:      else
6:          forward(Interest)
7:      end if
8:  end function
9:
10: function HANDLE_DATA(Data)
11:     CachingUtility ← w_1 · myEnergy + w_2 · (1 − myCacheOccupancy ) + w_3 · Data.ResidualFreshness
12:     if rand() < CachingUtility then
13:         cache(Data)
14:     end if
15:     forward(Data)
16: end function
```

where *currentTime* is the time at which the item was received at the caching node, t_s is the time at which the item was created, and f is the "freshness class" to which the item belongs. For example, $f = 10\,\text{s}$ means that the producer creates a new version of the item every ten seconds unprompted.

This means that FR is affected by both the absolute difference between production and reception of the data item ($currentTime = t_s \rightarrow FR = 1$) as well as the freshness class (items that are produced in larger intervals and are thus expected to remain fresh for longer result in a higher FR).

The final caching probability is then calculated using the caching utility function F_u, which is defined as:

$$F_u = w_1 \cdot EN + w_2 \cdot (1 - OC) + w_3 \cdot FR, \tag{2.2}$$

where EN is the energy of the caching node, OC is the cache occupancy of the caching node, and FR is the residual freshness of the received item as defined in Equation (2.1).

The weights w_i ($0 \leq w_i \leq 1$ and $\sum_{i=1}^{N} w_i = 1$) can be adjusted according to application preference. The value produced by F_u is the probability of a node caching the data item in question (see Algorithm 4).

The authors compare their caching strategy with *CEE* using *RR* (see Section 2.2.3) and find improvements in both energy efficiency and retrieval times. The utility function F_u also has the advantage that it can be modified to include other content or node parameters, making this approach highly flexible. The authors do not discuss alternative parameters in their evaluation, but it is easy to see that the strategy could be extended using any number of other relevant metrics.

pCASTING is just one representative of a broader family of caching strategies that make use of dynamically calculated caching probabilities. These can take a multitude of weighted factors into account when determining their probabilities. Such approaches may, for instance, use some combination of node battery level and cache occupancy in their calculations, along with some information about the incoming content, e.g. its freshness [50] or its popularity [42], whether the content is already cached in a neighbouring node [164], and/or topological information such as hop count [50, 164] or the caching node's centrality [42, 103].

Topology-based Caching

If the caching decision is to be based on more than just local information, there are many other factors that can be considered. One of the most promising approaches is represented by the family of strategies that considers the network topology.

Psaras *et al.* propose *ProbCache* [116], which computes the caching probability of a given content chunk based on the distance between producer and consumer as well as the location of the caching node on the path. For a given content chunk travelling a path between producer and consumer, *ProbCache* determines the *cache weight* at each node, which is determined by the Data packet's Time Since Birth (TSB) (see *LCD* above) as well as its Time Since Inception (TSI), which is the number of hops between the creation ("inception") of the corresponding Interest packet and the cache hit, i.e. the total length of the path between consumer and producer. This means that content chunks are cached with a higher probability

Algorithm 5 ProbCache

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          Data.TSI ← Interest.TSI
5:          Data.TSB ← 1
6:          reply(Data)
7:      else
8:          Interest.TSI ← Interest.TSI +1
9:          forward(Interest)
10:     end if
11: end function
12:
13: function HANDLE_DATA(Data)
14:     Data.TSB ← Data.TSB +1
15:     CacheWeight = Data.TSB / Data.TSI
16:     if rand() < CacheWeight then
17:         cache(Data)
18:     end if
19:     forward(Data)
20: end function
```

towards the edges of the network (i.e., closer to the consumer), adjusted by the length of the Interest packet's path. The authors find that in traditional ICN, *ProbCache* increases the cache hit ratio, reduces the average number of hops required to hit requested content, and reduces the number of cache evictions.

As will be discussed in more detail in Sections 2.2.4 and 3.3 as well as Chapter 4, depending on the network's logical topology, caching closer to the consumer may not necessarily be the most efficient caching strategy. In some topologies, caching closer to the producer is more beneficial. *ProbCache* is easily modified to take this consideration into account by simply inverting the caching probability such that content is more likely to be cached near the producer. This modified strategy, called *ProbCache-Inv* here, is identical to *ProbCache* in every way except that the final caching probability is inverted (i.e. line 16 in Algorithm 5 reads `if rand() < (1 - CacheWeight)`).

Other topology-based approaches make use of the concept of topology potential [135, 172] in their caching decision. Topology potential is modelled on the physical concept of the gravitational field, where nodes exert effects of different strengths on their neighbours depending on the distance between them. In a similar approach, Shekhawat *et al.* [132] propose a caching strategy that centrally allocates caching roles to nodes based on their topological importance, which is based on their reference localities. However, these strategies suffer from the fact that they require global knowledge (and often some form of centralised control) and are inflexible in the face of dynamic topologies.

Centrality-based Caching

Centrality-based caching strategies are a sub-family of the topology-based approaches. They are given their own section here because they make up the foundation for the contributions in Chapters 4 and 5.

Betweenness centrality [28, 154] is a metric originally devised to describe network effects in social networks. It describes the number of times a given node lies on one of the paths between all pairs of nodes in the network and has been found to be a useful indicator of node importance in a network [146]. This makes it a useful measure to characterise topology effects in any type of network, including in the domain of ICN. Its usefulness for caching-related decisions extends beyond the domain of ICN and it has also been used as the basis for caching strategies in a variety of other networking scenarios [59, 171].

In ICN caching, centrality has been utilised in several ways. Bernardini *et al.* [23] use it to prioritise the caching of content produced by more central (i.e. popular) nodes, while Rossi and Rossini [124] use several centrality measures, such as betweenness, closeness, degree, and others, to determine how much caching space should be available at a given node. They find that the simplest metric — degree centrality — is generally sufficient to achieve satisfactory results.

Algorithm 6 Betw/EgoBetw

```
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
```

For an actual caching decision strategy, Chai *et al.* [41] propose *Betw* / *EgoBetw*, which use betweenness centrality to cache content at the most central nodes in the network. The idea is 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.

In this scheme, the centrality of each node is pre-computed offline (the details of this computation are described in Section 4.1). Each Interest that is sent through the network then records the highest centrality value among nodes it encounters, and the corresponding content chunk is then cached at the node with the highest centrality on the way back (see Algorithm 6). That way, content is automatically stored at the most central locations in the network, where Interests are most likely to result in cache hits.

Topology- and centrality-based approaches have the advantage of offering a holistic approach to caching and — at least in theory — could allow us to cache data in the optimal location. However, when transferred to the IoT, there are some important questions and caveats that need to be addressed.

Caching based on centrality presupposes that some nodes in the network are more important than others. The question, then, is whether this holds true in IoT networks. What "important" means in this context mostly depends on the problem that is to be solved by applying centrality — in the case of in-network caching, it would be the usefulness of that node as a caching location. Connecting this back to the second research objective of this thesis as stated in Section 1.4, caching usefulness clearly makes a node more important for achieving the stated goal of increasing content availability. Since a node's centrality is directly correlated to the number of Interests it can satisfy, it can be assumed that this claim holds and that more central nodes are in fact more important.

However, in terms of implementation, centrality-based approaches require a costly setup phase before they can begin operation, and if the topology is dynamic — i.e. in a network with mobile participants — these network-wide calculations may have to be repeated periodically, leading to significant overhead. Furthermore, caching only at the most central nodes will lead to an uneven utilisation of resources and a shorter battery life for those nodes that are already likely to be more taxed than the nodes at the edges purely due to being more likely to receive a majority of Interests. Thus, it remains to be seen whether these topology-based approaches are a good fit for information-centric IoT.

Cooperative Caching

Cooperative caching is an umbrella term for caching strategies that take more than local information into account — i.e., strategies in which nodes either *implicitly* or *explicitly* coordinate with their neighbours to ensure optimal caching. In explicit coordination, nodes may exchange information about their cache contents and/or the contents they have received on a periodic or ad hoc basis in order to make caching decisions, or even forward content chunks to one another for caching.

Algorithm 7 Labels(k)

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          reply(Data)
5:      else
6:          forward(Interest)
7:      end if
8:  end function
9:
10: function HANDLE_DATA(Data)
11:     if Data.ID mod k = myLabel then
12:         cache(Data)
13:     end if
14:     forward(Data)
15: end function
```

In implicit coordination, nodes follow *a priori* rules that govern what content they can cache, thus avoiding the need for explicit coordination. One example of this was proposed by Li and Simon [89] (Algorithm 7), where each node is assigned a fixed label $l < k$ (at setup time) and only caches content chunks whose IDs modulo k are equal to l. This ensures that cached content is automatically stratified into equal subsets and evenly distributed across the network without the overhead of explicit coordination between nodes. By adjusting k, it is possible to control the level of stratification. This caching strategy is referred to as *Labels* in the remainder of this thesis.

Zeng and Hong [161] propose an implicitly coordinated caching strategy that uses hop distance to determine the caching decision (Algorithm 8). Data packets are extended by a predetermined *data interval* value i. Each node along the path decrements this value by 1 when forwarding the packet. If a node decrements its value to 0, the packet is cached at that node and the data interval is reset to i. This ensures that data are implicitly cached at regular distances from producers without requiring any topological information or coordination. This caching strategy is referred to as *Intervals* in the remainder of this thesis.

Algorithm 8 Intervals(i)

```
1:  function HANDLE_INTEREST(Interest)
2:      if canSatisfy(Interest) then
3:          Data ← getData(Interest)
4:          Data.Interval ← i
5:          reply(Data)
6:      else
7:          forward(Interest)
8:      end if
9:  end function
10:
11: function HANDLE_DATA(Data)
12:     if Data.Interval = 0 then
13:         cache(Data)
14:         Data.Interval ← i
15:     else
16:         Data.Interval ← Data.Interval −1
17:     end if
18:     forward(Data)
19: end function
```

Explicit cache coordination strategies tend to be more complex than their implicit counterparts, requiring more communication among the nodes and more calculations to maintain a consistent state. Liu *et al.* [92] propose a strategy that coordinates nodes by constructing a virtual backbone network using graph-theoretical concepts to create a node hierarchy with core nodes responsible for caching. Similarly, Naeem *et al.* propose *Periodic Caching Strategy (PCS)* [100], a hybrid approach in which the most popular content is cached at the edges, while content evicted from edge nodes is stored at the nodes with the highest centrality.

As an explicit coordination strategy that takes inspiration from the domain of Software Defined Networking (SDN), Khodaparas *et al.* [76] propose *Extended Multi-Criteria Cooperative Caching (EM3C)*, which uses a centralised cache controller as well as a clustering mechanism for groups of IoT nodes to control the flow of content requests and direct them to the relevant caches in the network. The caching decision is also managed in a semi-hierarchical fashion, with cluster heads making localised decisions on where in their own cluster to cache content.

While the authors report improvements in cache hit rate and content delivery latency in a simulated environment, it is questionable how transferable this approach is given that centralised entities with global knowledge are not necessarily easy to realise in any given IoT scenario.

Edge-only Caching

Fayazbakhsh et al. discuss a strategy they call *Edge Caching* [55]. In this approach, only edge (i.e. leaf) nodes can cache contents. This approach has since between expanded upon by others [156]. However, the applicability of these strategies in the IoT domain is questionable, since in IoT, whether a node is an edge node or not is relative. Routing trees are rooted at the producer, but since every node can be a producer, a node may be an edge node relative to one producer but not to another.

Meddeb et al. developed *Consumer-Cache* [95], which is a slightly more flexible variant of *Edge Caching*. Instead of caching exclusively at edge nodes, *Consumer-Cache* caches at any node that is directly connected to a consumer, regardless of that consumer's location. In a similar vein, Asmat *et al.* [17] propose *Edge Linked Caching (ELC)*, which essentially extends the Consumer-Cache approach using content freshness as an additional decision metric.

Freshness-based Caching

Quevedo *et al.* [121] propose a caching method that is based on content freshness. This method is consumer-driven, meaning that requesting nodes may specify the desired freshness of the content, which is an extension of the producer-driven freshness mechanics of CCN. Consumers are able to specify their minimum freshness requirements (i.e., the maximum age of the content chunk) when requesting content. All Content Stores on the route then check the freshness requirement against the freshness of their stored data and decide accordingly whether to supply their cached data or forward the request towards the producer. This method allows for flexibility in that applications that do not require the freshest data will not cause unnecessary network traffic, but all applications benefit from updated caches whenever an application causes stale content to be renewed.

Similarly, Zhang *et al.* developed *Lifetime-Based Cooperative Caching (LCC)* [170], which takes applications' data freshness requirements into consideration and adjusts nodes' caching thresholds based on their position relative to the network core. This allows for flexibility in the caching decision, but requires *a priori* knowledge about nodes' positions in the network and would be infeasible for deployments with dynamic topologies or mobile participants.

Freshness is used in a different manner by Amadeo *et al.* [12]. They propose *Freshness-Driven Caching (FDC)*, which prioritises longer-lived contents in its caching decisions on the basis that caching short-lived contents is not an effective use of caching space. While the approach is primarily aimed at Vehicular Ad-Hoc Networks (VANETs), which tend to have even shorter content lifetimes than other IoT types, it may be of use to the larger IoT space as well.

Popularity-based Caching

Vural *et al.* [142] propose a strategy that uses the popularity of content classes to calculate the cost function for the caching of incoming content. Nodes cache content until it reaches a certain age, and for each incoming content chunk, the caching node calculates how many Interests the chunk can be expected to satisfy during its lifetime. Content that is expected to serve more Interests — because its content is more popular and/or because it is more fresh — is cached with a higher probability.

Popularity is also used in *TCCN* [134], where content is enhanced with tags and nodes keep track of the popularity of each tag as well as how often it is cached in nearby nodes, and *CPDI* [83], which presents a hybrid approach that also takes distance between nodes into account.

These popularity-based approaches are quite demanding of node resources, especially memory, since all nodes need to keep track of the global popularity of all content. In addition, approaches that rely on neighbours sharing information to estimate probability would incur an additional communication overhead. This combination of factors makes this class of strategies largely infeasible for the IoT.

Other Strategies

Many more caching decision strategies have been proposed, which can vary strongly in the type of information they consider and the way they arrive at a caching decision. For example, Nour *et al.* have proposed the *Near-ICN Cache Placement (NCP)* [107] strategy, which aims to minimise both caching and retrieval costs by explicitly moving content to a few highly connected nodes that are not adjacent to one another, while also allowing for content prioritisation by traffic class. However, much like the popularity-based approaches, this strategy requires full knowledge of the network as well as additional operations to move the content to the selected caching node, which implies a significant communications overhead.

2.2.3 Cache Replacement Decision

The cache replacement decision needs to be made whenever a content chunk is to be stored in a CS that has reached its capacity. The cache replacement policy determines which cached content chunk is to be evicted in favour of the new content chunk. Regardless of whether the node uses the default *CEE* caching approach or one of the more sophisticated ones introduced above, it has to have some method of deciding which content objects to replace. It should also be noted that in general, the cache replacement policy cannot undo the decision to cache the new content chunk. In other words, once the caching decision has been made, the new content chunk is guaranteed to be stored and the evicted content chunk has to be one of the previously cached chunks.

It should be noted that considerably less research has gone into cache replacement policies as opposed to the caching decision. Some authors argue [162] that cache replacement in ICN should be performed as fast as possible and that it is thus more desirable to have a simple and fast cache replacement algorithm rather than an efficient but complex one. However, it remains to be seen whether this also holds true in IoT, where caches are more valuable resources and should be used as efficiently as possible.

Least Recently Used (LRU)

The most widely used cache replacement policy and common in several other domains, *LRU* likely needs no further introduction. Each node keeps track of the Interests it serves and when it served them, and, when required to evict a content chunk, chooses the one that was least recently requested. The rationale behind this is that unpopular or outdated content will naturally be more likely to be removed, thus ensuring that only the freshest and most popular content is cached. In practice, this approach has been shown to be effective, although its performance can degrade if Interest patterns are cyclical and the number of popular content objects is greater than the size of the caches — an effect commonly referred to as *thrashing*.

As a variation on *LRU*, *Least Frequently Used (LFU)* keeps track of how often the objects in the cache are requested and evicts the least popular ones. In other words, where *LRU* records *when* content is requested, *LFU* records *how often* it is requested. While *LFU* mostly avoids thrashing and has been shown to result in good cache diversity [35], it performs poorly with variable access patterns, as it tends to overly favour items that experience spikes in Interest frequencies. This will eventually result in higher server load, as caches accumulate stale items instead of storing fresher content, forcing more Interests to be routed to the original producers [137]. *Modified-LRU* [81] and *LRFU* [118] are novel hybrid approaches that consider both recency and frequency and as such should be less susceptible to frequency spikes.

Meddeb *et al.* [96] propose a freshness-oriented variation of LRU called *Least Fresh First (LFF)*. It is aimed at scenarios in which Data can be updated with newer versions while using the same prefix. The strategy uses a time series model to predict how long a version of a Data item is expected to remain fresh. This prediction, called T_{fresh}, is calculated by the producer upon creation of the Data and attached to it as metadata. Whenever a cache eviction is triggered, the cache replacement strategy checks the T_{fresh} and the timestamp of the last time a Data with the same prefix was received (called *cache_time*) for each content in the cache and evicts the content with the lowest $T_{fresh} + cache\_time$.

Random Replacement (RR)

The least complex of the basic cache replacement policies, *RR* simply evicts a random content objects every time it needs to replace a chunk. It tends to be used more as a benchmark for evaluating other replacement policies rather than as an actual policy.

Max Diversity Most Recent (MDMR)

MDMR, developed by Hahm *et al.* [66] specifically for information-centric IoT, is a cache replacement policy that, as the name implies, aims to maximise the content diversity of individual caches while also keeping them up to date. It uses the original producers of content chunks to achieve this (it is assumed that chunk names include their original producers). When a new content chunk is to replace an old one, *MDMR* first attempts to remove an older cached chunk from the same producer. If the new chunk is from a previously unknown producer, *MDMR* attempts to remove the oldest chunk from a producer with more than one chunk in the cache. If the cache contains exactly one chunk per producer, it simply replaces the oldest chunk. Hahm *et al.* show that *MDMR* achieves better cache diversity on average than *RR*.

NICE

Khan *et al.* [75] propose their ***N**etwork-oriented **I**nformation-Centric **C**entrality for **E**fficiency in Cache Management (NICE)*, which is based on the well-known measure of centrality [28, 154]. However, unlike with the previously discussed *Betw*/*EgoBetw* caching decision algorithm (see Section 2.2.2), here centrality is used to manage cache contents at the replacement step. The authors introduce the new measures *NICE betweenness* and *NICE closeness* (based on betweenness centrality and closeness centrality [22, 125] respectively). Rather than making decisions based purely on network centrality (i.e. the number of paths a given node is on), NICE betweenness also takes content popularity into account. Replacement decisions are only made when they increase the NICE value of the cache in question. The authors find that

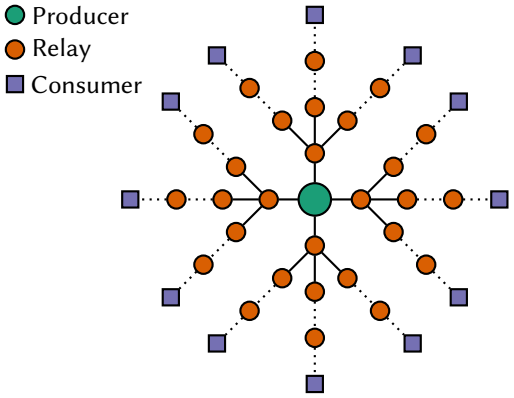


Figure 2.6.: Core topology

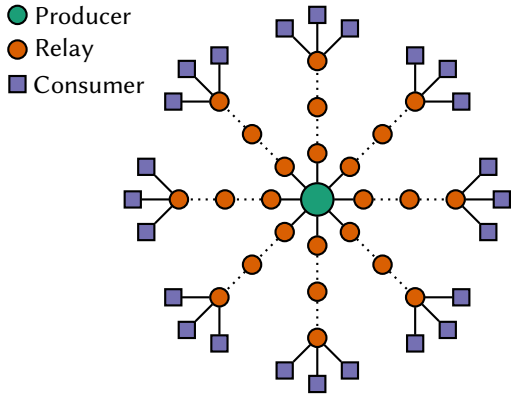


Figure 2.7.: Edge topology

NICE outperforms LRU in terms of cache hit rate, hop count, and content delivery latency. However, much like *Betw*/*EgoBetw*, while the actual replacement decision is made locally, the calculation of the centrality values relies on an *a priori*, centralised calculation, which makes it infeasible for deployment in IoT (this is expanded upon in more detail in Chapter 4).

2.2.4 Topology Effects on Caching

In the real world of IoT deployments, there are as many different network topologies[4] as there are networks. However, to get a sense of the effects of topology on caching performance, two generalised cases are introduced here — the *core topology* and the *edge topology* — that are on opposite ends of a spectrum of logical topology types. IoT topologies can fall anywhere on this spectrum, although most will tend to resemble one type more strongly than the other. These two extremes are showcased here in order to show the benefits of designing topology-agnostic approaches to caching, which will be explored in more detail in Section 3.3 as well as Chapter 4.

[4]Note that in this thesis, the term "topology" is generally used to refer to the logical topology from the perspective of ICN — i.e. the routes taken by Interest and Data packets as dictated by FIB entries, rather than the physical links between nodes.

Figure 2.6 shows an idealised *core topology*. A core topology is defined by the paths between the producer and the consumers intersecting nearer to the producer (the "core"); each path has only one consumer attached to it at the edge. In such a topology, the ideal caching location would be close to the producer (what Wang *et al.* call a *Type III* caching strategy [148]), as this would allow us to alleviate strain on the producer while serving the maximum number of consumers with cached copies of the data. Conversely, caching closer to the consumer would decrease the content delivery latency for that consumer, but no other consumer would gain any benefit from the cached copy.

Figure 2.7 shows an idealised *edge topology*. An edge topology is defined as having multiple individual paths from the producer out towards the consumers (the "edge"), which intersect further from the core. In this topology, it would be more beneficial to cache closer to the edge nodes where paths intersect (a *Type II* caching strategy [148]), as this would reduce the need for requests to be routed all the way to the core. Conversely, caching closer to the core would alleviate the strain on the producer, but the latency improvements would be minimal.

Looking at these two cases, it is easy to see that the network's logical topology has a significant effect on where content should be cached if latency is to be minimised. For this reason, as shown previously [113], 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.

2.2.5 Cache Fairness

Traditionally, most approaches to ICN caching assume a baseline of fairness and collaboration, with nodes cooperating to achieve the best performance on the global level. However, this is not necessarily the case. Furthermore, even if nodes ostensibly cooperate, the nature of the caching strategy may result in some nodes being treated "unfairly", that is, being starved of content or receiving much higher load than other nodes. This problem and how it relates to caching has been studied outside of the ICN domain, particularly in how Internet Service Providers (ISPs) and/or Content Delivery Networks (CDNs) may be encouraged to collaborate using protocol design [44, 45, 85, 110]. Naturally, this discussion has since found its way

to ICN [111, 147]. The first attempt to study in detail how fairness could be implemented in ICN was *FairCache* [151, 152], which interprets the caching problem as a Nash bargaining game [102] and derives heuristics that ensure individual fairness without sacrificing global performance. FairCache presents a distributed solution, which makes it ostensibly feasible for use in IoT. However, the performance price of implementing such complex heuristics on constrained hardware is unclear. Furthermore, IoT networks, which often come in the form of siloed deployments with all nodes controlled by the same entity, may not have a need for extensive fairness safeguards. Nevertheless, the issue of load balancing, which can be interpreted as a fairness problem, is present in IoT and may need to be addressed. A potential solution is found in *Off-Path Caching (OPC)*, which is discussed in the next section and in Chapter 5.

2.2.6 On-Path versus Off-Path Caching

In addition to asking the questions of which content to cache and which content to replace, it is also worth examining the paradigms of every ICN router acting as a cache and only caching content on the path it is propagated along. Regarding the latter, considerable research has been done to evaluate the relative merits of on-path versus off-path caching [43, 51, 140]. In off-path caching approaches, the caching decision is not restricted to whether nodes that receive content should cache it, but rather *where* received content should be cached. Nodes gain the capability to move received content to neighbouring nodes to cache it there, so as to achieve better cache spread and diversity as well as balance the load on individual nodes and thus increase cache fairness.

Saha *et al.* [126] take a similarly implicit approach to off-path caching as Li and Simon [89] (see *"Cooperative Caching"* in Section 2.2.2) did for cooperative caching. Their approach likewise uses the IDs of content chunks to manipulate the routes that content objects take through the network, coupled with minimal cooperation between adjacent Autonomous Systems (ASs) to coordinate which AS is interested in what kind of data.

As an alternative to off-path caching, it is also possible to instead allow content discovery to leave the straightforward delivery path. This can be achieved by using e.g. Bloom filters [25] to allow nodes to discover their neighbours' cache contents [86]. However, whether a lightweight implementation of these concepts that is suitable for IoT devices can be found is questionable.

2.3 Summary of Literature Review

The literature review shows some of the open questions in the field of ICN, particularly how it relates to the domain of IoT. ICN, being a very young field, has seen a flurry of research activity over the last few years, but also still has a wealth of open questions. As a candidate for a new IoT networking architecture, the potential of ICN is clear. However, there is currently no clear path towards fully accommodating the specific constraints of the IoT domain. Particularly the question of memory limitations poses significant hurdles and requires in-depth analysis. The aim of this thesis is to address some of these open questions in order to move towards a feasible solution.

CHAPTER 3

Performance and Content Delivery Latency of Caching Strategies for Information-Centric IoT

This chapter aims to investigate a range of ICN caching decision strategies and cache replacement policies in an IoT context. Particular attention is given to the evaluation of content delivery latency. To this end, a number of metrics suitable for measurements in the IoT domain are introduced. All potential solutions are examined through the lens of suitability for deployment in a constrained environment, which means that solutions have to be lightweight and ideally should not rely on global knowledge.

Since the stated goal of this thesis is the development of strategies for enabling reliable and efficient data delivery in the IoT, a definition of these terms is needed. In computer networking, reliability is often defined in the context of transport protocols. A protocol is called *reliable* if it notifies the sender about whether a transmission was successful or not (e.g., TCP versus UDP). However, this definition of reliability is not necessarily very useful for the IoT domain, where dropped packets are expected due to the wireless medium. A better way to frame reliability in the IoT context would be to view it in terms of metrics such as the *packet delivery rate*, which describes what percentage of packets are successfully transmitted, or the *retrans-*

mission rate, which counts how often packets need to be retransmitted before they reach their destination. In information-centric IoT, we can also examine reliability on the content level instead of the individual packet level, using metrics such as the *Interest satisfaction rate*, which measures how many Interests result in Data arriving at the consumer.

Efficiency is less straightforward to define, as its interpretation can vary strongly depending on the requirements of the system in question. This thesis will mainly interpret efficiency to mean low *content delivery latency*, which is given particular attention in the latter half of the chapter.

In this chapter, multiple different caching strategies are evaluated in two series of experiments using real IoT devices in a large physical testbed. Furthermore, the measurable effects of logical network topology, as introduced in Section 2.2.4, are examined.

3.1 Performance Metrics

In the existing literature, there is a wide range of metrics that have been used to evaluate the performance of caching strategies. This section will focus on a subset of these metrics that constitutes the most widely used ones, which will be used to evaluate the presented strategies.

3.1.1 Server Load and Cache Hit Ratio

In ICN, a *cache hit* occurs whenever an Interest is served by a cache in the network instead of the requested content's original producer. Conversely, a *server hit* is when the Interest needs to travel all the way to the original producer. The dual metrics of *server load* and *cache hit ratio* [41, 68, 137, 162, 165] measure the percentage of Interests that result in server hits or cache hits respectively. Cache hit rate can also be extended to *byte hit rate* [150], which additionally weights the hit rate by the size of the cached objects.

The server load L_S is derived as:

$$L_S = \frac{C_{server}}{C}, \tag{3.1}$$

where C is the total number of content objects retrieved and C_{server} is the number of content objects retrieved directly from the producer that owns their prefix. Analogously, the cache hit ratio R_{CH} is derived as:

$$R_{CH} = \frac{C_{cache}}{C}, \tag{3.2}$$

where C_{cache} is the number of content objects that are retrieved from the cache of an intermediate node that is not the prefix owner.

Ignoring dropped packets and retransmissions and counting only the content objects that arrive at their requesting consumers,

$$C = C_{cache} + C_{server}. \tag{3.3}$$

The two metrics L_S and R_{CH} give an indication of how well copies of popular content are distributed among caches across the network. The higher the cache hit ratio, the lower the server load, which in turn means less strain on content producers.

3.1.2 Content Delivery Latency

Many applications in the IoT space are time-sensitive. Therefore, finding an ICN caching strategy that minimises the delay between content request and delivery is desirable. The corresponding performance metric will be referred to as *content delivery latency* in this thesis; it is also sometimes called *data retrieval delay* [68].

It is intuitive that efficient in-network caching can reduce content delivery latencies by making content more readily available across the network [37]. Operating under the assumption that content, once produced, will remain useful for a certain amount of time and will be requested by multiple consumers during its lifetime, an effective caching strategy will minimise the distance between consumers and cached copies of the content they require.

Since content delivery latency is among the most important factors in efficient data delivery, it is examined in further detail in a separate series of experiments in Section 3.3, where the effects of network topology on latency are also studied.

3.1.3 Interest Retransmission Ratio

The *Interest retransmission ratio* [14] measures the percentage of Interests that are retransmitted due to having timed out. This metric is related to content delivery latency in that it is affected by congestion and density, but also the efficiency of the caching strategy. An ideal caching strategy would result in cache diversity that is high enough for Interests to always be satisfied upon initial transmission.

It should be noted that in ICN with on-path caching, as explained in Section 2.2, a retransmission does not necessarily imply that the content needs to be re-fetched from the same node it was originally sent from. If the content was transmitted part of the way before being lost, it may be cached in an intermediate node, thus decreasing the time required to obtain it. In other words, the amount of additional load and delay caused by a given retransmission varies depending on the state of the caches on the path, and subsequent retransmissions are likely to be less costly than the original.

By default, CCN-lite (the ICN implementation used for these experiments, see Section 3.2) allows a maximum of three retransmissions per Interest. This number is local to each individual node; i.e., each node will retransmit unsatisfied Interests up to three times before giving up. With particularly lossy links, this means that a single Interest might theoretically be retrans-

mitted dozens of times on its way from the consumer to the producer. In this evaluation, the *Interest retransmission ratio* is obtained by counting all Interests that were sent — regardless of whether they are retransmissions or not — and calculating how many of those Interests were retransmissions.

An alternative metric to this is the *Interest satisfaction rate* [6], which simply calculates the percentage of Interests that were satisfied without breaking down the number of retransmissions — an Interest is either satisfied, or it is not. These two metrics serve similar, but not identical purposes, in that the Interest retransmission rate indicates how reliable Interest transmission is in the hop-by-hop sense without necessarily indicating the proportion of ultimately successful requests, whereas the Interest satisfaction rate indicates the overall end-to-end success rate without considering hop-by-hop link quality.

In the experiments conducted for this thesis (see below), it was found that the Interest satisfaction rate was typically very high (above 90%), which means that the Interest retransmission rate is the more interesting metric, as hop-by-hop retransmissions may still be prevalent even if the end-to-end success rate is high. Therefore, Interest retransmission rate is the preferred metric in the rest of this chapter.

3.1.4 Total Cache Evictions

The number of *total cache evictions* [116, 126] is an indicator for how well the caching strategy is able to adapt the cached contents to the popularity and propagation of content in the network. If a caching strategy results in thrashing (see Section 2.2.3), this will be evident in an increased number of cache evictions. Conversely, if a caching strategy is able to effectively distribute contents within the network such that they can satisfy as many Interests as possible, cache evictions will be rarer.

3.1.5 Diversity Metric

The *Diversity Metric (DM)* [66] is a measure that indicates the diversity of cache contents across the network. DM is calculated as:

$$DM = \frac{|C_{disj}|}{|S|}, \tag{3.4}$$

where $|S|$ is the number of content producers and $|C_{disj}|$ is the number of disjoint name prefixes in all caches. The prefix, in this case, is an identifier for the producer of a content chunk. In other words, DM measures the percentage of individual content producers that are represented in caches in the network at any given time. Ideally, if cache diversity is a desired goal in a given application, DM should approach 1, meaning that every content producer has at least one content object stored somewhere in the network.

While DM can be a useful metric in scenarios where diversity is a priority, it only measures the diversity in terms of *producers*, not in terms of actual content. In a real network, it is possible for some content producers to be much more prolific than others, producing significantly more unique and distinct content objects than others. In this case, DM would not be able to capture how well this producer's contents are distributed across the network as opposed to the remaining content. In other words, an additional content-level diversity metric is required.

3.1.6 Cache Retention Ratio

The *Cache Retention Ratio (CRR)* [126] is a diversity metric that works on the content level. As such, it complements DM by measuring the ratio of distinct objects that are stored in caches at a given time to all generated objects. CRR is given by:

$$CRR = \frac{D_q}{D_p}, \tag{3.5}$$

where D_q is the number of unique objects currently stored in at least one cache and D_p is the total number of unique objects generated during the network's lifetime. It should be noted here that since cache size is limited, CRR will obviously decrease over time as new content objects are constantly being created while cache capacity stays the same, meaning objects

will eventually vanish from the network entirely. Although this is expected, when comparing caching strategies, it is interesting to examine how fast CRR deteriorates. In other words, CRR reduction over time describes how effective the caching strategy is at keeping content objects available in the network for as long as possible.

3.2 Performance Evaluation

This section constitutes a comprehensive comparison and evaluation of several classes of caching strategies for information-centric ICN. For this comparison, a series of experiments were run on the **FIT IoT-LAB** [3] open testbed. The IoT hardware used is IoT-LAB's specially developed **M3 node**[1], which has an STM32 (ARM Cortex M3) microcontroller with 512 kB ROM and 64 kB RAM and an Atmel AT86RF231 [18] 2.4 GHz transceiver operating on IEEE 802.15.4 [1]. The firmware for the nodes is a simple **RIOT-OS** [19, 20] application using **CCN-lite**[2] as the ICN implementation, modified to support the different caching strategies. ICN cache sizes are set to hold up to 20 objects, with all content objects produced in the experiment having uniform size.

The experiment setup consists of 60 M3 nodes distributed evenly across a single building (specifically, the *Lille* site[3] of the IoT-LAB testbed). The transmission range of individual nodes is not large enough to reach all other nodes in the network, thus creating a multihop setup with path lengths of two to three hops on average. Figure 3.1 shows the distribution of path lengths in all IoT-LAB experiment runs in this chapter.

The experiment is managed by a control script using the IoT-LAB API, which provides full control over all node serial interfaces. This makes it possible to manipulate their FIBs as required and extract status information, such as cached contents, at any time. All content a given node produces is prefixed with that node's unique ID. In an initial setup phase, all nodes broadcast their own prefixes. Nodes record the prefix announcements they receive

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

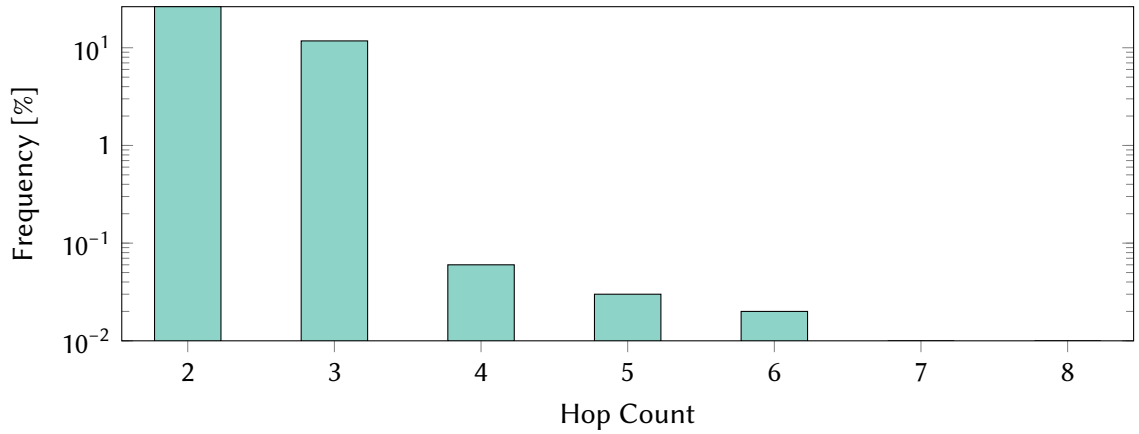


Figure 3.1.: Path length distribution (logarithmic). Single-hop paths are not counted as they can not benefit from caching. Paths with more than 8 hops can occur, but are too infrequent as to warrant inclusion here.

from their neighbours along with those neighbours' hardware addresses and the hop count value (which is initially 0) in their FIBs. All prefixes that resulted in new FIB entries are then rebroadcast with an increased hop count value. When forwarding Interests, nodes choose the corresponding FIB entry with the lowest hop count, with broadcast as a fallback.

After setup is completed, all nodes begin producing random content chunks in a random interval of 1 s to 5 s. The names of all produced chunks are collected by the experiment controller. The controller instructs each network participant to request a random piece of existing content every second (with up to ±500 ms of random jitter). The frequencies for publishing and requesting data were chosen as sensible maxima, as at higher frequencies, performance tended to degrade in this experiment setup and results became inconclusive. A more in-depth study of the effects of high-frequency content creation and consumption on the various performance metrics may prove useful in future research.

The distribution of the content requests can have two modes: they are either uniformly random or follow a Zipf-like pattern [32]. To achieve a Zipf distribution, the requestable content chunks are first ranked randomly and then assigned a request frequency that is inversely proportional to their rank. Thereby, the highest ranked content will be requested approximately twice as frequently as the second highest ranked content, three times as frequently as the

third highest ranked content, and so on. This results in a skewed request distribution. The Zipf distribution is commonly used in traditional ICN research because web content generally follows this pattern [55, 127, 144, 167]. However, IoT content requests are very different from web content; they depend on the application but generally tend to approximate uniform distribution [94, 122]. For this reason, this evaluation treats these two modes as the edge cases of request distribution, with real-world patterns likely to fall somewhere between the two. Thus, by examining both modes, the range of possible content distributions is covered by the presented experiments.

Each experiment runs for a total of 20 minutes, during which content is continuously produced and requested at the frequencies specified above. Each experiment is repeated 100 times.

The controller takes snapshots of cache contents every 5 seconds and logs all cache and server hits, cache evictions, and Interest packet retransmissions. This data is used to evaluate the caching strategies according to the performance metrics introduced in Section 3.1.

IoT is a large space, and the number of potential deployment scenarios is much too high to be reflected in a single experiment. Therefore, instead of focusing on a specific application, the experiment setup is intentionally generic and includes common elements that are likely to be found in many IoT deployments — class 2 constrained devices [29], a low-power and lossy network, and multihop machine-to-machine communication. This does not necessarily mean that the obtained results will be universally applicable to any IoT use case. Rather, the evaluation presented here is intended to serve as a baseline performance comparison using a variety of metrics that are useful in estimating the viability of a given caching strategy in a given scenario.

The following sections will evaluate the two aspects of caching strategies — the caching decision and the cache replacement decision — independently. In Section 3.2.1, caching decision strategies are compared by fixing the cache replacement policy to *LRU*, while Section 3.2.2 compares cache replacement policies by fixing the caching decision strategy to *CEE*.

3.2.1 Caching Decision Strategies

The three caching decision strategies *CEE*, *Prob(p)*, and *pCASTING* were evaluated using the most common cache replacement policy, *LRU*.

Recall that Section 2.2.2 explained that *pCASTING* can take into account three factors — the freshness of the incoming content, the node's cache occupancy, and the node's battery level — when calculating the caching probability. However, since the nodes in the IoT-LAB testbed have access to a constant power source, this implementation of *pCASTING* ignores the battery level and considers only cache occupancy and content freshness.

Server Load

The top plot of Figure 3.2 shows the average server load L_S, as defined in Section 3.1.1, for the different caching decision strategies, separated by request distribution.

It is intuitive that a probabilistic caching strategy would result in a higher server load, because the probability that a given content chunk can be found in a content cache is lower if content is not guaranteed to be cached. *pCASTING* causes less server load than the static probability approach, implying that on average, the dynamic caching probability calculated by its heuristic is greater than $p = 0.5$.

The differences between the Zipfian and the uniformly random request distributions are minor. When using the uniform distribution, server load is slightly higher for *CEE* and slightly lower for the other strategies. The reason *CEE* performs better for the Zipfian distribution is due to the skew in requests. Since every satisfied request results in the corresponding data being stored along the request path, a distribution in which some content is much more likely to be requested than other content makes it more likely for that content to be found in a cache. Due to cache sizes being limited, a uniform request distribution reduces this likelihood. On the other hand, probabilistic caching, while decreasing the overall likelihood of a given content chunk being stored, evens out the caching probabilities between less and more popular content by increasing cache diversity, which has an adverse effect on server load given a

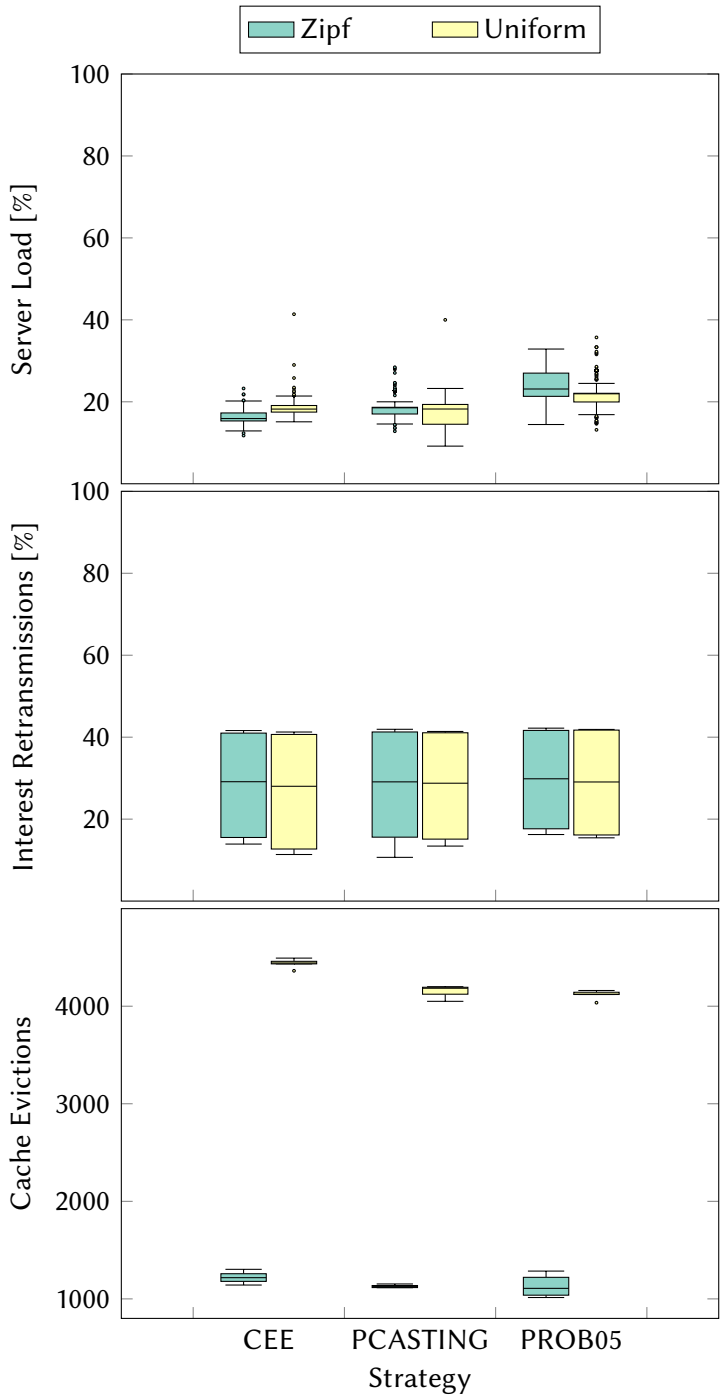


Figure 3.2.: Load, retransmissions, and evictions of different caching decision strategies

Zipfian distribution. Wherever there are outliers, these tend to be spread in the direction of higher load, implying a lower bound for server load. This is intuitive as some amount of server load is inevitable no matter how efficient the caching strategy, as at the very least the first copy of each new content chunk will have to be retrieved from the original producer.

Interest Retransmission Ratio

The results shown in the middle plot of Figure 3.2 make it fairly clear that different caching decision strategies have no significant effect on the percentage of Interests that have to be retransmitted. It is likely that the Interest retransmission ratio is influenced more strongly by factors in the network itself, such as topology and congestion. Interestingly, this would mean that another possible conclusion to be drawn from this is that caching strategies have no measurable effect on network congestion. One explanation for this observation is that the request frequency in this experiment is not high enough to observe significant congestion; however, even in experiment runs with increased request rates, this tendency did not change significantly.

It should be noted that these results conflict with the results reported by Hail *et al.* [69], who found more significant differences in average Interest retransmissions between different caching strategies. However, these differences may also be attributed to differences in experiment setup — their experiment features a less dense network with fewer nodes, where not all nodes act as consumers or producers, some nodes being pure relay nodes instead. It is possible that with increased reliance on multi-hop communications, differences in Interest retransmission rates become more noticeable. Hail *et al.*'s experiments were also carried out in a network simulator instead of a physical testbed, which might indicate that the influence of physical effects such as signals interference were not adequately modelled in the simulation.

Cache Evictions

The number of cache evictions is correlated with the rate at which content objects fill the cache. This can be observed when considering the results shown in the bottom plot of Figure 3.2, although the differences between strategies are minor and very consistent across experiment runs. The *CEE* strategy, which on average caches more packets than the probabilistic approaches, also exhibits a higher number of cache evictions.

Unsurprisingly, this metric shows a clear distinction between the two request patterns. Due to the skew in the Zipfian distribution, caches will accumulate the most popular content, reducing the likelihood of thrashing when compared to the uniform distribution. However, using a probabilistic caching strategy with a constant probability does increase the variance when the Zipfian distribution is used, as a content chunk's popularity has no bearing on its likelihood to be evicted. Conversely, *pCASTING* has minimal variance, showing that its probability distribution is better adapted to the Zipfian request pattern.

The number of cache evictions should also be examined in the context of total traffic produced in the network. In an average experiment run, the number of packets, including retransmissions, that are produced by all nodes varies between 40 000 and 75 000, while cache evictions peak at around 1250 or 4500 depending on request pattern. In other words, for the uniform distribution, between 5% and 12% of packets result in cache evictions, while for the Zipfian distribution, this number is between 1% and 4%. It should also be noted that the number of evictions is heavily dependent on cache size. The caches used in these experiments hold up to 20 content objects. If memory is more constrained and/or content objects are larger, the rate of cache evictions will increase.

Content Delivery Latency

Figure 3.3 shows the average content delivery latency — i.e. the average round trip time between the original transmission of an Interest and its satisfaction by the corresponding content, including any retransmissions — as a time series and averaged over the first quarter of the experiment duration respectively.

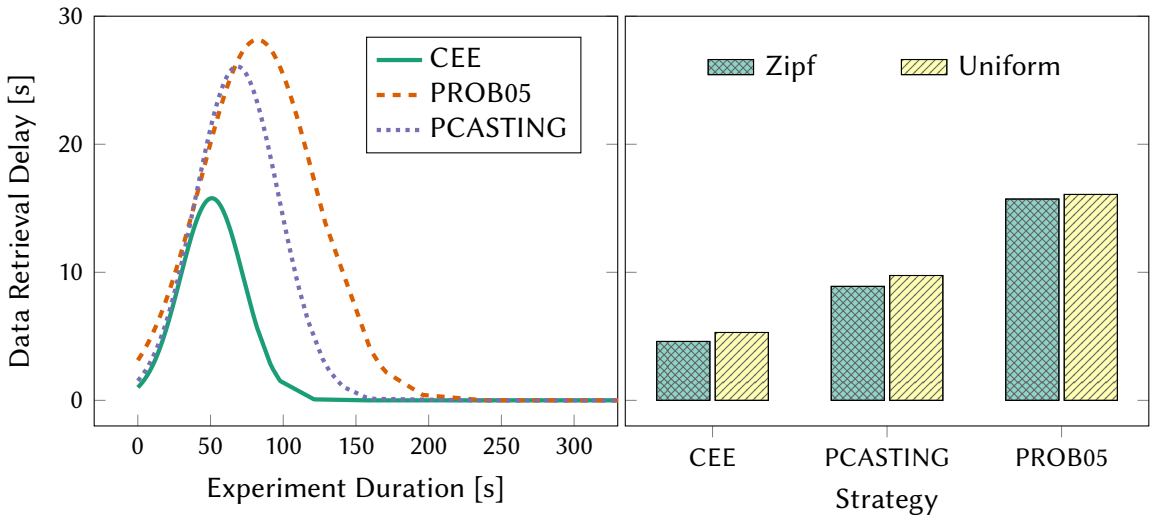


Figure 3.3.: Content delivery latency

At the beginning of the experiment, the latency is obviously orders of magnitude larger than after the caches have filled up, simply because any new content object can only ever be found at its original source until it has been requested at least once.

Unsurprisingly, the probabilistic approaches have much higher peaks than the *CEE* strategy, since they make it less likely, especially at the beginning of an experiment, for a content chunk to be found at a given cache in the network, thus necessitating more direct requests to sources. For the same reason, the average latency in the probabilistic approaches also declines at a slower rate, since caches take longer to fill.

Note that the left hand plot in Figure 3.3 is scaled to show only the first quarter of the experiment's duration, since there is no significant change in the average delivery latency after about 200 seconds.

To cut down on visual noise, the time series in Figures 3.3 to 3.5 show only the measurements taken with the uniformly random request distribution, as this distribution is closer to expected request patterns in a real IoT scenario. The right hand bar plots, meanwhile, show the comparisons between the uniform and the Zipfian distributions. For the content delivery latency, there is no significant difference between the distributions.

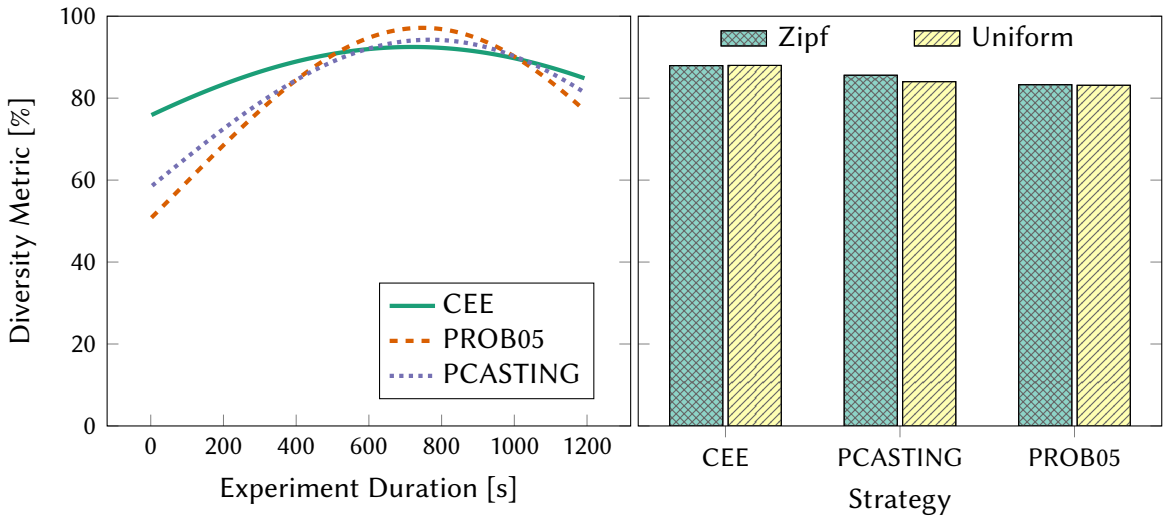


Figure 3.4.: Diversity metric

Diversity Metric

Figure 3.4 shows the Diversity Metric (DM) as a time series and as an average for the different caching decision strategies. As explained in Section 3.1.5, DM indicates the diversity of cache contents in all caches across the network by calculating the ratio of disjoint name prefixes in all caches to the number of distinct content producers. It therefore shows the percentage of content producers whose content is cached somewhere in the network at a given time.

Overall, the majority of content producers are represented in the network at any given time. However, the caching strategies differ in both the rate of distribution and the value at which they peak. *CEE* starts off at a higher diversity than the probabilistic strategies, since it caches all content objects from the start, thus filling its caches more quickly. However, as the caches fill up, the probabilistic strategies are actually able to surpass the diversity of *CEE*. This is due to the fact that *CEE*, while ensuring caches are used to their full capacity, leads to high redundancy — all caches in the network will have very similar contents since they cache every object they encounter. The probabilistic methods, on the other hand, result in more diverse caches, thus increasing the probability of all content producers being represented in the network to almost 100%, a peak that is not reached by *CEE*.

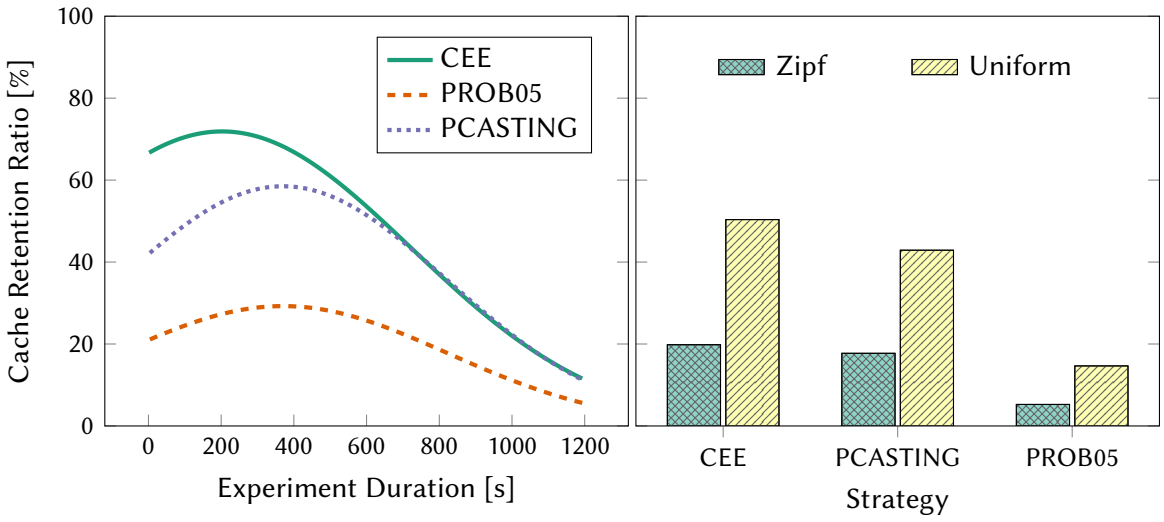


Figure 3.5.: Cache retention ratio

Cache Retention Ratio

As noted in Section 3.1.6, CRR naturally decreases over time as new content objects are constantly being created while cache size stays the same, meaning content chunks will inevitably fade out of the network entirely after a finite time. However, as shown in Figure 3.5, the rate at which CRR decays as well as its starting value vary depending on the caching strategy.

If a probabilistic caching strategy is used, caches take longer to fill up and peak at a lower value than the *CEE* approach. This is simply due to the fact that the rate of content creation is the same in all experiment runs, but the probabilistic approaches take longer to accumulate data, meaning that by the time they have reached cache saturation, the total number of content objects created is already greater than what the caches can hold. The static probabilistic approach exhibits a smoother rate of decline, since fewer cache evictions take place (see Section 3.2.1) and content thus has a longer average lifetime.

It is here in particular that a significant difference can be found between the probabilistic caching with static $p = 0.5$ and *pCASTING*. At the beginning of the experiment, *pCASTING* starts at a much higher peak than $p = 0.5$, and its CRR convergence is more similar to that of *CEE* than $p = 0.5$. This is an effect of the different factors that go into the decision made

by *pCASTING*. Since it takes into account both the freshness of the incoming content chunk as well as the cache occupancy (see Section 3.2.1), it acts differently when the cache is empty than when it is full. With an empty cache, the caching probability is much higher, as the corresponding part of the decision is inversely proportional to cache occupancy. This means that at the beginning of the experiment, *pCASTING* will act more like *CEE* or $p = 0.9$ until the caches fill up, at which point the probability will drop.

CRR also exhibits a clear distinction between Zipfian and uniform request patterns. The strong skew of the Zipfian distribution means that less popular content will fade much quicker while popular content is retained, making caches more homogeneous over time. A uniform distribution ensures similar lifespans for all content chunks, thus increasing the average CRR.

3.2.2 Cache Replacement Policies

The three cache replacement policies *LRU*, *MDMR*, and *RR* were evaluated using the most common caching decision strategy, *CEE*.

Server Load

The top plot of Figure 3.6 shows the average server load L_S for the different cache replacement strategies. It is immediately obvious that all three cache replacement strategies result in very similar server load rates; both the average values and the variances are very close across all combinations. It can thus be concluded that the cache replacement strategy has a negligible effect on this metric. Compare these results to those discussed in Section 3.2.1, where it was found that the caching decision strategy did have a measurable effect on server load. Therefore, while it might not be the most conclusive metric, it is still valuable for distinguishing between caching decision strategies. It may also be worth investigating how much of an impact other factors, like network topology, congestion, traffic shape, etc., have on the server load rate.

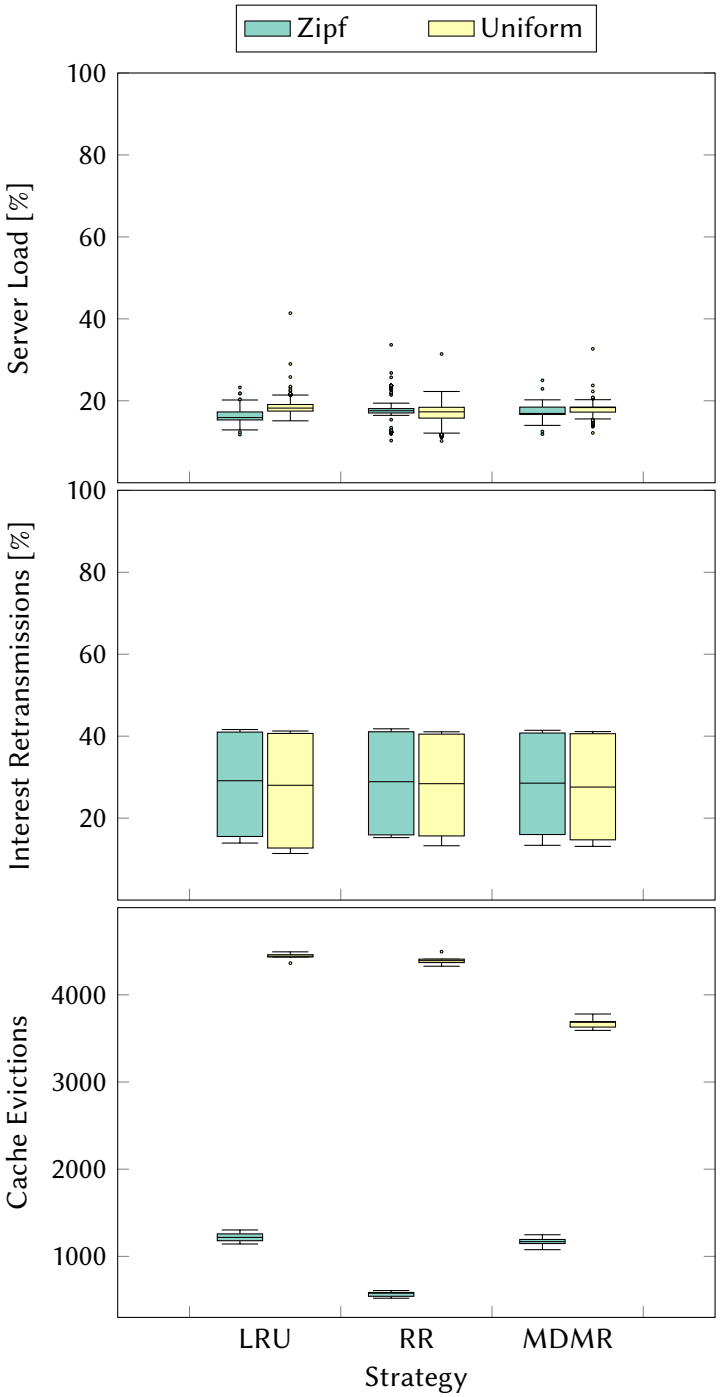


Figure 3.6.: Load, retransmissions, and evictions of different cache replacement policies

Perhaps most importantly, it is clear that *RR*, the simplest of the cache replacement strategies, does not perform any worse than the other approaches, which come with a much larger overhead. This supports the argument put forward by Zhang *et al.* [162] that complex heuristics in the cache replacement policy are not worth the effort and that *RR* meets all requirements with the added benefit of being simple and fast.

Interest Retransmission Ratio

The results shown in the middle plot of Figure 3.6 confirm what was established in Section 3.2.1 — the caching strategy appears to have no measurable effect on the average percentage of Interests that have to retransmitted. This is still a valuable result, however, since it shows that any variations in retransmission rates one might observe between different experiments most likely have causes independent from the chosen caching strategy.

Number of Cache Evictions

As noted in Section 3.2.1, the request distribution has a significant effect on the number of cache evictions. Furthermore, this metric is also affected by the cache replacement strategy. Interestingly, the relative performances of the replacement strategies differed depending on the distribution pattern: given a Zipfian distribution, *RR* outperforms the other approaches, whereas with a uniform distribution, *MDMR* produces the best results. *MDMR* was designed to maximise diversity (see Section 2.2.3) and was originally evaluated in a scenario with uniform request patterns [66]. As such, it is unsurprising to see it perform well in this scenario. Given a skewed request distribution, however, it is not able to outperform *LRU*. This is because caches in this scenario are already much more likely to hold more popular content, reducing the impact of cache-shaping strategies. The fact that *RR* performs better than the other approaches in the Zipfian scenario is surprising. It may be that there is still some thrashing at the tail of the Zipf distribution, where popularities even out, and that random replacement counteracts this effect.

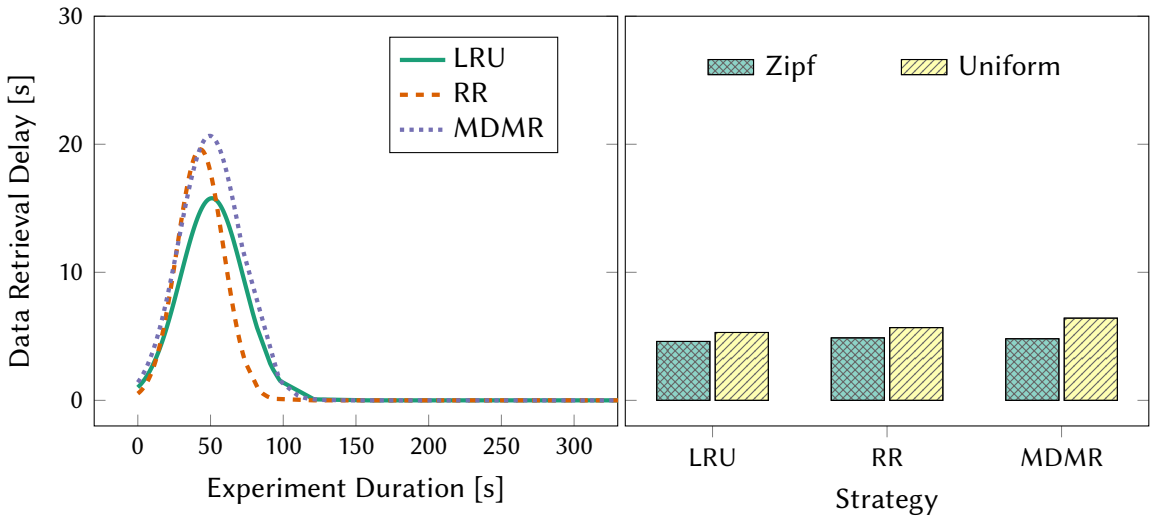


Figure 3.7.: Content delivery latency

Content Delivery Latency

Figure 3.7 shows the average content delivery latency as a time series using a uniform request pattern and averaged over the first quarter of the experiment using both request distributions. While the overall expectation for the average delivery latency is the same for all strategies — a peak at the beginning followed by a gradual decline — the strategies differ in both the height of the peak and the rate of decline. *LRU* exhibits a lower initial peak in the delivery latency, but converges as slowly as *MDMR*. *RR*'s latency peaks similarly to *MDMR* but declines much faster.

Since the caching decision strategy used is *CEE*, it can be assumed that caches fill up at the same rate on average. The reason *MDMR* exhibits a higher initial peak is most likely that content freshness, which is prioritised by *MDMR*, only becomes a significant factor after the experiment has run for a certain time and the ages of content objects begin to diverge. In other words, *MDMR* needs a minimum amount of time to become effective at making content available.

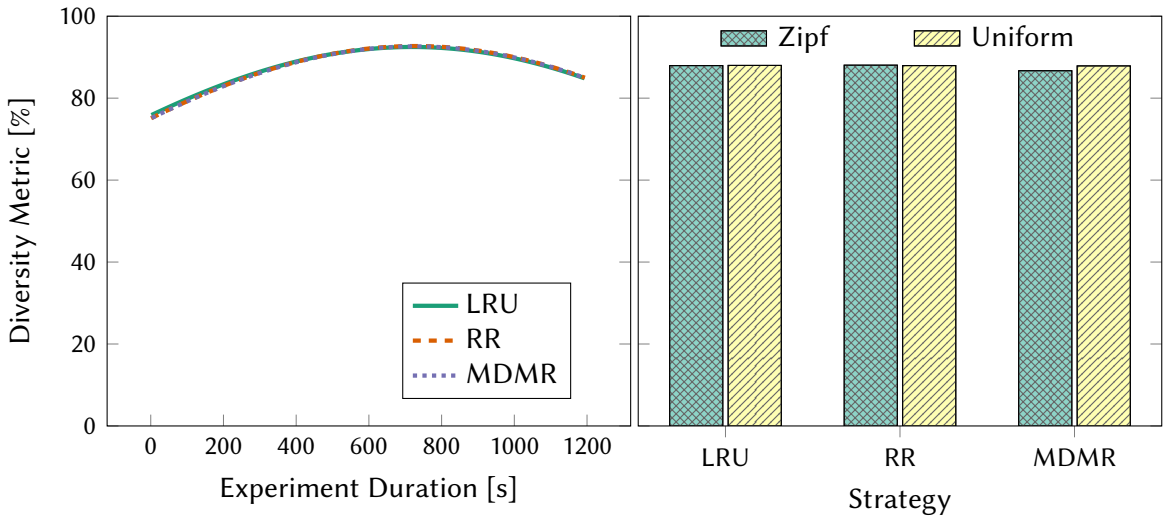


Figure 3.8.: Diversity metric

After a few minutes, the content delivery latency stabilises at a very low value, ensuring timely delivery during the majority of the network's lifespan. The significance of the early spikes in delivery latency is more in the strain that is put on individual devices, which becomes particularly important if the devices are battery-powered, putting data producers in greater danger of early failure.

Diversity Metric

It is immediately obvious from Figure 3.8 that the cache replacement policy seems to have no measurable impact on the diversity metric. All policies follow roughly the same curve described in Section 3.2.1, which exhibits consistently high diversity, but never reaches the peak diversity the probabilistic approaches do.

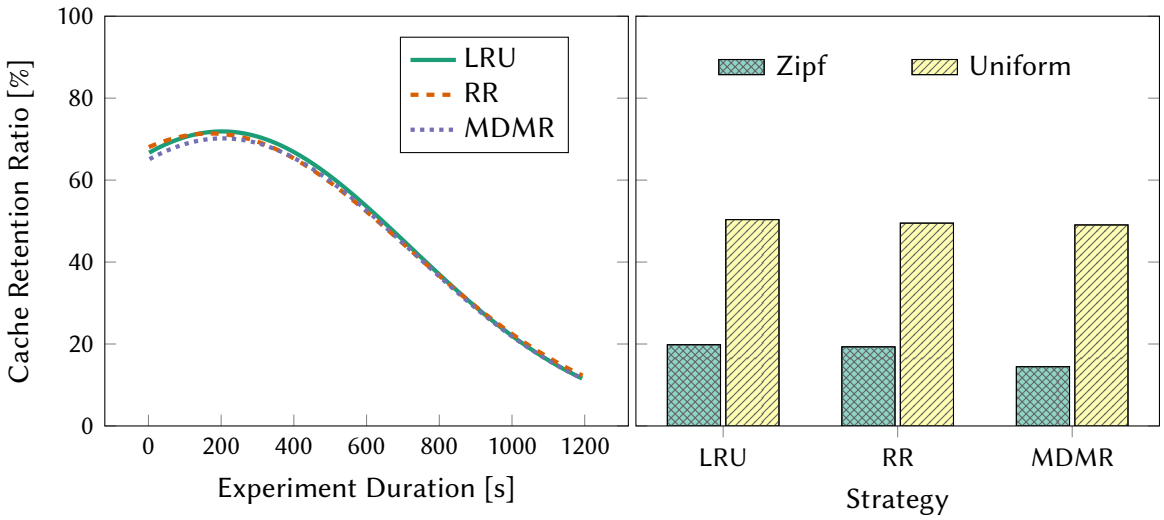


Figure 3.9.: Cache retention ratio

It is not necessarily intuitive that the cache replacement policy would have no impact on the diversity metric. After all, the replacement policy directly affects cache contents, and some policies, such as *MDMR*, make it their intended goal to maximise diversity. These results are thus slightly surprising. However, it needs to be noted that *MDMR* was explicitly intended to maximise diversity in a scenario in which nodes would periodically sleep [66], so it may not be best suited for this experiment setup.

Cache Retention Ratio

Figure 3.9 demonstrates that the CRRs for the cache replacement policies all follow the same pattern; what cache contents are replaced does not seem to have an impact on how long contents are retained in the network. As observed in Section 3.2.1, the right hand bar plot shows that content fades significantly quicker given a Zipfian distribution.

3.2.3 Summary of Performance Evaluation

This section presented a comparison and evaluation of different caching strategies for information-centric IoT. The results indicate that it is likely not worth implementing overly complex cache replacement strategies, as simple stateless policies can perform equally well or perhaps even better. This confirms that the findings published by Zhang *et al.* [162] for traditional ICN also hold true for information-centric IoT, which is very encouraging as it means that effective caching can be achieved even when using resource-constrained IoT devices.

Identifying the ideal caching decision strategy for an IoT application depends on the requirements of that particular application as well as the available resources. Some applications may produce highly homogeneous content that needs to be disseminated as rapidly as possible, while others may prioritise maximising the diversity of cached content over fast response times. Depending on whether the devices are battery-driven or have access to a constant power source, alleviating strain on single nodes may be more or less important. Therefore, offering a universal solution in the form of an ideal caching approach for all information-centric IoT deployments might not be possible. Instead, the research presented here should serve as a comprehensive overview of the benefits and shortcomings of the different approaches in relation to different metrics, not all of which may be of equal importance for any given application.

The results presented so far only scratch the surface of recent developments in caching in ICN in general and information-centric IoT in particular. Many more strategies than the ones evaluated here have been proposed, and research in this area remains very active. Section 3.3 will therefore examine a greater number of caching strategies, with a further evaluation focusing on the effects of network topology on content delivery latency.

3.3 Content Delivery Latency

This section presents an in-depth evaluation of a number of different in-network caching strategies in regard to latency and hop count reduction. As in Section 3.2, an experiment was designed using real IoT hardware in a physical testbed meant to emulate the conditions of a typical IoT application as closely as possible. Based on the results of the previous evaluation, cache replacement strategies — which were found to have little impact on caching performance — are ignored in favour of a larger selection of caching decision strategies (all of which are described in Section 2.2). Content delivery latencies are measured, as well as the average reduction in hop count between cached content and original storage location. Furthermore, the extent to which the effectiveness of a given strategy is influenced by the network's logical topology — and thus by the routing algorithm — is investigated.

The question of where in the network content should be cached is one of the most defining problems of in-network caching research [35, 41, 52, 116, 173]. In particular, it has not been definitively established whether it is better to cache content closer to the consumer or closer to the producer. Intuitively, caching closer to the consumer would seem to make more sense; after all, if content is kept close to the producer, the load on that particular node can be reduced, but the potential latency improvements seem to be minimal. If content is requested from a particular region of the network, it is probably safe to assume that it will be requested from that region again in the future. If the content was already cached close to that region, retrieving it will be much faster. This is the argument for caching towards the consumer and there are a number of caching strategies that employ this paradigm [55, 95, 116, 151]. However, as shown in Section 2.2.4, depending on the logical topology, there is also a strong argument for the inverse approach. The experiment presented in this section was designed to shed light on this dichotomy.

(a) Deployment of nodes in the IoT-LAB Grenoble site (b) IoT-LAB nodes in the Grenoble Senslab space

Figure 3.10.: The Grenoble site of the IoT-LAB testbed was used for experimental evaluation

3.3.1 Experiment Description

Given an information-centric IoT application where a reduction in content delivery latency is the first priority, the primary objective of this experiment is to compare and contrast the effect different approaches to ICN in-network caching have on this metric.

For this comparison, a series of experiments were run on the **FIT IoT-LAB** [3] open testbed, using the same IoT hardware and firmware as in Section 3.2. The *LRU* cache replacement policy is used in all experiments. As mentioned in Section 3.2, the choice of cache replacement policy has little to no impact on the performance of in-network caching.

The experiments were conducted on the *Grenoble* site[4] of the IoT-LAB testbed, which is larger than the *Lille* site used in the previous experiments. The site features more than 380 M3 nodes, which are distributed across the rooms and corridors of one floor of an office building (see Figure 3.10). 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 the behaviour of the network is very close to what might be expected in a real-world deployment.

[4] https://www.iot-lab.info/deployment/grenoble/

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. This ensures that the logical topology is different in each experiment run and also that the nodes will not be too strongly connected due to having a large number of one-hop neighbours. This is desirable as it allows us to study the effects of unreliable connections more closely. The transmission range of individual nodes is not enough to reach all other nodes in the building, so communication will be predominantly multihop. In a typical topology generated by this random selection of nodes, the mean path length is between 2 and 3 hops, as shown in Figure 3.1. This kind of multihop setup is commonly found in the industrial monitoring domain. Real-world deployments tend to have slightly longer average paths, but this scale is infeasible to achieve within the constraints of physical testbeds such as IoT-LAB.

For this experiment, 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) [60], leaving about 50 kB that have to be shared between the CCN-lite heap (comprising CS, FIB, and 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. Class 1 devices with RAM on the order of 10 kB [29] also need to be considered. 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 [21]. Depending on the nature of the data transmitted by the application, available cache space may thus be severely limited. This motivates the 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 provides full control over all node serial interfaces. An experiment begins with a brief (30 seconds) setup phase, in which every node advertises its own prefix (dictated by its address), which is then propagated through the rest of the network using **HoPP**'s [61] 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** [87] algorithm. The fact that the routing algorithm is based on Trickle also means that nodes' FIBs can be kept up to date during runtime. After setup is complete, every node will request a piece of content with a random ID in $\{0, \dots, 49\}$ from each of the prefixes in its FIB. Interest and Data packets are handled as specified by the 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 this 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 network's logical topology (i.e. the routes taken by Interest and Data packets) 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. When forwarding interests, nodes will always prefer the FIB entry with the lowest rank.

The caching decision strategies examined in this experiment are *CEE*, *LCD*, *Prob(p)*, *ProbCache*, *ProbCache-Inv*, *Labels*, and *Intervals* (see Section 2.2).

Relation between Hop Count and Latency

Figure 3.11 shows how the latency (specifically, the content delivery latency, which is the time between Interest generation and satisfaction) is affected by the number of hops taken to retrieve the content. This does not differentiate between cached content and content produced by the prefix owner, i.e. the hop count shown here is the number of hops traversed by the Data to the requester from either its original producer or from a caching node. This

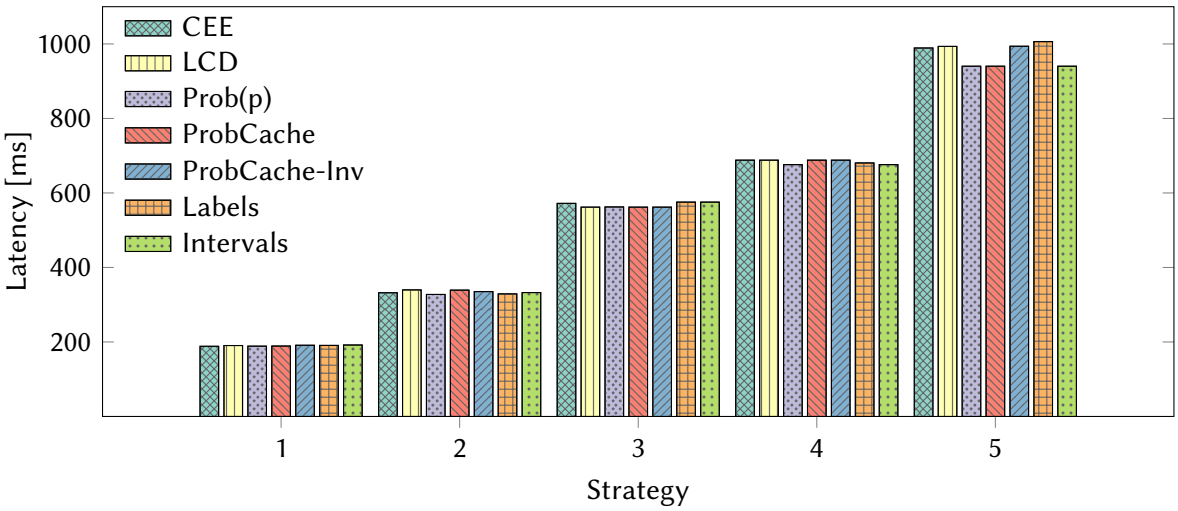


Figure 3.11.: Relation between hop count and latency

means that this figure shows only the correlation between hop count and latency for individual transfers, which is linear because nodes are evenly spaced in the network. The caching strategy does not have an impact on this metric in terms of packet travel time, because the caching strategy affects the hop count and not the actual per-hop transfer speed.

The only impact the caching strategy can have on this metric is the computational overhead as each hop on the Data path needs to make a decision. However, the difference appears negligible. Thus, it can already be concluded that the complexity of all caching strategies is within acceptable bounds, making them worth considering. For the rest of this section, this graphic mostly serves to illustrate the ground truth of what latency to expect depending on the hop count. It will be useful for contrasting with the metrics that will be discussed in the rest of this section, as it effectively also represents the expected latencies if in-network caching was entirely disabled.

Hop Count Reduction

For each Interest, the number of hops between its origin and the owner of the prefix it is requesting is denoted as the *distance to source*. In other words, this is the number of hops the Interest/Data packet would always have to travel if there were no caches in the network. This distance can then be compared to the actual number of hops taken by the Data packet on the way back. This measure is called *hops to hit* as it denotes the number of hops it actually took for the Interest to reach a cache hit. The more efficient a caching strategy, the more content will be available in a cache closer to the consumer, leading to a lower average hops to hit value. The difference between the distance to source and the hops to hit is denoted as the *hop count reduction*.

The hop count reduction is obviously closely related to the *cache hit ratio*. That metric was introduced in Section 3.1.1, but will not be elaborated on in the context of latency, as it is subsumed by the hop count reduction — any reduction in hop count implies a cache hit and vice versa. Similarly to the cache hit rate, the hop count reduction can be extended to be weighted by content size (commonly referred to as *footprint reduction* [148, 150]).

The top plot of Figure 3.12 shows the average hop count reduction for the different caching strategies at different distances. The first obvious effect is that most of the strategies only show a significant hop count reduction starting from a minimum distance to source. In all strategies except for *LCD*, there is a slight reduction at 3 hops and then a substantial one at 4 hops. The reason for this is that at shorter distances, there is less cache space between the producer and the consumer, which means fewer opportunities for content to be cached on the path. This makes it much more likely that a request will have to be routed all the way to the prefix owner to be satisfied. After a distance of 4 hops, the hops to hit will increase again as the distance to source increases. The "turning point" at which caching begins to have a noticeable impact seems to lie between 3 to 4 hops for most strategies. After this point, there is enough cache space on the path that content is likely to be found at a closer node. *LCD*, on the other hand, already shows a noticeable hop reduction at a distance of 2 hops, which gets even stronger as the distance increases.

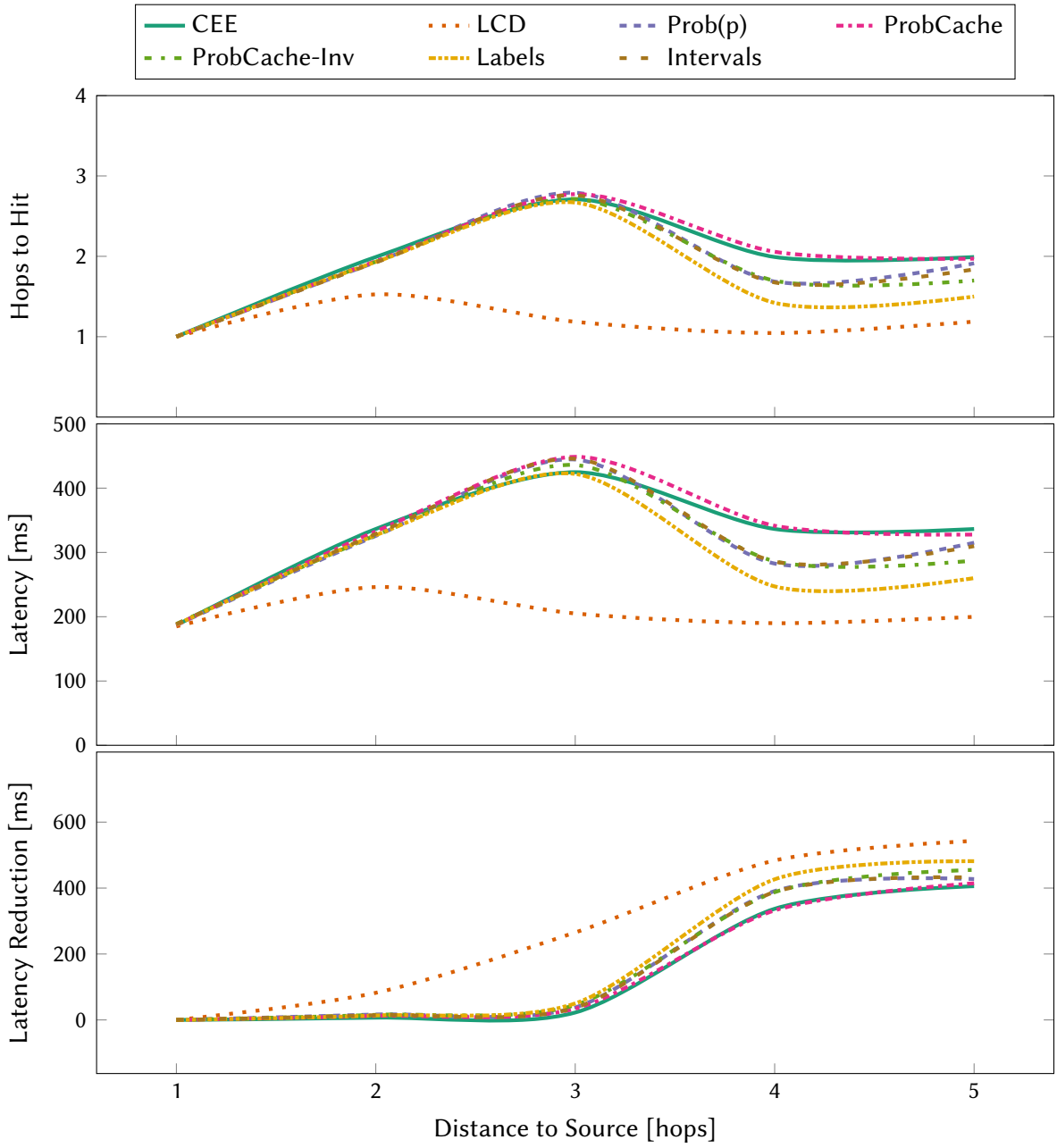


Figure 3.12.: Hop count, latency, and latency reduction by strategy. Note that while the graphs are plotted as continuous lines here, this is only for presentation purposes as it allows for better contrasting between the various strategies. The x-axis is discrete and the interpolation between points is purely visual.

Another strategy that stands out is *Labels*, which in terms of hop count reduction is second only to *LCD*. This strategy takes a very different approach from *LCD* in that it favours an even distribution of content by stratifying it according to content and node IDs [89]. It appears that this approach, which prioritises making individual pieces of content available in the same number of nodes across the network, results in a beneficial cache distribution.

It has been observed multiple times [35, 112, 116, 117] that *CEE* is not an optimal caching strategy for ICN, and this is supported by the results presented here. The reason is that *CEE* is vulnerable to *thrashing* effects (especially if the *LRU* replacement policy is used, which is almost always the case) when nodes are caching high volumes of diverse data. The limited size of caches in IoT only exacerbates this effect.

Topology Effects

The fact that *LCD* exhibits a significantly greater reduction in hop count points to a phenomenon introduced previously: the logical topology of the network has a significant impact on the effectiveness of the chosen caching strategy. This is explained in more detail in Section 2.2.4.

In ICN, the logical topology is a direct result of the forwarding paths stored in the nodes' FIBs. The FIBs codify how Interests are forwarded and thus how content is distributed across the network. Therefore, getting a sense of a network's logical topology requires knowledge about how its FIBs are constructed. There is no universal answer to this, because ICN enforces no standards for how FIBs are populated. However, in most cases, the contents of the FIBs are the direct result of the routing algorithm that is used by the producers to advertise their content. The way in which nodes learn about their neighbours' contents and in turn inform their own neighbours will dictate what their FIBs will look like. Ultimately, this means that the routing algorithm dictates the entire network's logical topology.

The HoPP/Trickle routing algorithm used in this evaluation makes the individual FIBs organise themselves into tree topologies based on rank [61]. This means that they resemble a core topology such as the one shown in Figure 2.6. Thus, the paths between consumers and producers are more likely to cross closer to the producer, making nodes closer to producers more

valuable caching candidates. The results in the top plot of Figure 3.12 show that a strategy like *LCD*, which always caches close to the producer, clearly benefits from this fact. It is also evident that *ProbCache-Inv*, which has increased caching probability near the producer, performs a little better than default *ProbCache* with increased probability towards the consumer, especially at higher distances to source, although this difference is not as pronounced.

Although *Labels* does not reach the same hop count reduction as *LCD* does, it should be noted that, unlike that strategy, it is agnostic of topology and thus should be similarly effective in an edge topology, where *LCD* would not be expected to reach the same performance.

Latency and Latency Reduction

The middle plot of Figure 3.12 shows the average content delivery latency in relation to the distance to source. It is immediately obvious that the latencies for different distances follow the same pattern as the hop reduction but are slightly more vertically stretched, which also follows from the measurements shown in Figure 3.11. Since the increase in latency with each hop is linear, the predominant change in latency is solely determined by the average hops to hit per distance.

Of course, the most relevant measure for this evaluation is the actual *reduction* in latency, i.e. how much faster on average content can be retrieved when using a given caching strategy. For this, the average latency of a given strategy by distance to source is compared to the average latency for that number of hops, i.e. the expected latency without caching (cf. Figure 3.11). The results are shown in the bottom plot of Figure 3.12. It is clear that the sweet spot in terms of latency reduction, i.e. the distance at which caching has the biggest impact, is found at a distance of 4 hops for all strategies except *LCD*, where it is 3. This was already implied by the middle plot of Figure 3.12, where a distance of 4 (3 for *LCD*) exhibited the first dip in hop count and thus latency, but when related to the expected latency it becomes even more pronounced.

Once again, *LCD* shows by far the best performance at the peak, whereas the other strategies are relatively close together. As might be expected, *Labels* is once again the strongest contender out of the remaining strategies, with *ProbCache-Inv* coming in third.

Overall Performance

Figure 3.13 shows an overview of the mean hops, the mean latencies, and the mean reduction in latency across all path lengths for the different strategies. Evidently, across all strategies, the mean hops needed for each Interest can be reduced to a little over 1. *CEE*, *ProbCache*, and *Prob(p)* perform the worst overall, whereas *LCD* and *Labels* perform the best — *Labels* has a slight advantage over the other strategies and *LCD* has a significant one. Although default *ProbCache* performs no better than *Prob(p)*, its inverted variant shows some improvement.

There are several conclusions to be drawn from this. The first is that as described above, due to the fact that the routing algorithm chosen for these experiments results in a core topology, caching near the producer is the more effective strategy, making *LCD* a good choice. From the results, it can also be seen that strategies that do not take elements of the topology into account (i.e. *CEE* and *Prob(p)*) generally do not perform as well as those that do, although *Labels* goes against that trend. The middle ground is covered by *ProbCache-Inv*, which does consider topology but still introduces an element of chance. A tentative conclusion from this is that deterministic, topology-based approaches provide better content placement than probabilistic ones.

3.3.2 Summary of Content Delivery Latency

This section presented a comparison and evaluation of several different caching strategies for information-centric IoT, focusing on their effects on content delivery latency. The results indicate that the logical topology of the network has an impact on which cache distribution is optimal, and thus it may be fruitful to use caching algorithms that are optimised for the given topology. However, caching strategies that ignore topology in favour of other approaches, such as stratifying the data evenly across the network, are also promising and may be preferable if the topology is unknown, mutable, or a hybrid between different types. The network's logical topology is influenced by the choice of routing algorithm, thereby creating a direct link between routing algorithm and caching strategy. In other words, a holistic caching solution for information-centric IoT should ideally take all of these aspects into account.

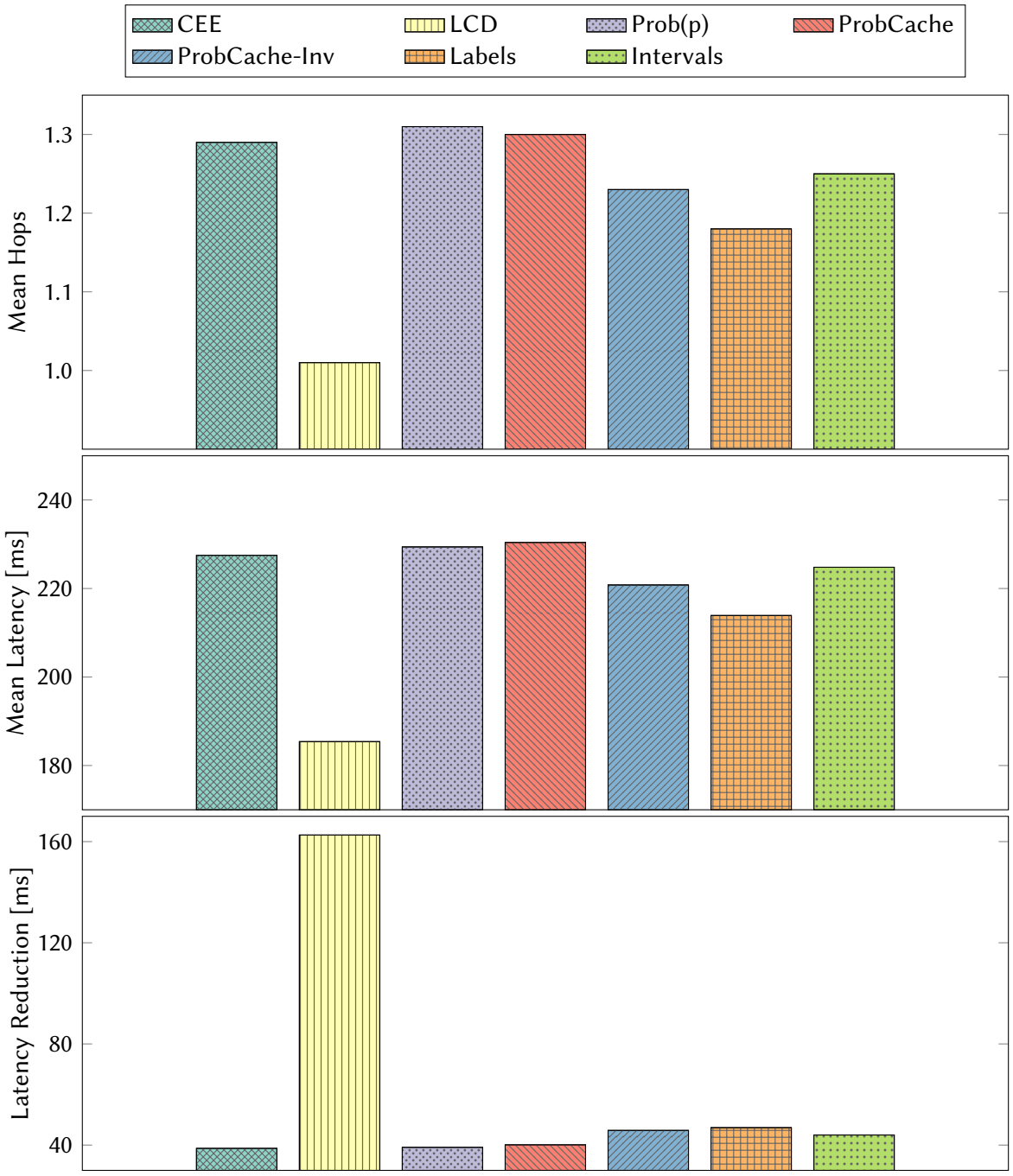


Figure 3.13.: Mean hops, mean latency, and latency reduction by strategy

Of course, determining the optimal caching strategy for a given IoT application also depends on the requirements of that particular application as well as the constraints imposed by the hardware. More computationally intensive caching strategies may yield better results, but may take too much resources away from the actual application. Overall, however, the results indicate that even simple caching strategies such as *LCD* or *Labels* can improve performance compared to indiscriminate caching, so they will almost always be worth considering. This is especially promising in light of the stated goal of keeping any proposed solution as lightweight as possible.

3.4 Conclusions

This chapter has presented a series of evaluations of the performance of various caching decision strategies and cache replacement policies for information-centric IoT, with a special focus on content delivery latency. The results indicate that the caching decision strategy has significantly more impact on caching performance than the cache replacement policy. For the latter, simple stateless policies were found to be sufficient in most cases.

While it is most likely impossible to determine a one-size-fits-all caching strategy that will guarantee the best performance in all scenarios, the results do indicate that strategies that are able to adapt to the network topology show a lot of promise. This is because the ideal location for a given in-network cache appears to be strongly influenced by the shape of the topology, which in turn is a result of the routing algorithm chosen to construct FIBs.

The direct link between the choice of routing algorithm and the effectiveness of the caching strategy is a phenomenon that had not been explicitly investigated in ICN caching research. Most research that compares caching strategies does not take the routing algorithm into account at all; often, FIBs are just assumed to be populated *a priori*. The lessons learnt from the presented research indicate that an in-depth comparison of caching strategy performance on topologies created by different routing algorithms should yield valuable insights and pave the way towards a holistic solution. Therefore, Chapter 4 will compare the performance of different caching strategies in core as well as edge topologies and present a lightweight caching solution designed to be effective regardless of topology.

CHAPTER 4

A Lightweight Centrality-Based Caching Strategy for Information-Centric IoT

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 efficient caching strategy in this regard is one that minimises the effective distance between the consumer and the content it needs. The aim of this chapter is to develop a strategy that can achieve this under the constraints imposed by IoT hardware.

Section 3.3 showed that caching heuristics that take logical 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. This chapter will discuss a pair of caching strategies for traditional ICN that aim to take advantage of topological effects to maximise the benefits of in-network caching. However, these strategies suffer from the issues described above: they require every device to have at least partial knowledge of the network, and they incur significant overheads. To address these shortcomings, this chapter then introduces a new content caching strategy called *Approximate Betweenness Centrality (ABC)* [114], which makes use of the topology-based heuristics of existing strategies, but requires no knowledge of the network and incurs no communications overhead. This new strategy is compared to

several existing ICN caching strategies and its effectiveness is evaluated using real IoT devices in a large physical testbed. The results show that this lightweight approach can deliver results that are comparable to those of more expensive strategies while incurring almost no additional costs.

4.1 Centrality-based Caching Strategies

Section 2.2.2 briefly introduced two caching strategies — *Betw* and *EgoBetw* [41] — which rely on the *betweenness centrality* measure [146, 154] in order to make a caching decision. This section will discuss these approaches in more detail.

Betweenness centrality 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:

$$C_B(v) = \sum_{i \neq v \neq j \in V} \frac{\sigma_{i,j}(v)}{\sigma_{i,j}}, \tag{4.1}$$

where $\sigma_{i,j}$ is the total number of paths between two nodes i and j, with $i \neq v \neq j$, 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, it can be assumed without loss of generality that the shortest path between i and j is always used as the content delivery path. Therefore, the definition can be simplified as:

$$C_B(v) = \sum_{i \neq v \neq j \in V} \sigma'_{i,j}(v), \tag{4.2}$$

where

$$\sigma'_{i,j}(v) = \begin{cases} 1, & \text{if } v \text{ on path } (i,j) \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

Algorithm 9 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
```

Betweenness centrality has been found to be a useful indicator of node importance in a network [146]. This can be applied 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.* [41]. There are other centrality measures apart from betweenness, such as *closeness centrality*, *degree centrality*, *eigenvector centrality*, and others, all of which could potentially be used in the same way as betweenness centrality is used here. Some of them have been evaluated in related contexts in ICN [75, 124, 153], but only betweenness centrality has been applied specifically to the caching decision strategy.

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 greater than or equal to the centrality value recorded in the Data packet. This mechanism is illustrated in Algorithm 9. 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 CS using `getData()`; and `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 chapter 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* [41].

Betw is a straightforward implementation of the betweenness centrality measure as described by Wasserman and Faust [154]. Before nodes begin exchanging Interests and Data, there is a setup phase in which all nodes are assigned a centrality value using Equation (4.2). 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 (*i*) communications (exchanging all of the neighbour lists), (*ii*) memory (storing neighbour information for the whole network), (*iii*) computational cost (converting the neighbour information into a centrality value), and (*iv*) 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 [53]. 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 Equation (4.2), 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 it is assumed that the centrality value is to be calculated entirely on the nodes without any *a priori* knowledge, each node needs to flood its own 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 network. If *Betw* is used 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 $\mathcal{O}\left(n^2\right)$. 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 $\mathcal{O}\left(n\right)$, 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 $\mathcal{O}\left(n^2\right)$ for *Betw* and in $\mathcal{O}\left(d^2\right)$ for *EgoBetw*, where d is that node's degree[1] ($d \leq n-1$).

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 $\mathcal{O}\left(n^2\right)$ for *Betw* and in $\mathcal{O}\left(d^2\right)$ 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.

[1] In graph theory, the degree of a vertex (node) is the number of edges (links) connected to it. Translated to wireless networking, this means that a node's degree is the number of one-hop neighbours it has.

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.

4.2 Approximate Betweenness Centrality

Given the fact that, as shown above, *Betw* and *EgoBetw* are difficult or even impossible to implement on IoT hardware, the aim of the research presented in this chapter 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. This approach is motivated in part by the fact that Rossi and Rossini [124] found that even the simplest centrality measure — *degree centrality* — was sufficient to achieve acceptable performance for their use case of allocating caching space, and that the more complex approaches offered little improvement. Therefore, it is reasonable to assume that a similarly simple approach may be promising as a caching decision strategy.

To that end, this chapter introduces a novel 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.

Algorithm 10 Centrality Approximation

```
1: function UPDATE_CENTRALITY(Source, Destination)
2:     if (Source, Destination) ∉ myPaths then
3:         myPaths ← myPaths + (Source, Destination)
4:         myCentrality ← myCentrality +1
5:     end if
6: end function
```

4.2.1 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. It is assumed 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, Interest packets are extended 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 4.2.2.

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, calculating these centrality values requires an *a priori* setup phase that is not needed in *ABC*.

Algorithm 10 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 9.

4.2.2 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 $\mathcal{O}(1)$.

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 $\mathcal{O}(p)$, where p is the number of paths in the network[2] ($p \leq n(n-1)$). 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 the evaluation will be discussed in Section 4.3.

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 $\mathcal{O}(1)$.

For ease of comparison, the overheads of *Betw*, *EgoBetw*, and *ABC* are shown in Table 4.1.

Section 4.1 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 compu-

[2] It is assumed 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.

Table 4.1.: 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.

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

tational 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 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 be adapted to accommodate dynamic topologies. This can be achieved by using time-outs for the information stored on recorded paths or by weighting path information based on recency.

One open question concerns the convergence time of *ABC* in case of dynamic topologies. In general, the approximate centrality values should converge to the actual values if there is enough path information, i.e. if the request frequency is high enough. However, a scenario is conceivable in which the topology is highly dynamic but the request frequency is low, meaning that the frequency of path updates is not high enough for centrality values to converge. One way to address this is with the aforementioned time-outs. If the time-out window is adapted to the request frequency (the lower the request frequency, the shorter the time-out window), then this scenario would, in the worst case, lead to the performance of *ABC* degrading to that of *CEE* as all nodes would have the same or similar centrality values. However, in such a scenario, this would in fact not be a drawback, as a low request frequency would mean that caches would be underutilised anyway, allowing us to simply cache at all nodes. In fact, requests in such a scenario may be critical, in which case caching everywhere would be an advantage.

Since this time-out mechanism would still only use 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 4.2.1, it was mentioned that *ABC* relies 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 with 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, in practice, the operation of *ABC* would not be significantly hindered by this as this group of nodes could simply be treated as a single producer for its path counting purposes.

Extending the information carried by the Interest with information on both source and destination is strictly speaking a break with ICN principles. Recall from Section 2.1 that ICN is based on treating named content objects instead of hosts as first class entities. This principle is somewhat weakened here by the reintroduction of source and destination pairs. However, the only purpose of these host identifiers is to facilitate centrality approximation and thus caching; the operation of the core ICN forwarding loop is otherwise unchanged. The break with ICN principles would thus only present a problem if the goal was to deploy *ABC* in the wider Internet, where the required information would not be available. However, *ABC* is designed specifically with the siloed environment of the IoT in mind, where devices are

assumed to be under the control of the same entity and adhering to the same protocol. This means that protocol breaks that are confined to the deployment are less of an issue. It must however be noted that this extension would not be able to interoperate with services relying on host 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.

4.3 Evaluation

This section presents a comparison of the *ABC* caching strategy with a number of other ICN caching strategies. Focus is placed on hop reduction and latency as the main performance metrics, since content delivery latency is typically the most important factor in time-critical IoT applications. Cache hit rate is also discussed, 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 main motivation for using *ABC* is the fact that it can be feasibly implemented and deployed in an IoT environment, which would not be possible for *Betw/EgoBetw* due to their *a priori* calculation requirements and significant overheads.

The goal of these experiments is therefore 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 much more realistic candidate for deployment on constrained devices.

4.3.1 Experiment Setup

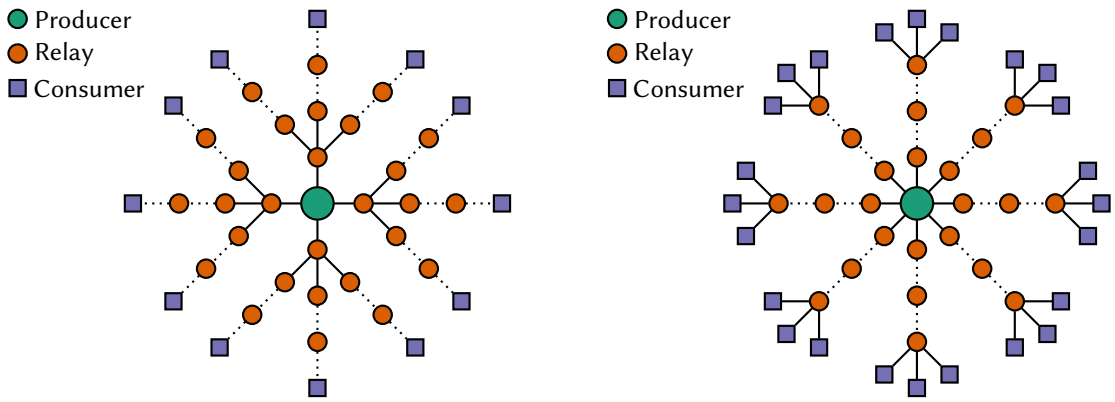
In order to compare *ABC* to the other strategies, a series of experiments were run on the *Grenoble* site[3] of the **FIT IoT-LAB** [3] open testbed. As in Section 3.3.1, the experiments were conducted on a random subset of 50 out of the 380 available nodes in the site, with a cache size of 5 content objects. The experiment setup is identical to the one used in that section, and details are described there.

As before, the experiment is managed by a control script using the IoT-LAB API. The API can execute shell commands on individual nodes, which is used to provide the *a priori* logical topology knowledge required by the *Betw*/*EgoBetw* algorithms. This makes it possible to circumvent the issues mentioned in Section 4.1, where it was 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, the experiment has access to a controller that has perfect knowledge of the entire network, making it possible 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.

4.3.2 Experiment Topologies

The evaluations were performed on two different network topologies: The *core* and *edge* topologies, as introduced in Section 2.2.4. For reference, the topology types are shown again in Figure 4.1.

[3] https://www.iot-lab.info/deployment/grenoble/



(a) Core topology (b) Edge topology

Figure 4.1.: Topology types (identical to figures shown in Section 2.2.4, repeated here for convenience)

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 [61] 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** [87] 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 resulting logical 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 resulting topology is a core topology.

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. The control script has access to nodes' physical locations through the IoT-LAB API. This means that it can use the nodes' neighbour information to construct an edge topology. The FIB entries for each content prefix are generated by the control script in such a way that delivery paths run directly from the producer to the most distant consumers. All other nodes are then connected to the most distant connected node in range. 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 has the effect that forwarding paths are more likely to converge at the edge, where the outermost nodes are found. The resulting topology therefore resembles an edge topology.

4.3.3 Experiment Description

After topology setup is complete, every node will request a piece of content with a random ID in $\{0, \dots, 49\}$ from each of the prefixes in its FIB (with 50 producers, there are thus 2500 distinct objects that can be requested). Requested content IDs follow a uniformly random distribution to model the typical request distribution found in IoT applications [94, 122]. Interest and Data packets are handled as specified by the 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 this 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. This information is used 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 4.1, two more strategies are included in the evaluation: *CEE* [73] and *LCD* [84] (see Section 2.2). *CEE* is included here to show how much there is to gain from employing a caching heuristic rather than simply caching all content. *LCD* is included in order 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 logical topology on the caching strategies, the visualisation of the results is divided by topology type.

4.3.4 Cache Hit Rate

Although content delivery latency will be the main focus of analysis for this chapter, 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. As in Section 3.1.1, the cache hit ratio R_{CH} is defined as:

$$R_{CH} = \frac{C_{cache}}{C}, \tag{4.4}$$

where C is the total number of content objects retrieved and C_{cache} is the number of content objects that are retrieved from the cache of an intermediate node that is not the prefix owner.

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.

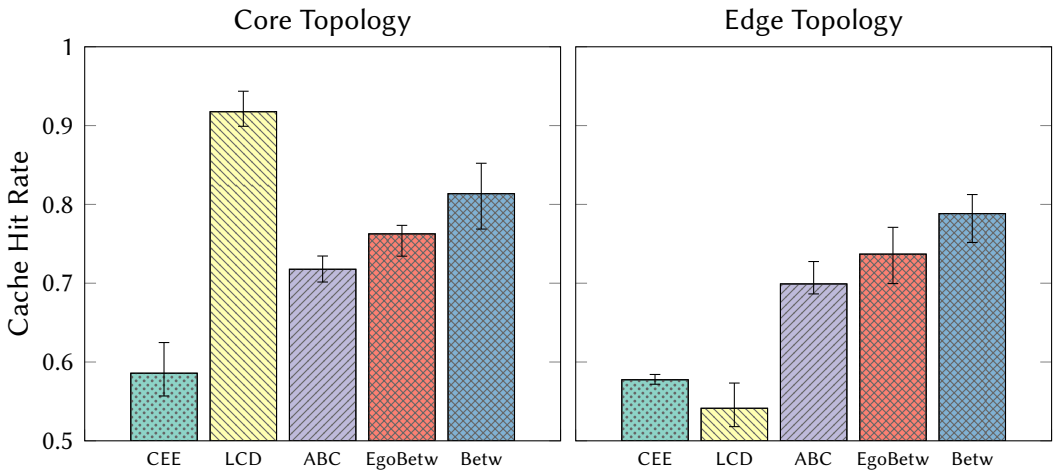


Figure 4.2.: Mean cache hit rate by strategy and topology

Figure 4.2 shows the mean cache hit rate for the different strategies in the two topology types. As has been shown in previous literature [35, 116, 117], *CEE*'s mean cache hit rate is lower than 60%, meaning that almost half of all content requests need to be routed to the original producer to be fulfilled. This is in contrast to the results reported in Figure 3.2, where *CEE*'s cache hit rate could reach around 80%. The difference is due to the fact that the experiments in this chapter were conducted with drastically reduced cache sizes (5 items instead of 20), which as expected exposed stronger performance differences between the strategies. 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.

4.3.5 Hop Count and Hops to Hit

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. This is contrasted with 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 mean hops to hit for the different strategies in the core and edge topologies are shown in the top row of Figure 4.3. For both topology types and all caching strategies, there is a measurable reduction in hops to hit in relation to the distance to source, which 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.

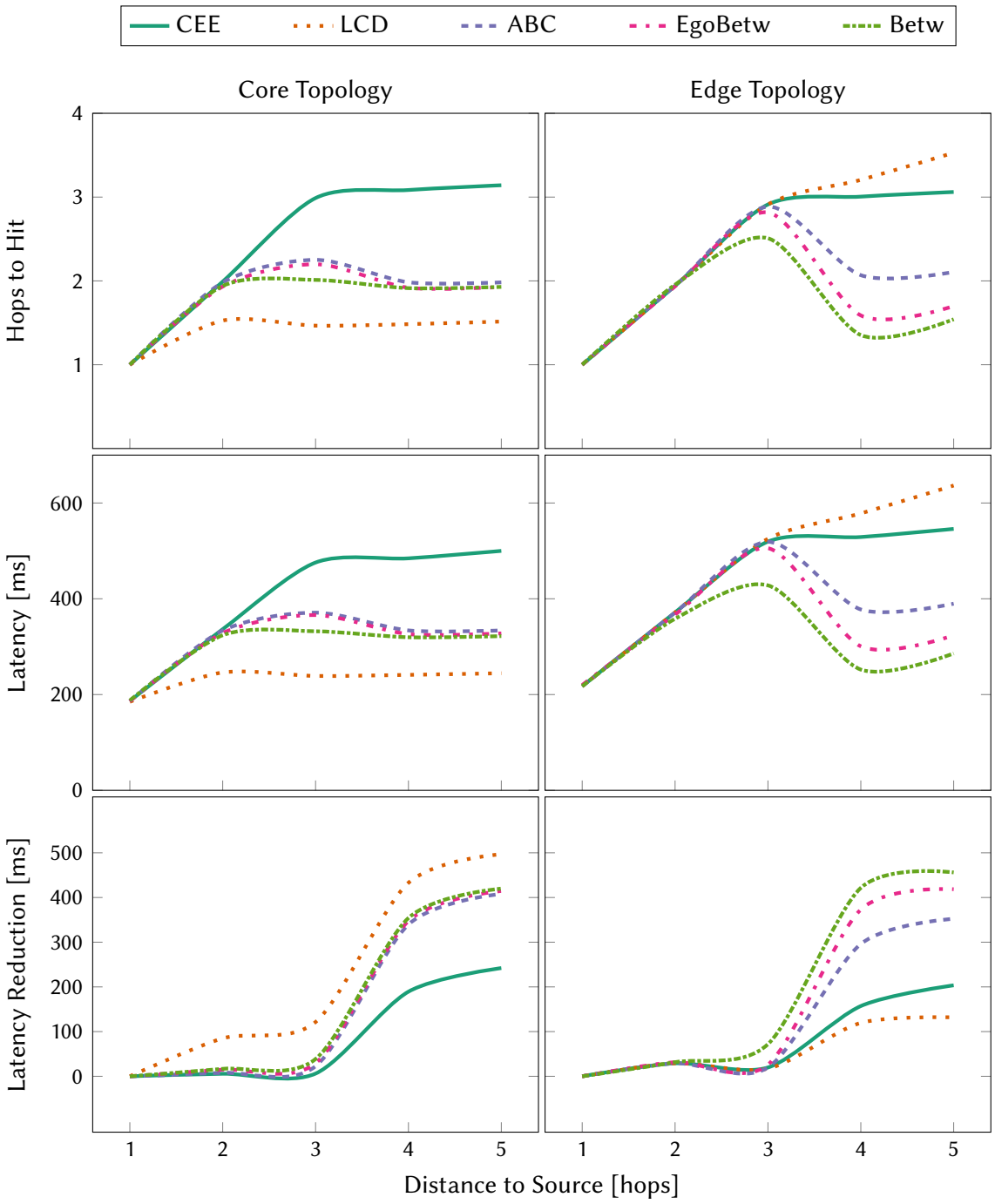


Figure 4.3.: Performance comparison: Core vs. edge topology. Note that the interpolation between discrete data points is purely for presentation reasons, compare Figure 3.12.

As with the cache hit rate as shown in Section 4.3.4, it appears 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. 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 reduction in hop count. 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 are achieved by the centrality-based schemes. Interestingly enough, their hops-to-hit value is even lower at a distance of 4 hops to the producer than it is at a distance of 3 hops, particularly in the edge topology. This observation can be attributed 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. 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, *Betw* does indeed boast the strongest performance, closely followed by *EgoBetw*, and *ABC* slightly behind the two. This follows the observations in Section 4.3.4 and is to be expected; after all, *Betw* has knowledge of the entire network's logical 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 ad-

ditional 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*. Unlike *Betw* and *EgoBetw*, it was actually possible to implement and deploy *ABC* in an IoT environment, making it a very promising choice.

4.3.6 Hop Reduction Ratio, Latency, and Latency Reduction

The previous section discussed the hops-to-hit metric in close relation to the distance to source. This relation can be formalised into a new metric for easier discussion. The difference between the distance to source and the hops to hit is called the *hop count reduction*, and the *hop reduction ratio* is the ratio between the hop count reduction and the distance to source. For a single content delivery operation c, the hop reduction ratio is thus defined as:

$$HRR_c = \frac{to\_source_c - to\_hit_c}{to\_source_c}, \tag{4.5}$$

where to_source_c is the *distance to source* between the prefix owner of c and the consumer that requested it and to_hit_c is the *hops to hit* the content chunk c in a cache. Thus, the more hops a delivery path is reduced by (i.e. the lower to_hit_c), the higher HRR_c for that content delivery operation.

Since the hops to hit and the hop reduction ratio are closely linked to the actual *content delivery latency*, we need to examine that metric next. Furthermore, analogous to the hop reduction ratio, we can define the *latency reduction ratio* which described the reduction in latency compared to the expected latency without any in-network caching.

The middle row of Figure 4.3 shows 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. Nevertheless, the results 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 4.3.1, *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. This source of latency is avoided in this evaluation in order to compare the strategies in the most favourable terms.

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. Similar to the hop reduction, The latency reduction is calculated 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 the bottom row of Figure 4.3, is the *latency reduction* of each caching strategy. Following directly from the hop count and latency results shown in the top and middle rows, the biggest gains in performance can be seen at a distance to source of 3 to 4 hops. As shown in Section 4.3.5, *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. This is a very encouraging result and shows that it is indeed possible to achieve satisfactory performance with a lightweight approach.

4.3.7 Cache Access Factor

While it is easy to interpret at a glance, the cache hit rate only paints a partial picture of content accessibility in the network, as it contains no information about how accessible the caches are. For example, LCD or another caching strategy that keeps contents close to the core might in theory have a very high cache hit ratio while not significantly reducing path lengths. The hop reduction ratio introduced above provides a more detailed understanding of this behaviour, but can be quite complex if the caching strategy behaves differently depending on path length. A single metric that combines both aspects would be desirable. To that end, the *cache access factor* F_{CA} is proposed here, which takes both cache hit and hop reduction ratio into account to produce a single metric that weights pure content accessibility with the average reduction of delivery paths.

The cache access factor is defined as:

$$F_{CA} = R_{CH} \cdot \frac{\sum_{i=0}^{n} HRR_i}{n}, \tag{4.6}$$

where R_{CH} is the cache hit rate, n is the number of content objects delivered, and HRR_i is the *hop reduction ratio* for a content object i as defined in Equation (4.5).

In effect, F_{CA} condenses into a single metric the expected gains in cache utilisation of a given caching strategy. In a deployment without caching, F_{CA} is 0 because R_{CH} is 0. Maximising F_{CA} requires both maximising R_{CH} as well as minimising the hops to hit. A high cache hit rate alone does not suffice if path lengths are not significantly reduced.

Figure 4.4 shows the cache access factor for the different strategies in the two topology types. Given what was shown in the two previous sections, these results should not come as a big surprise given that they are a synthesis of the previous metrics. However, it is evident how the combination of cache hit rate and hop reduction helps bring some more nuance to the differences between the strategies — LCD, for example, manages to outclass the other strategies in the core topology further than it did in either of the individual metrics thanks to its strong results in both. By contrast, in the edge topology, its relative weakness compared to CEE in terms of cache hit rate is mitigated by its very similar performance in terms of hop

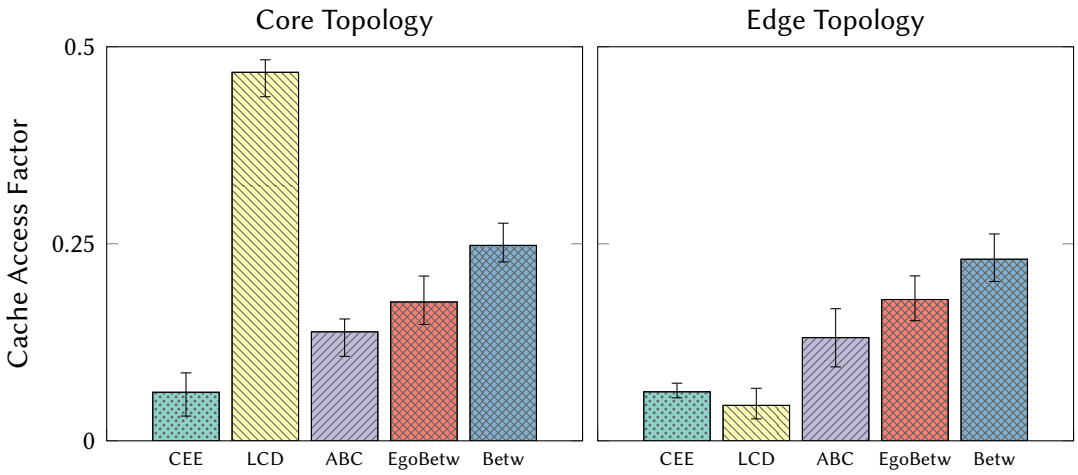


Figure 4.4.: Cache Access Factor by strategy and topology

reduction, where for paths of up to 3 hops in length (which make up a plurality of content requests), the two strategies perform identically. The relative differences between the three centrality-based strategies remain the same, with ABC being within a reasonable distance of the more complex strategies — the difference in cache access factor between it and *Betw* is around 0.1 for both topology types.

4.3.8 Memory Use of *ABC*

In Section 4.2.2 it was 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. During this evaluation, there were only a few nodes with high centrality values (core nodes in the core topology and nodes connected to 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.

4.3.9 Convergence Time of *ABC*

As mentioned in Section 4.2, *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 the experiments performed here, 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 real centrality value C_{real} (obtained by running the centrality calculation of *Betw* on the controller) such that $\left|C_{real} - C_{app}\right| \leq 2$.

4.3.10 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 4.2.2, 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.

4.4 Conclusions

This chapter 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. 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.

If the network's logical 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 has not been addressed yet 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 (see also Section 2.2.5). 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.

A related drawback of using betweenness centrality for the caching decision is the fact that this approach does not consider the *transitive importance* of nodes, i.e. nodes that are directly connected to very central nodes but do not have a high betweenness centrality value themselves. These nodes should perhaps also be considered as caching targets. An advanced centrality measure that takes nodes' transitive importance into account is the *eigenvector*

centrality [27]. While this centrality measure would likely be too complex to implement on limited hardware, the notion that nodes directly connected to central nodes have transitive importance is nevertheless useful and can be applied to a potential solution to the load balancing problem.

Although there was no immediate evidence for load-related performance degradation in the experiments presented in this chapter, a more detailed study of the relative load placed on different nodes when using centrality-based caching would shed more light on this issue and could suggest reasonable upper bounds for how much central nodes should be preferred before the risks outweigh the benefits. This study will be presented in the next chapter.

CHAPTER 5

A Lightweight and Modular Off-Path Caching Extension for In-Network Caching Strategies

The last chapter ended with the conclusion that while *Approximate Betweenness Centrality (ABC)* is certainly promising, there is one potential caveat in the form of load imbalance that needs to be addressed.

Because the nodes that are the most central in the topology are most likely to be selected to cache content, they will experience more strain from both cache read and write operations than less central nodes. However, the fact that they are more central means that they are already subject to more strain even without the caching load, since centrality implies that they lie on more paths between pairs of nodes, meaning that they have to relay more Interest and Data packets passing through them.

This twofold strain — in the shape of both communications as well as memory access operations — may result in detrimental side effects. The nodes' performance may suffer as they receive too many requests at once, leading to dropped packets as transmission buffers overflow. Furthermore, if the nodes are battery-powered, the most central nodes will run out of power earlier than others, which may negatively affect the performance of the network as a whole.

One potential way to counteract this effect is to make use of an *Off-Path Caching (OPC)* approach (see Section 2.2.6), where instead of always caching content strictly on the return path of the Data packet, it is possible to offload content to other nodes in the network as needed. As described in Section 2.2.6, OPC can take the form of a complete caching strategy or it can be a supplement to an existing strategy. This chapter will focus on the latter by developing an extension of *ABC* that makes use of OPC to counteract *ABC*'s side effects while not changing its fundamental centrality-based approach.

This chapter consists of a series of studies performed in order to first understand the impact of load imbalance and then to counteract these effects. First, a holistic definition of "load" is introduced, broken down into individual metrics that can be applied to quantify the strain a given node may be experiencing. Then, an experiment is presented that quantifies the effect of purely centrality-based caching by measuring the above metrics in relation to node centrality. Then, a modular extension of the *ABC* caching algorithm called *ABC+OPC* is introduced, which builds an OPC mechanism on top of *ABC*. This extension does not change the fundamentals of how *ABC* reaches its caching decision. Rather, it extends the cache replacement policy to move unused (i.e. less popular) items to adjacent nodes in order to make use of underutilised cache space. Finally, another experiment is conducted that validates the load improvements of *ABC+OPC* using the same load metrics as before while showing that overall performance in terms of content delivery is not negatively affected.

5.1 Towards a Holistic Definition of Load

Although it is sometimes treated as such, load is not a monolithic phenomenon that can be expressed with a single value. Rather, load is an emergent property that can be observed when some combination of factors is present. The circumstances leading to states that would be described as load can be varied and depend heavily on context. This section introduces the metrics that are most relevant for determining load in an ICN-IoT context.

5.1.1 Load on ICN Data Structures

In ICN, the network layer load on individual nodes can be expressed by measuring the read and write load experienced by the data structures responsible for ICN operations. While the Forwarding Information Base (FIB) contents are typically fairly static during normal operation and are almost exclusively read and not written, the Pending Interest Table (PIT) and the Content Store (CS) are subject to a large number of read and write operations, the frequency of which can be measured to characterise the node's load. In particular, a node handling a large number of Interests will exhibit high PIT load, while a node that is under high caching load will be subject to a large number of CS operations. From this distinction, it can already be seen that different nodes in the network will be experiencing different kinds of load (i.e., the nodes experiencing high PIT load will likely not be the same as the ones experiencing high CS load). The following sections will show that this observation holds for all load metrics.

5.1.2 Thrashing

Thrashing occurs when the contents of the CS are being frequently replaced and the average time a piece of content stays in the CS is low. While this can be caused by insufficient cache space or inadequate caching or cache replacement policies (see Sections 2.2.2 and 2.2.3), it is also an indicator of load, as the frequency of incoming contents correlates with the rate of thrashing. We can measure thrashing directly by observing how frequently content is replaced in a given CS, as well as indirectly by observing the average age of content objects in a given CS. It needs to be noted that it should not be the aim of any proposed solution to *minimise* this metric, as some rate of replacement is desirable in order to keep cached contents up to date and make use of the available caching resources. Rather, the replacement rate should be kept within reasonable bounds and not exceed a (scenario-dependent) threshold.

5.1.3 Memory Load

As described extensively in previous chapters of this thesis, the IoT nodes that are the target platform for the research presented here are characterised by extremely limited memory in the order of tens of kilobytes. Therefore, it is a very common occurrence for nodes to run out of memory for operation. In ICN, every operation (i.e., receiving, storing, and forwarding Interest or Data packets) incurs a temporary memory cost. It is possible to ensure a baseline of functionality by reserving a constant amount of memory for certain constant-size resources such as packet queues or the CS. However, this is implementation-dependent and not necessarily guaranteed. The temporary memory costs of ICN transactions, on the other hand, depend entirely on the rate at which the node receives Interest and Data packets. In CCN-lite, every received packet results in the allocation of a temporary data structure to hold the received prefix, which lives until the packet has been fully processed. In addition, every send operation incurs temporary allocation of a packet buffer. Therefore, if a node attempts to carry out too many send and/or receive operations simultaneously, it may run out of space to allocate the required data structures, resulting in failed transactions. Depending on which phase of the core ICN transaction loop fails, a failed transaction can have several adverse effects, including but not limited to (*i*) failing to create a new PIT entry, leading to unsatisfied Interests down the line; (*ii*) failing to cache incoming content, leading to decreased content delivery performance; or (*iii*) dropping incoming or outgoing packets, necessitating retransmissions. All of these situations should be avoided if possible. It should be noted, however, that these operations fail "gracefully", i.e., they do not lead to memory leaks and only result in individual operations being aborted. This also means that there are no lasting effects from such memory failures. Should the request frequency drop again, the node can resume normal operation.

5.1.4 Energy Consumption

Measuring energy consumption can be a reliable indicator for how much load a given node is experiencing. It can indicate either increased computational strain, meaning the node is performing more calculations that drain its energy, or network load, meaning the node is sending and/or receiving more packets, causing higher power usage by the transceiver. However, while increased power consumption generally implies higher load, constant power consumption does not necessarily indicate constant load. A node may experience load on its constant-size resources, such as the CS, packet queues, or RAM (see above), all of which can result in dropped packets or aborted operations and thus performance degradation, without exhibiting higher power consumption, because these overloaded states do not increase computational complexity or result in more transmissions.

5.2 Load Characteristics of Centrality-Based Caching

Since the negative side effects of centrality-based caching were not readily apparent in the experiments performed in Chapter 4, it is necessary to perform some further experiments in order to quantify them. To that end, an experiment is conducted in this section that examines the performance of *ABC* in respect to the metrics introduced in Section 5.1.

5.2.1 Experiment Setup

As in the previous experiments, the *Grenoble* site of the **FIT IoT-LAB** [3] open testbed is used. The experiment parameters are identical to those described in Section 3.3.1, except that content requests follow a Zipfian [32] rather than a uniformly random distribution in order to model a popularity distribution where some pieces of content are more popular (i.e., more likely to be requested) than others. This is because the negative side effects of centrality

caching are more easily observed if the content request distribution is not uniform, since this will result in less popular contents being starved as the most central nodes are occupied solely by a small number of popular content objects. The OPC solution that will be presented later in this chapter aims to counteract these effects.

5.2.2 Experiment Topology

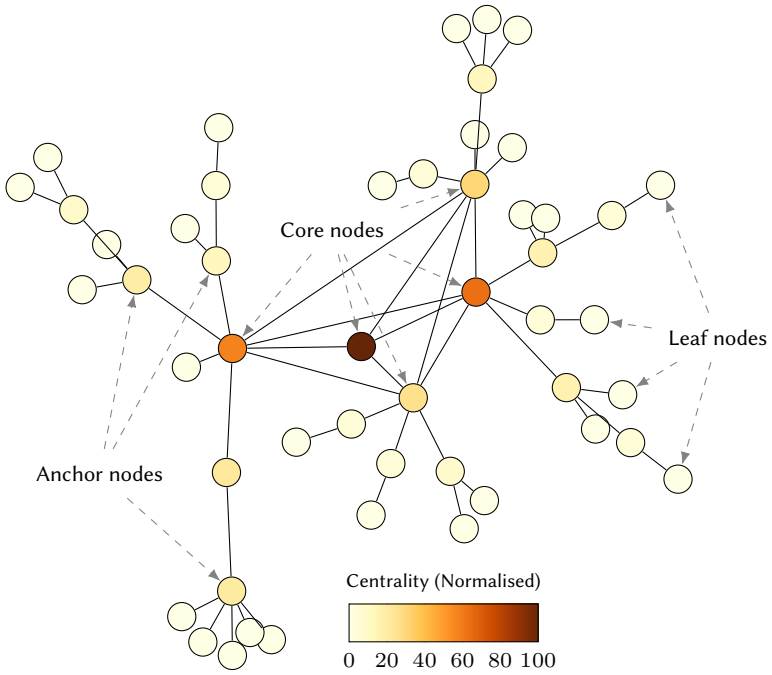


Figure 5.1.: Node centralities

The topology used for these experiments was designed to show the impact of centrality-based caching on the metrics introduced in Section 5.1. It features a central core with a small number of highly connected (meshed) nodes, as well as several branches going out of that core towards a large number of leaf nodes.

Figure 5.1 shows the logical topology of the experiment setup along with the normalised centralities of each node. For ease of reference, the nodes in the topology are roughly divided into three categories: *Core*, *Anchor*, and *Leaf* nodes, examples of which are labelled in Figure 5.1. Core nodes are the fully connected nodes in the centre of the topology. They have the highest centrality values of all the nodes in the network. Leaf nodes are the nodes with only one neighbour, situated at the edge of the network. They have a centrality of 0. The nodes connecting one or more leaf nodes or small sub-trees to the rest of the network are called anchor nodes. These three categories are not intended to be all-encompassing. There are a few nodes that do not neatly fall into a single category and are somewhere between core and anchor nodes. However, for the sake of the metrics discussed in this section, these three categories should be sufficient to distinguish between nodes' overall roles in the topology.

The experiment topology is created in IoT-LAB by explicitly populating nodes' FIBs to create the edges. FIBs are bootstrapped by randomly selecting five nodes in the topology to serve as producers. To set up the topology, it is irrelevant which nodes are selected as producers. The IoT-LAB control script simply determines the shortest path between any consumer and any producer using the *a priori* topology definition, generates the FIB entries accordingly, and saves them in the nodes. This way, a new set of producers can be randomly selected for each experiment run while keeping the edges of the topology constant between runs.

After topology setup is complete, every node will request a piece of content with a random ID in $\{0, \dots, 49\}$ from one of the five prefixes in its FIB every second (with some jitter applied). Requested content IDs follow a Zipfian distribution as explained in Section 5.2.1. Interest and Data packets are handled as specified by the NDN standard. Content is cached according to the selected caching strategy (see below). 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 this experiment) and sends it back towards the consumer.

To properly demonstrate where centrality-based caching can run into load issues, it is necessary to compare it to another caching strategy that would not exhibit these effects due to having different caching characteristics. While any of the caching strategies discussed in previous chapters would be viable candidates for a comparison, *LCD* [84, 162] was chosen as a benchmark, because it is another strategy that is strongly affected by topology (as shown

in Chapter 3), but is not centrality-based and is therefore expected to exhibit different load characteristics. Other strategies, such as the various members of the probabilistic family discussed in Chapter 3, are suboptimal candidates for this particular comparison because they tend to spread cache contents much more evenly across the network. While this in and of itself would be an advantage these strategies have over *ABC*, Chapter 4 showed that *ABC* tends to outperform them, whereas *LCD*'s performance is often comparable to *ABC*. These two factors make *LCD* the all around best candidate for a benchmark caching strategy.

Alternative Experiment Scenarios

A number of different experiment scenarios were considered and explored before settling on the one described above. Other variations included having only a single producer (either randomly selected or always in the centre of the network) as well as slightly less connected topology that did not have the full mesh between the core nodes shown in Figure 5.1 and instead all paths passing through the core would pass through the same node. Ultimately, the topology shown above was chosen as the most representative, as a topology where all connections run through one central node would make it harder to attribute performance losses to load issues instead of just poor topology design. Furthermore, choosing multiple producers instead of only one was more conducive to bringing the load issues to light. Nevertheless, for the sake of completeness and additional comparison, some results of the load characteristics experiments with these alternative topologies are presented in Appendix A.

5.2.3 Results

The experiment setup described above was used to perform a series of experiments in order to characterise the load characteristics of centrality-based caching. These results will then be used later in this chapter as a baseline to compare against an Off-Path Caching (OPC) approach, which aims to reduce load across all the relevant metrics. The results of the characterisation experiments are presented here.

Load on ICN Data Structures

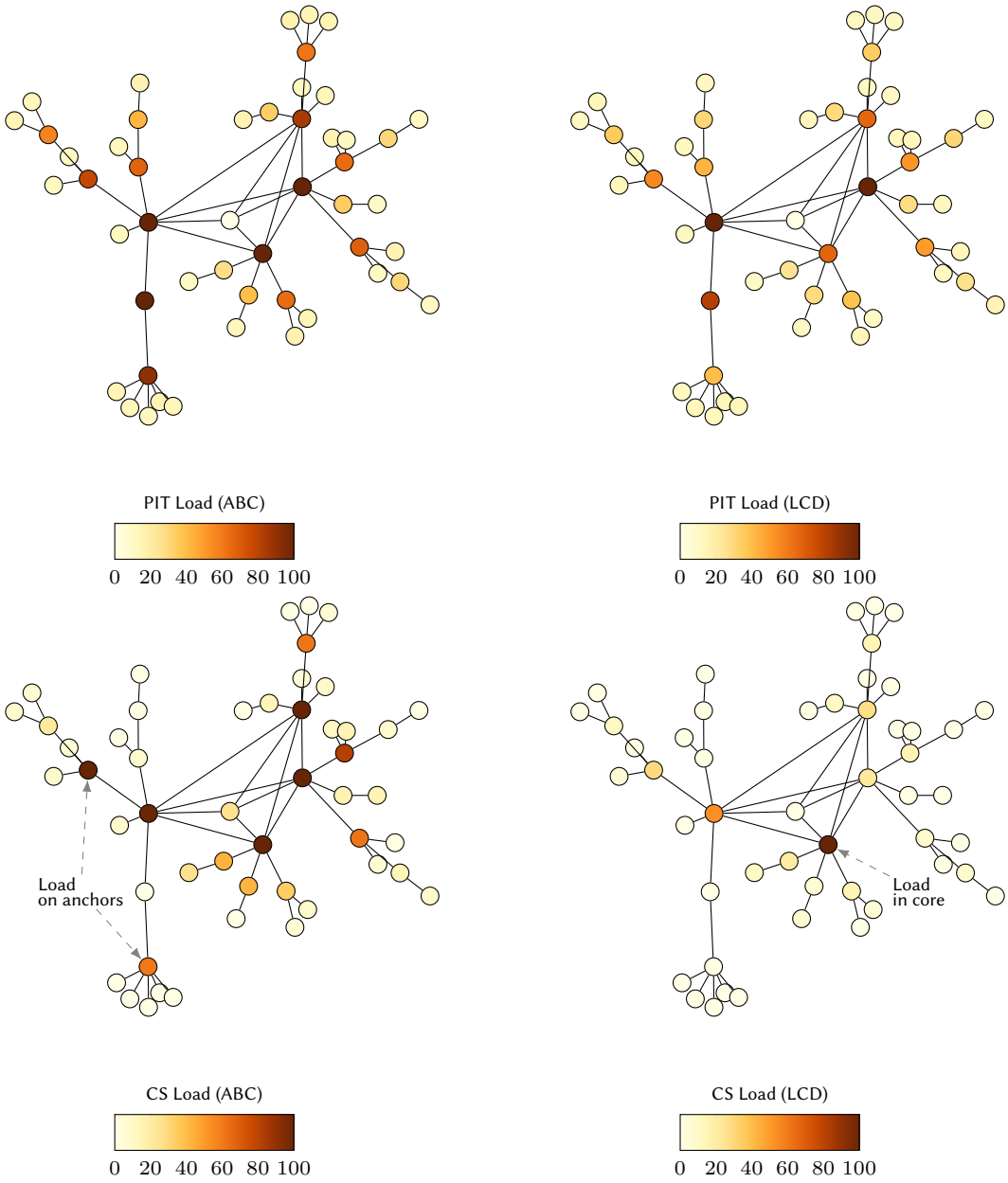


Figure 5.2.: Load on ICN data structures: *ABC* vs. *LCD*

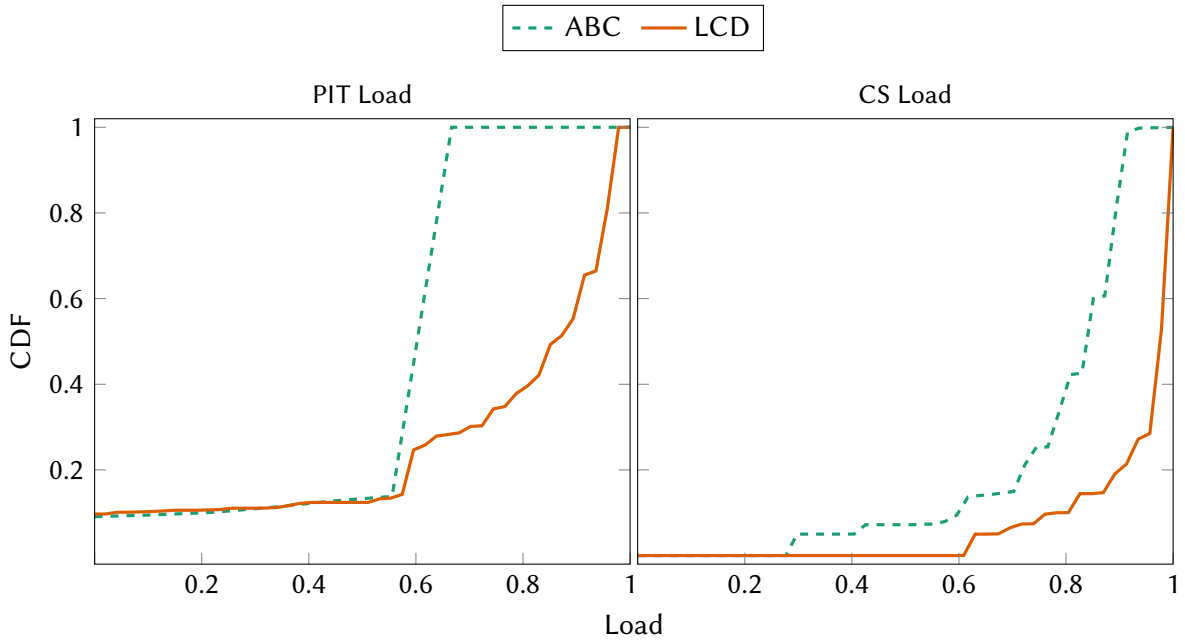


Figure 5.3.: Load on ICN data structures (CDF)

Figure 5.2 shows the load placed on the ICN data structures of each node relative to its position in the topology, while Figure 5.3 shows the cumulative load distribution. PIT load measures the average number of unsatisfied (waiting) Interests in a node's PIT at any given time, while CS load measures the frequency of read and write operations on a node's CS. Both measures are normalised across both strategies in the visualisation to show relative load.

It is immediately obvious from Figure 5.2 that while both load metrics correlate broadly with node centrality, the impact on the two data structures differs slightly depending on the node's role in the topology. In *LCD*, PIT load is distributed slightly more evenly across the network, with each non-leaf node holding some amount of PIT entries at any given time. We can see a distinct cutoff point at which PIT load increases, both in the topological view and especially in Figure 5.3. Here, we can also see the stark difference between *LCD* and *ABC*, which has an almost binary delineation between low-load and high-load nodes. CS load, by contrast, is concentrated in a smaller number of (mostly core) nodes in both strategies and although overall load is slightly higher in *ABC*, it follows the same distribution.

The difference in load distribution between the strategies can be explained using both the caching approaches of *ABC* and *LCD* as well as the different roles nodes play in terms of PIT and CS load. In *ABC*, data is only ever cached at the node with the highest centrality value along the delivery path. This means that the core nodes carry the highest caching load, and leaf nodes carry virtually none (the only CS load at leaf nodes comes from storing content they themselves have requested and subsequently received). *LCD*, on the other hand, gradually pushes content out towards the edges the more it is requested, which alleviates the load on all but the most central of nodes (these core nodes still experience load simply because of the number of requests they handle due to their central position in the network.) In contrast, when looking at the anchor nodes, we can see how differently content disseminates in a centrality-based strategy. Once the core nodes begin storing content, requests that travel from the edge to the core nodes will result in content being stored on the most central of the intermediate nodes between edge and core — in other words, the anchor nodes that connect several leaf nodes or small sub-trees to the core. Nodes between anchor and core nodes, with lower centrality, are essentially ignored by *ABC*. We can see that anchor nodes experience a level of CS load that is about halfway between that of the core and the leaf nodes, while their load under *LCD* is much closer to leaf nodes since content takes almost the same time to reach them.

Contrast this observation with the PIT load experienced by anchor nodes in both strategies. There is a much clearer distinction between leaf nodes on the one hand and core and anchor nodes on the other. This is because anchor nodes have the crucial role of aggregating the requests coming in from their respective leaf nodes. In *ABC*, this role falls to the small number of high-centrality nodes, while *LCD* smooths out the distribution. This creates the stark difference in PIT load distribution.

It is evident that there is indeed an uneven distribution of load on the ICN data structures of individual nodes, with more central nodes handling a much larger proportion of the requests. While this effect is present in both *LCD* and *ABC* to some extent, particularly the uneven PIT load is much more apparent in the latter. Although this observation alone does not necessarily imply a loss of performance — and the experiments in Chapter 4 did not directly observe any such effects — it is still an issue that may cause problems in critical applications and could easily be exacerbated in situations with high traffic load.

Thrashing

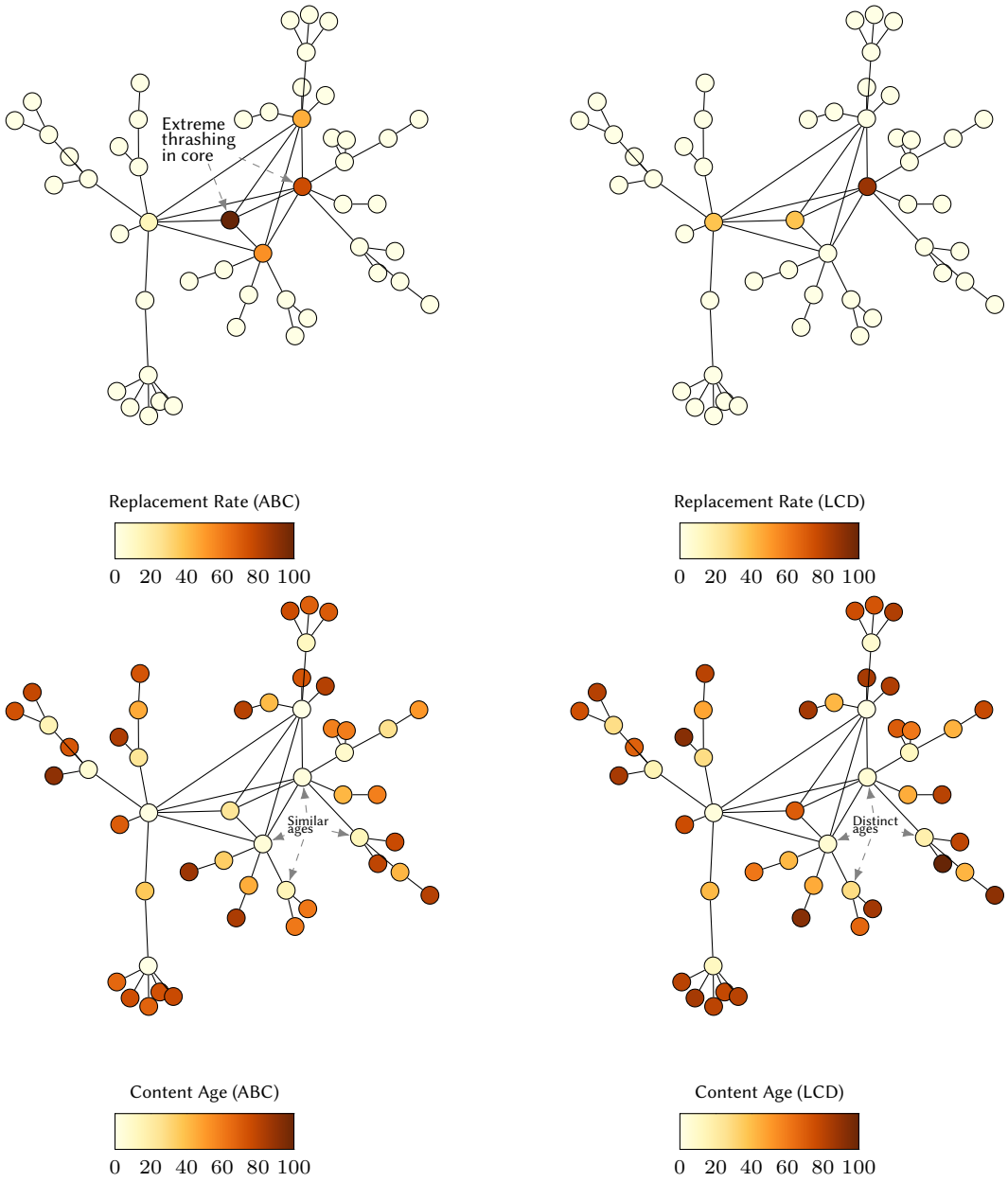


Figure 5.4.: Thrashing: *ABC* vs. *LCD*

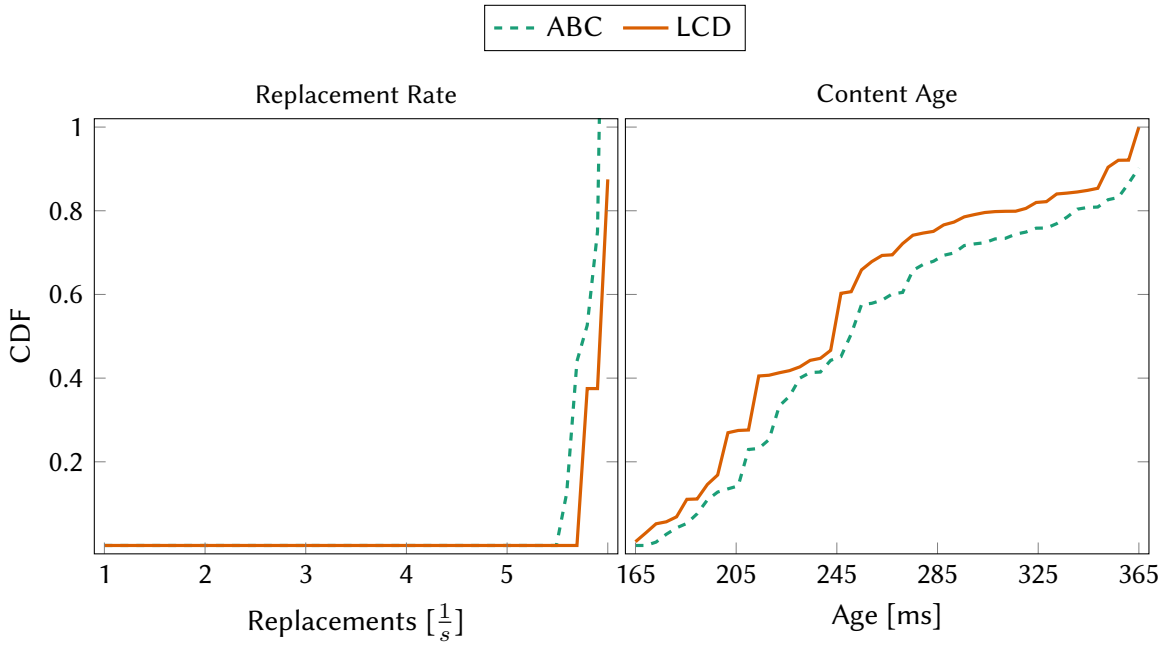


Figure 5.5.: Thrashing (CDF)

Figures 5.4 and 5.5 show the observed rate of thrashing at each node in the network, measured by both the rate at which content is replaced as well as the average age of the contents in each node's CS. There is a clear distinction between the core and non-core nodes in both strategies. The entire replacement load of the network appears to be confined to only the core nodes, while the content age distribution is continuous but clearly linearly increasing with decreased centrality. While we would not expect any significant thrashing to occur in the leaf nodes (since they only cache the contents they themselves requested), it is remarkable that not even the anchor nodes exhibit any notable thrashing. This can be explained by the fact that a Zipfian distribution is used for content requests, meaning that content that is more likely to be requested is also more likely to be found in a given CS. This mirrors the findings from Section 3.2.1. Coupled with the fact that anchor nodes are less likely to cache a given content chunk in the first place, this leads to negligible thrashing rates. By contrast, the core

nodes in *ABC* are exhibiting noticeable thrashing, with two nodes in particular appearing to churn through their contents at an extreme rate. These same nodes are also affected in *LCD* — again due to their central position in the network and being the likeliest caching candidate for any content originating in the core — but the thrashing rate is at a much lower level.

There is considerably more variation in average content age between the nodes than implied by the replacement rate. First and foremost, this is due to the fact that content requests follow a random distribution and are triggered at random intervals. The overall trend, however, is maintained. The oldest contents are found in leaf nodes, as these only cache their own requests, while core nodes have mostly recent contents. *LCD* is able to hold content for longer on average, especially at the leaf nodes. This is not an unequivocal positive, however, since very high average content age can also be interpreted as an underutilisation of cache space at these nodes.

There is an obvious difference in pure thrashing rate and average content age in regard to anchor nodes. While anchor thrashing rate is virtually identical to that of leaf nodes, their average content age is much closer to core nodes, particularly in the centrality-based approach. *LCD*, by contrast, sees anchor nodes in their own distinct age category, that is close to, but distinguishable from, that of core nodes. This shows how differently thrashing and average content age are distributed and that both metrics are necessary to give a holistic view of caching. For the thrashing rate, the distribution is heavily skewed towards the core nodes, with the majority of the nodes' values being relatively closer to each other, whereas for content age, the skew is on the opposite side, with the leaf nodes exhibiting higher values than the rest of the network.

The high rate of thrashing in centrality-based caching is likely to have a detrimental effect on the overall performance of the network, as individual content chunks will not stay in these nodes long enough to result in many cache hits.

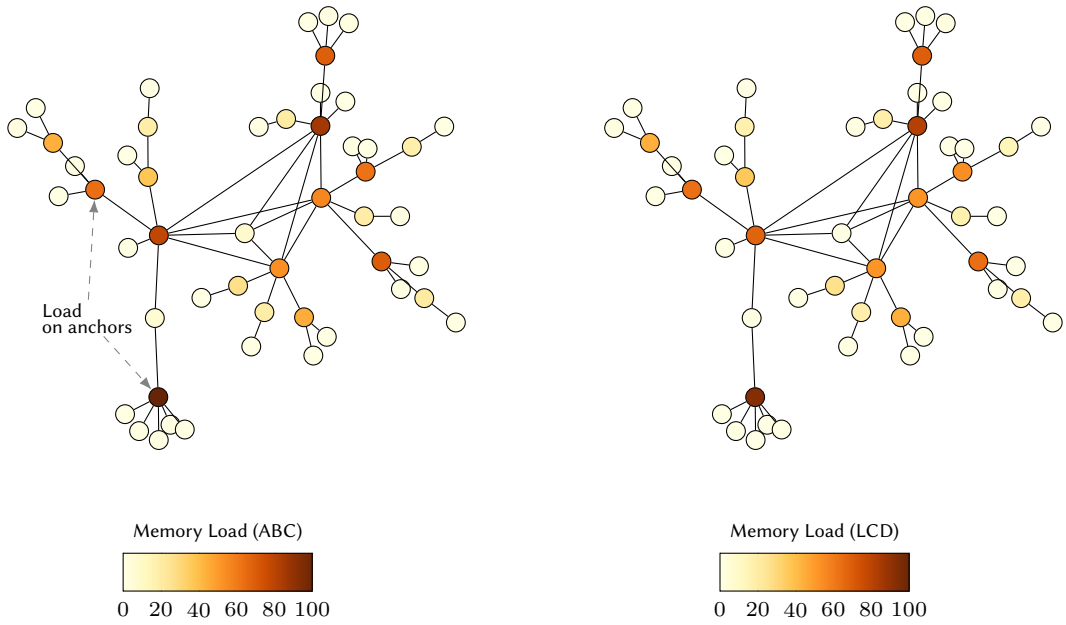


Figure 5.6.: Memory load: *ABC* vs. *LCD*

Memory Load

Figure 5.6 and the left-hand side of Figure 5.8 show the frequency at which nodes are forced to abort operations due to insufficient memory. In both strategies, this appears to affect anchor nodes above all, which when viewed in conjunction with Figure 5.2 implies that it is primarily PIT load that causes memory load and that both metrics are thus less dependent on the choice of caching strategy. This hypothesis is further corroborated by the fact that the maximum number of PIT entries is 20 by default, whereas the maximum CS size has been limited to 5 entries as in all previous experiments. This means that in the experiments presented here, nodes experiencing high PIT load will put a higher strain on their memory resources than those experiencing high CS load. However, it also means that this expression of load is highly configuration-specific and can be interpreted as a simple extension of the load on ICN data structures as described above. The differences in memory load between *ABC* and *LCD* are largely negligible.

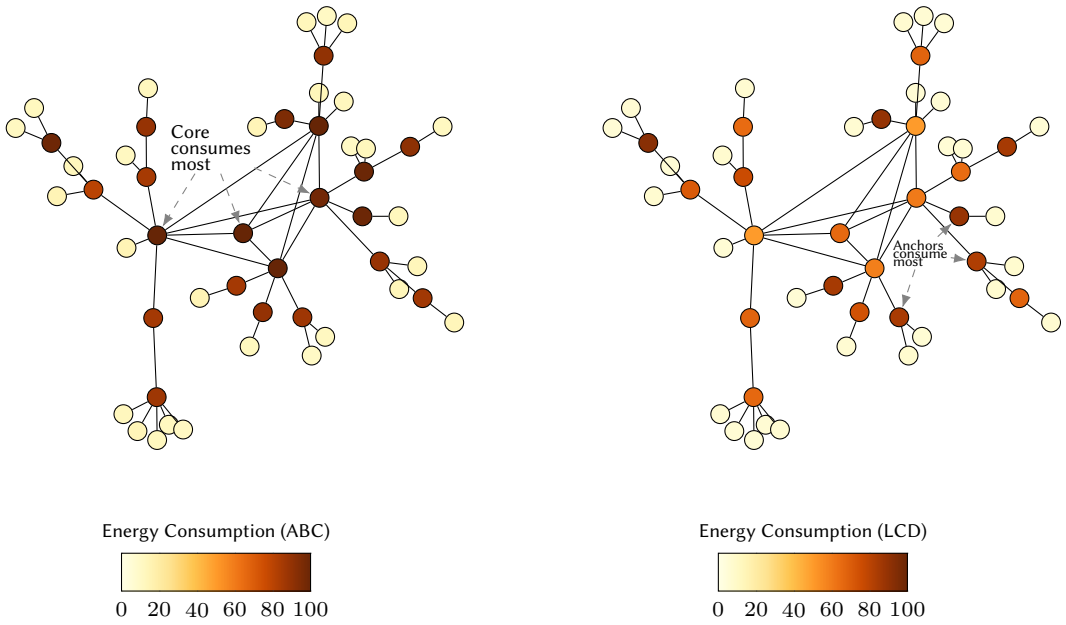


Figure 5.7.: Energy consumption: *ABC* vs. *LCD*

Energy Consumption

Figure 5.7 and the right-hand side of Figure 5.8 show the peak energy consumption of each node in the topology. Immediately, there is an obvious difference between *ABC* and *LCD* in that the former has a starker contrast between leaf and non-leaf nodes. While this difference can also be seen in *LCD*, energy consumption values there are less varied and lower overall. A further qualitative difference is that *LCD* appears to consume the most energy in anchor nodes, while *ABC* follows the familiar centrality curve with the highest consumption found in the core. Thus, it is possible that the main cause for energy consumption differs between the two strategies. In the centrality approach, it appears that the high CS load causes the affected nodes to consume more energy due to the increased frequency of read/write operations, while *LCD* energy consumption is more closely correlated to PIT load and the number of Interests handled by a given node.

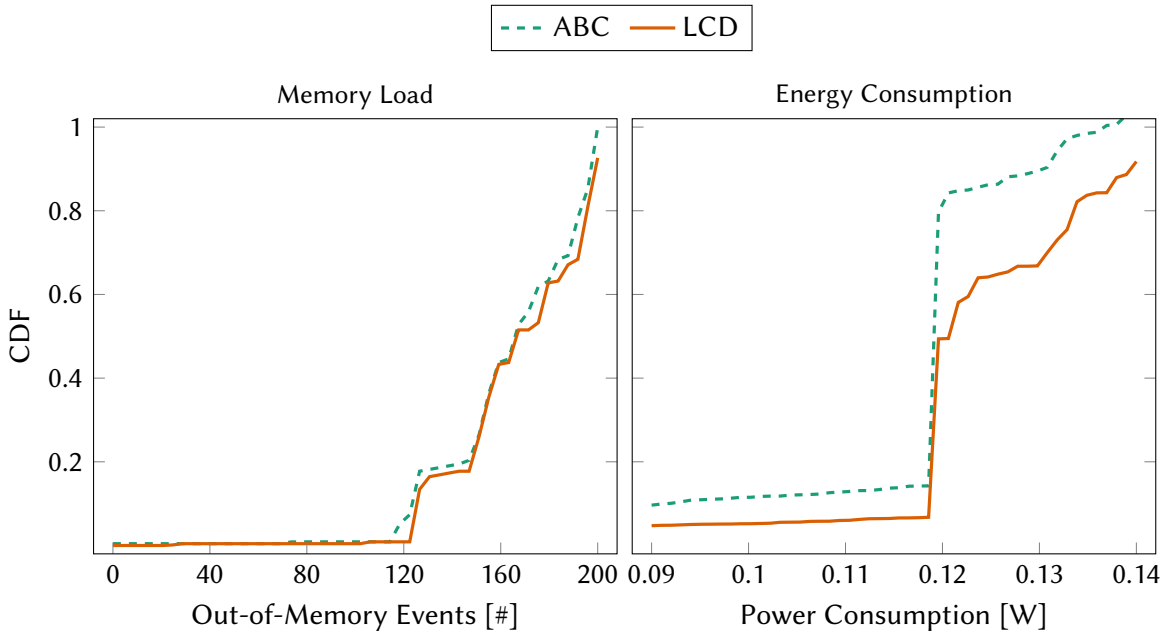


Figure 5.8.: Memory Load and Energy Consumption (CDF)

5.2.4 Summary of *ABC* Load Characteristics

The load placed on nodes by the *ABC* caching approach can be observed in a number of ways. The overarching observation is that, as postulated, load is correlated with centrality. While core nodes are affected by all of the effects falling under the mantle of "load", anchor nodes are only affected by a subset, namely anything resulting from PIT operations and, by extension, Interest management. Leaf nodes are largely unaffected. Almost all examined load metrics show that centrality-based caching experiences more — and more uneven — load than *LCD*. All of these observations can be traced back to *ABC*'s uneven caching characteristics. It obviously directly affects CS load and the rate of thrashing, but PIT load is also an indirect consequence, as fewer caching nodes means fewer paths in the network for Interests to take and thus a stronger concentration of pending Interests at fewer nodes.

Overall, then, it is clear that *ABC* could stand to benefit from a solution that would spread out the caching load more evenly across the network without sacrificing the performance benefits afforded by caching at central network locations. The following sections will explore how to implement such an approach.

5.3 Approaches to Off-Path Caching

As stated in the introduction to this chapter, a promising way to achieve a more even distribution of cache load is by applying the principle of *Off-Path Caching (OPC)* to the existing caching strategy. The following sections will discuss how to approach such a solution given the constraints of information-centric ICN.

The first question to answer when developing an OPC approach is whether the choice of caching node should be based on local or global information; in other words, whether the node that decides to offload its content to another node can choose its target autonomously based on the information it has or whether that target is determined by a central authority with full knowledge of the topology. As discussed in Section 2.2.6, most existing OPC approaches in traditional ICN choose the centralised route, because this allows making more informed decisions that are based on the state of the entire network. However, this approach is infeasible for the IoT domain, as both obtaining a global view of the topology and then disseminating either that information or a decision based on it to any node that needs it represents an unacceptable overhead.

Therefore, a solution based on nodes' local knowledge is necessary. Essentially, whenever a node is about to replace a content chunk in its cache with a new content chunk it has just received, it will need to make a local decision on which node to send the old content chunk to.

The offloading operation can be implemented using one of two general approaches. It could either be a direct command — i.e., the offloading node sends the offloaded content to the selected node directly — or it could be a request, meaning the offloading node broadcasts its intention to offload the content to its neighbours, who then choose whether or not to cache it based on their own state. However, the latter approach is far more resource inten-

sive, as it requires at least two broadcasts. The first broadcast offloads the content, the second notifies the caching node's neighbours of their decision to cache. This approach also raises non-trivial questions about contention and decision-making. With only local information, there is no guarantee for either a minimum or a maximum number of nodes caching the offloaded content, which could lead to either an over-replication of content, reducing effective cache utilisation, or an under-replication, where the benefits of OPC do not become apparent. Furthermore, the amount of FIB updates implied by multiple nodes caching each offloaded content chunk could easily counteract the desired gains in memory use. All in all, this approach does not seem feasible, which is why the former approach was chosen, named *Targeted Offloading (TO)* in the following.

5.3.1 Off-Path Caching by Targeted Offloading

In this approach, the offloading decision is confined to the node doing the offloading. Every one-hop neighbour of the offloading node is a potential offloading target. The offloading node chooses a neighbour based on a given criterion (see below) and sends the replaced content to that neighbour, who is then required to cache it. In the case of *ABC+OPC*, the criterion for the offloading decision is the neighbour's centrality value. However, this could easily be replaced with another measure, as discussed later in this section.

In the approach chosen for *ABC+OPC*, the offloading node chooses the node with the lowest centrality value among its neighbours. Nodes are informed of their neighbours' centrality values via regular link-local broadcasts, which incur a small, but manageable, communications overhead. Choosing the neighbour with the lowest centrality directly counteracts the load issue and makes the most efficient use of underutilised cache space. Cache space at more central nodes is still likely to be used effectively because of the higher number of requests going through them.

An argument could be made for caching at the neighbour with the highest centrality value instead. This would be a more conservative variant of the strategy, as content would tend to remain at high-centrality nodes while reducing some strain on the most central nodes. Keeping content at well-connected nodes means that the likelihood of direct cache hits remains high, thus benefiting content delivery latencies. This would be an approximation of the *transitive importance* measured by the eigenvector centrality approach (see Section 4.4).

On the other hand, it is questionable whether this alternative approach would adequately address the load balancing problem, as content would still be concentrated in core and anchor nodes with higher load. Keeping content central also means that the available cache space of leaf nodes would largely remain underutilised.

The main potential drawback of the lowest-centrality approach, on the other hand, is that it might negate the benefits of centrality-based caching, as content will now be stored at less well-connected nodes, meaning that the hops-to-hit value (see Section 3.3.1) is expected to increase. It will therefore be necessary to quantify the potential performance drawbacks of this approach compared to pure centrality-based caching without OPC.

As mentioned in the introduction to this section, making the offloading decision based on neighbour centralities is only one of many possible approaches. The decision could be made based on any other metric or combination of metrics, such as the neighbour's load, remaining battery life, content popularity or age, or wireless link quality, without affecting the basic functionality of the strategy. OPC is designed to be modular and the overall functionality is independent of how the offloading decision is reached.

Another open question concerns updating of nodes' FIBs. Once an offloading decision has been reached, it is necessary to enable forwarding of Interests to the new caching node. In normal (on-path) caching, this is not an issue since contents are always cached on the path taken by Interests anyway. If content is cached outside of that path, on the other hand, other nodes need to be made aware so that Interests can be forwarded to the new caching node. This raises several questions.

The easiest way to ensure Interests can reach the new caching node would be to simply update the offloading node's FIB to point to the caching node for the offloaded content. This would be enough to guarantee Interest delivery. However, it would also mean that most requests for the content would still be routed through the offloading node, meaning average hops to hit would increase while the amount of traffic handled by the offloading node would not decrease significantly; essentially, the only gain would be in better utilisation of cache space.

Therefore, it is better for the caching node to directly inform other nodes of the new caching location by broadcasting the new content to its one-hop neighbours so that they can update their FIBs accordingly. This makes it easier for new Interests to reach the content's new location. However, any such broadcast does imply additional communications overhead.

The last question that needs to be addressed is what to do if there is no suitable candidate node to offload the content to — e.g., when offloading to the neighbour with the lowest centrality, what happens if the offloading node is already the node with the lowest centrality in its one-hop neighbourhood. However, this question is easily answered by asserting that if a piece of content has been offloaded so much that it has reached a local minimum in terms of caching node centrality, it is likely unpopular enough that it can be removed from cache circulation entirely without affecting performance. Therefore, if the offloading node has no viable candidate to offload to, it will fall back to pre-OPC behaviour of simply evicting the content chunk.

The additional functions needed to extend *ABC* with OPC are shown in Algorithm 11. The extension consists of three simple steps:

1. Whenever a node would discard a content chunk from its CS, it instead sends that content chunk to the node with the lowest centrality among its one-hop neighbours with the instruction to cache it (see the OFFLOAD() function in Algorithm 11). Ties are broken by Random Number Generation (RNG).
2. The receiving node caches the content chunk and broadcasts a notification containing the content prefix to its one-hop neighbours (see the RECEIVE(`offloadedContent`) function in Algorithm 11). This serves as both an acknowledgement to the offloading node that offloading was successful as well as an instruction to all neighbours to update their FIBs.

Algorithm 11 Off-Path Caching by Targeted Offloading

```
1:  function OFFLOAD(content)
2:      targetNode ← n ∈ neighbours: ∀m ∈ neighbours: centrality_n ≤ centrality_m
3:      send(n, content)
4:  end function
5:
6:  function RECEIVE(offloadedContent)
7:      cache(offloadedContent)
8:      broadcast(offloadedContent.prefix, myAddr)
9:  end function
10:
11: function RECEIVE(offloadPrefix, addr)
12:     addFIBEntry(offloadPrefix, addr, static=false)
13: end function
```

3. Any node that receives the broadcast creates a *temporary* FIB entry for this prefix (see the RECEIVE(`offloadPrefix, addr`) function in Algorithm 11). By default, FIB entries do not expire in this setup because routes are static, but the NDN standard allows setting an expiry time. This option is utilised in order to allow Interests to be routed to the new caching location while accounting for the fact that the offloaded content will not be stored at that new location forever.

5.4 Evaluation of *ABC+OPC*

ABC+OPC was evaluated in the same experiment setup as used in Section 5.2. The new approach was compared to pure *ABC* across all load metrics introduced in Section 5.1. Furthermore, the performance was evaluated using the cache hit and hop reduction rate metrics introduced in previous chapters.

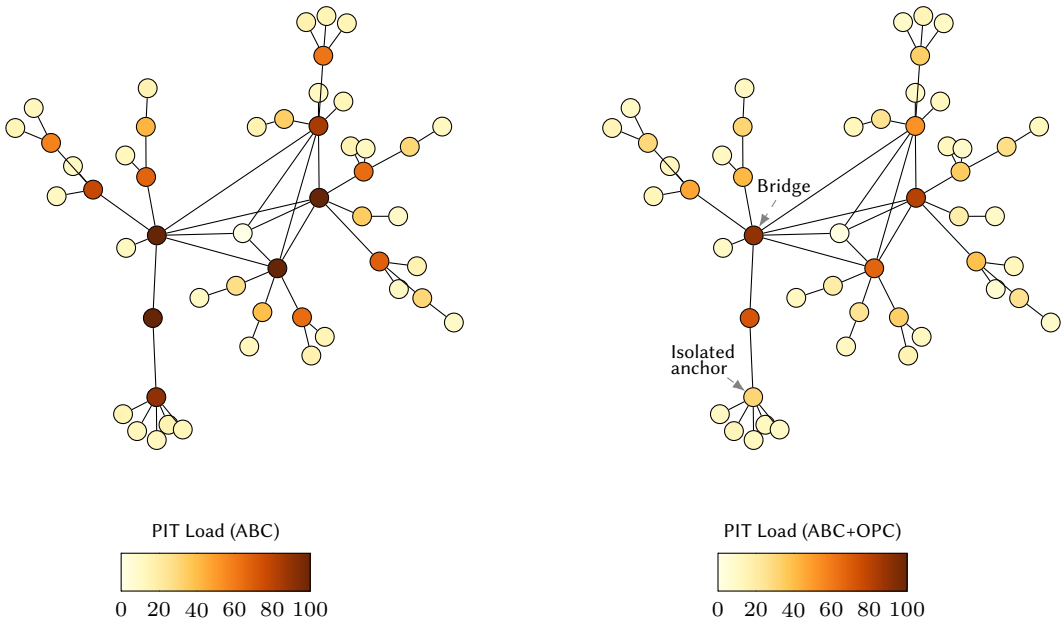


Figure 5.9.: PIT load: *ABC* vs. *ABC+OPC*

5.4.1 Load Improvements

Load on ICN Data Structures

Figure 5.9 and the left-hand side of Figure 5.11 show the change in PIT load after applying OPC to *ABC*. As can be seen, OPC results in reduced overall PIT load for almost all nodes, forming a similar distribution as *LCD*. Anchor and core nodes, which were previously under heavy load, see significantly reduced PIT usage in almost all cases, while load on leaf nodes stays similarly low to before, with minor improvements in some cases. Even relatively isolated anchor nodes (such as the one labelled in Figure 5.9) are able to offload their contents to the leaf nodes attached to them, allowing those leaves to exchange contents without going through the anchor. This in turn reduces load on the next anchor upstream.

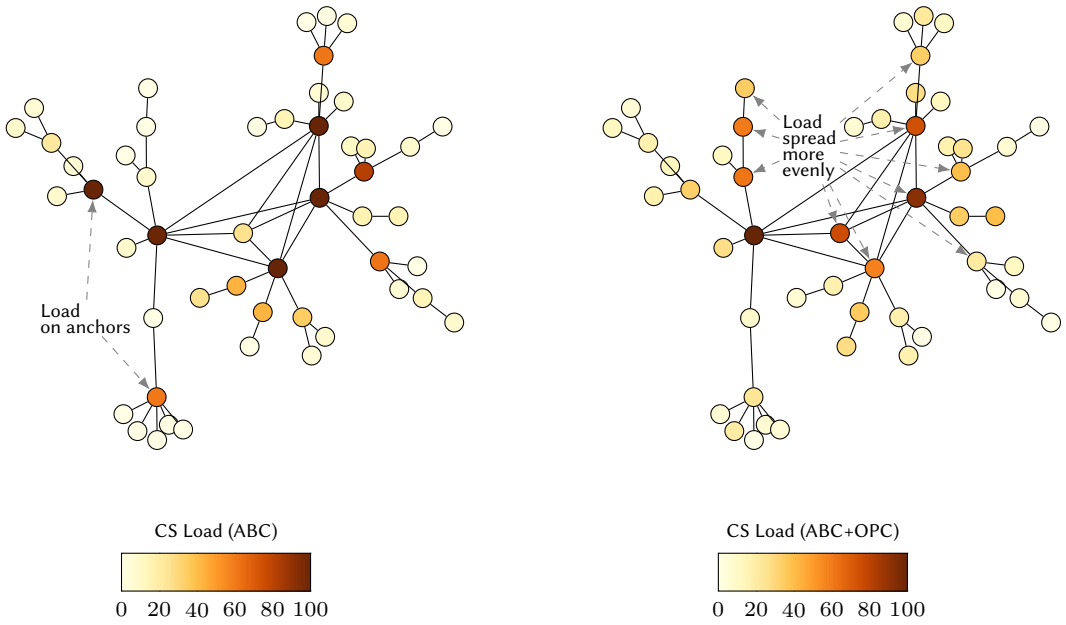


Figure 5.10.: CS load: *ABC* vs. *ABC+OPC*

In this experiment topology, there is only one core node (labelled as "Bridge" in Figure 5.9) that still experiences similar load to before. This node has the responsibility to bridge the connection between multiple large sub-trees and the core of the network. This shows the limits of OPC. If the topology has a single bridge between two partitions, with no alternative nodes to form alleviating paths, there is little opportunity for OPC to improve connectivity. However, performance issues arising from such a situation would be observed regardless of the chosen caching strategy, as they are simply a result of the topology. Thus, they would have to be addressed by improving the topology itself, e.g. by deploying additional nodes near the problem area.

Figure 5.10 and the right-hand side of Figure 5.11 show OPC's improvements in terms of CS load. In contrast to PIT load, which was decreased overall under OPC, the effect on CS load is more of an equalisation — the distribution is smoothed out. Total load is not decreased by much and remains higher than *LCD*, but load on core nodes is alleviated, while previously untapped resources in anchor and especially leaf nodes are utilised more. The result is higher

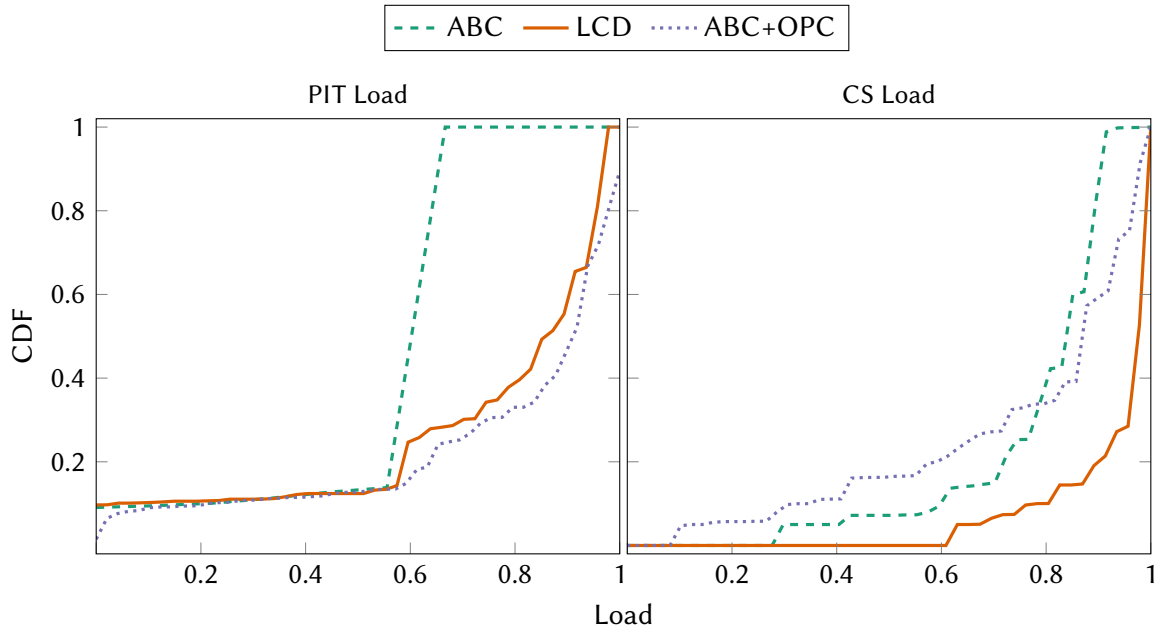


Figure 5.11.: Load on ICN data structures (CDF)

cache diversity, as explained in more detail in Chapter 3. By choosing to offload contents to low-centrality nodes that would otherwise be ignored by *ABC*, we have effectively added a large amount of caching space to the network, making it much easier e.g. for leaf nodes to exchange contents without having to go via core or even anchor nodes.

Thrashing

Without OPC, thrashing was observed at high rates in core nodes and at negligible rates everywhere else. That general trend is still clearly present even with OPC. However, Figure 5.12 and the left-hand side of Figure 5.14 show that the overall replacement rate has gone down slightly, pointing to improvement in caching efficiency. The replacement rate is now comparable to that of *LCD*, so the remaining differences between core and non-core nodes can be

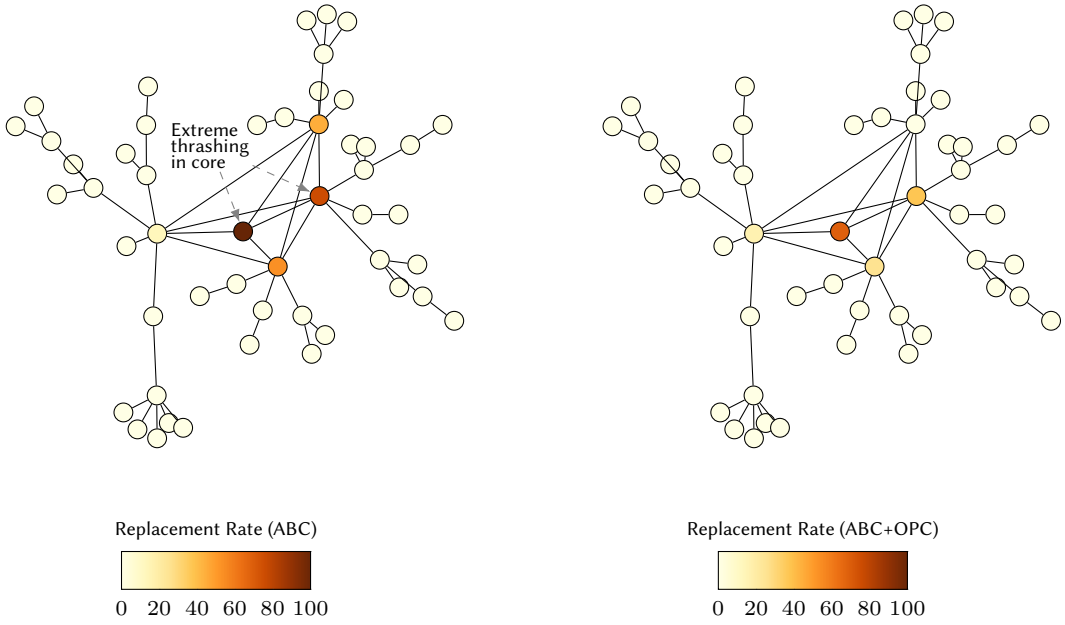


Figure 5.12.: Replacement rate: *ABC* vs. *ABC+OPC*

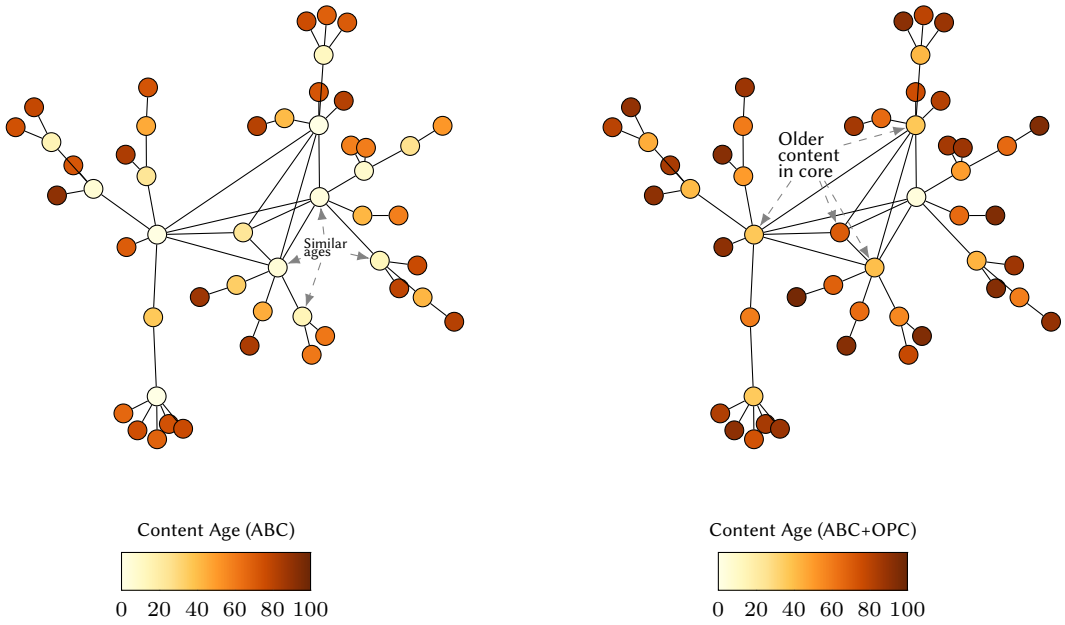


Figure 5.13.: Content age: *ABC* vs. *ABC+OPC*

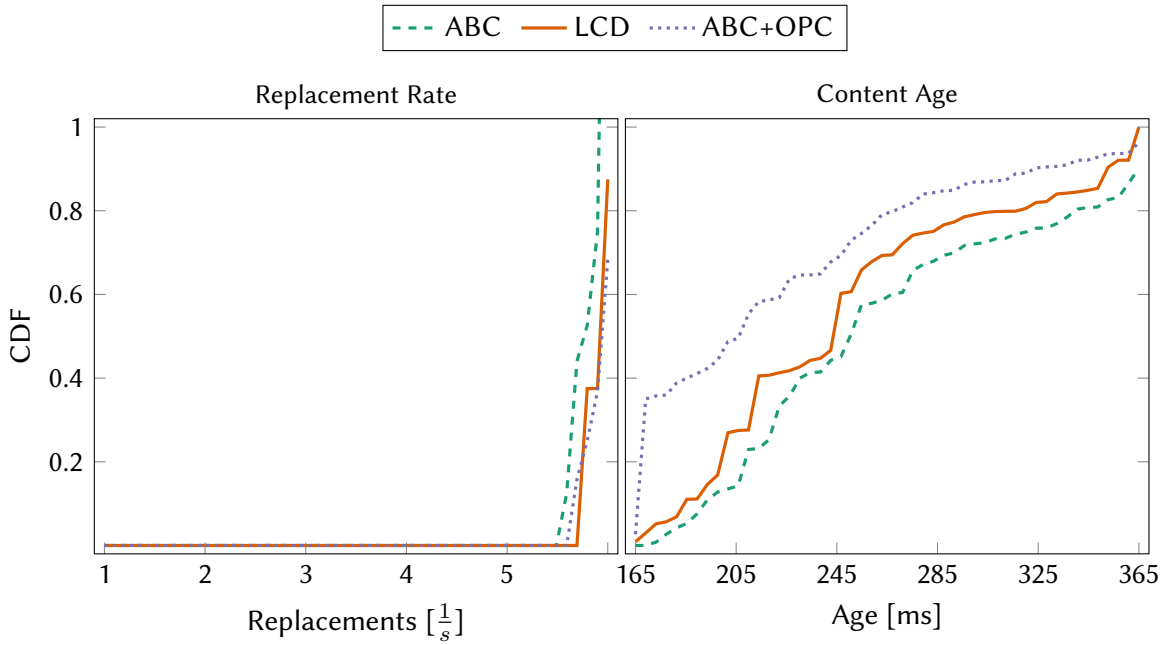


Figure 5.14.: Thrashing (CDF)

attributed to their positions in the topology. Since the replacement rate of non-core nodes stays the same while that of the core nodes is reduced, this means that OPC uses previously unused resources and thereby enabled the rest of the network to pick up some of the core nodes' slack.

The reduction in thrashing is more evident in Figure 5.13 and the right-hand side of Figure 5.14, which shows that the average age of content objects has increased for almost all nodes. While there is still a significant difference between core and leaf nodes — which is to be expected given the sheer difference in traffic handled by these node roles — content has a longer lifetime in most core nodes, which should have a positive effect on the cache hit rate (see Section 5.4.2).

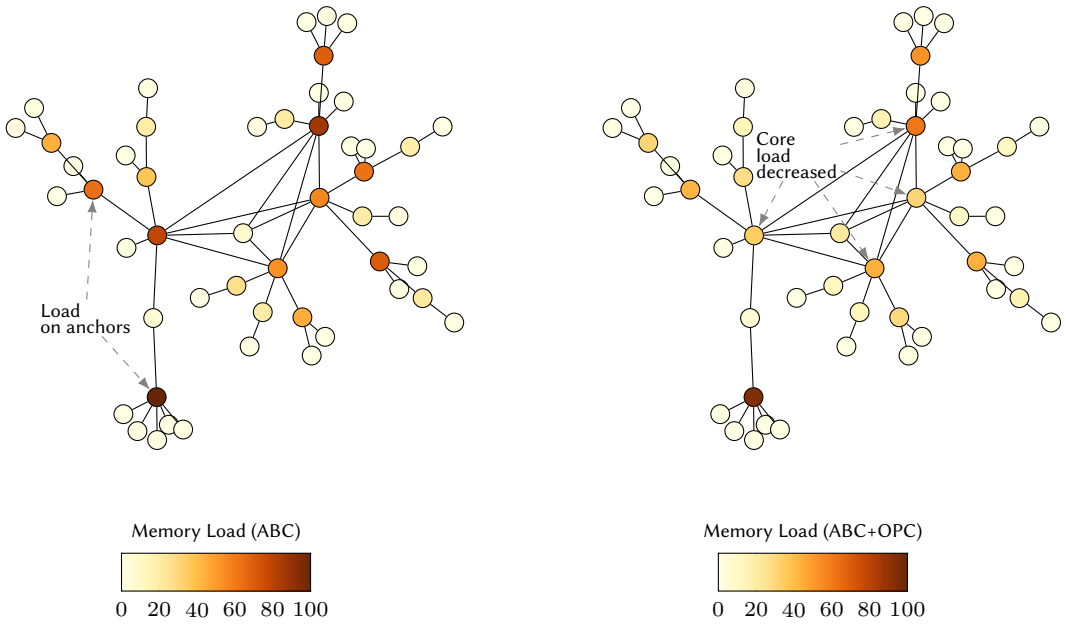


Figure 5.15.: Memory load: *ABC* vs. *ABC+OPC*

Memory Load

Under OPC, memory load still affects anchor nodes the most. Interestingly, the improvements to memory load are stronger in core nodes, implying that spreading out CS load as shown in Figure 5.10 has a bigger impact on memory use than the overall reduction in PIT load. Taken in conjunction with previous observations on PIT load, the implication here is that the memory load experienced by anchor nodes (which is an extension of PIT load) is more of a result of network topology rather than caching decisions, whereas core memory load (which comes from CS load) is clearly the result of the caching strategy. Therefore, it is intuitive that OPC would impact core nodes more strongly than anchor nodes, because their much higher CS access rates mean that they have more opportunities to offload contents and thus experience load equalisation, while PIT load can not be equalised as easily.

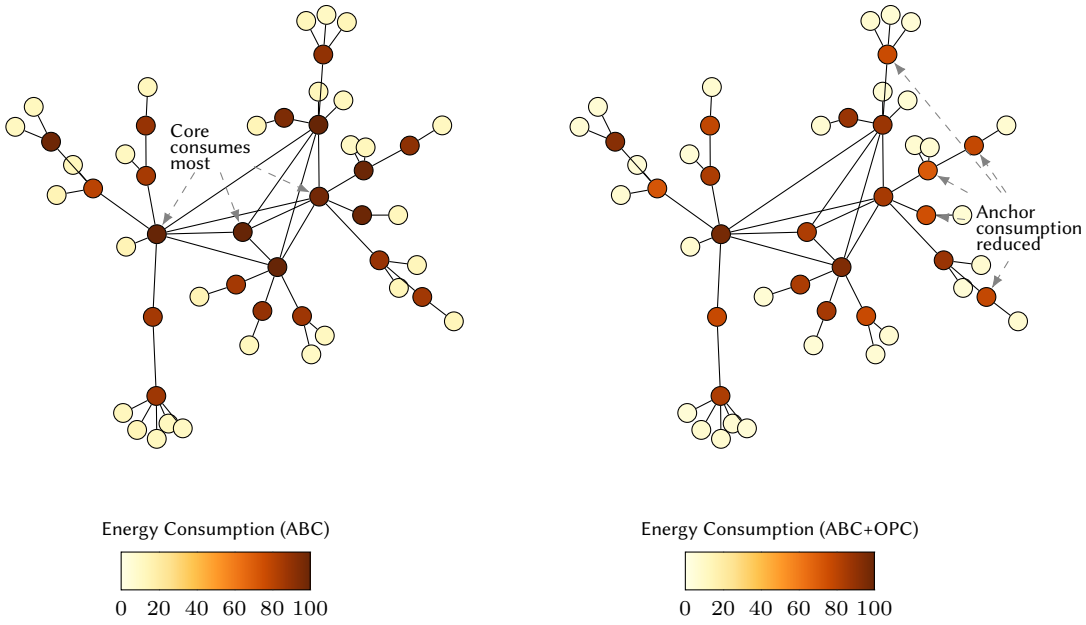


Figure 5.16.: Energy consumption: *ABC* vs. *ABC+OPC*

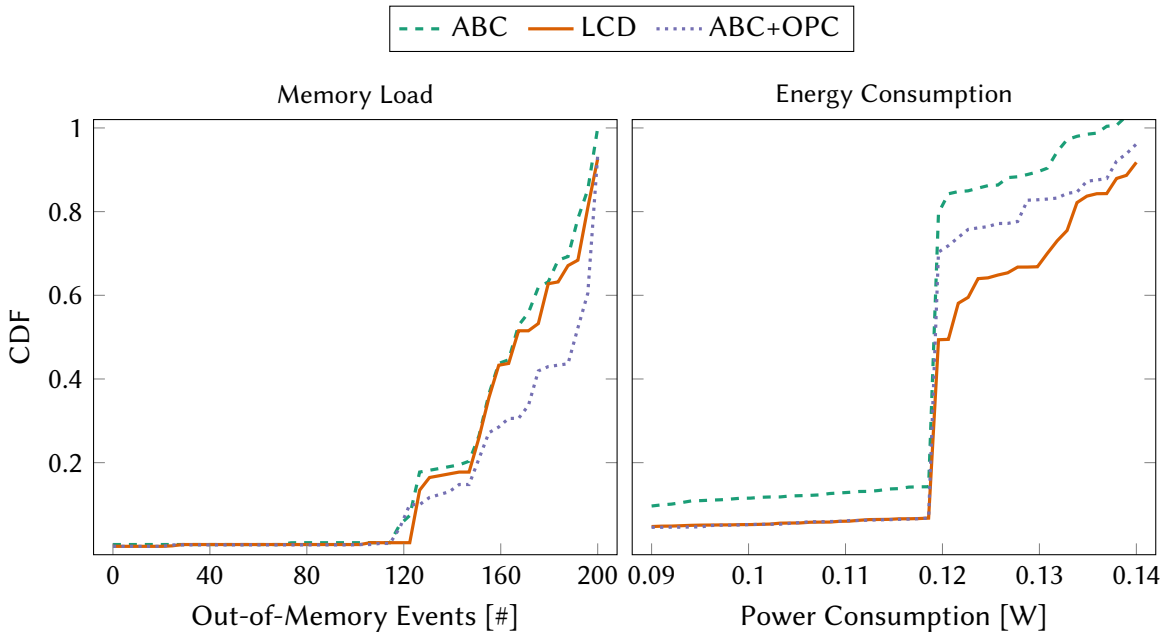


Figure 5.17.: Memory Load and Energy Consumption (CDF)

Energy Consumption

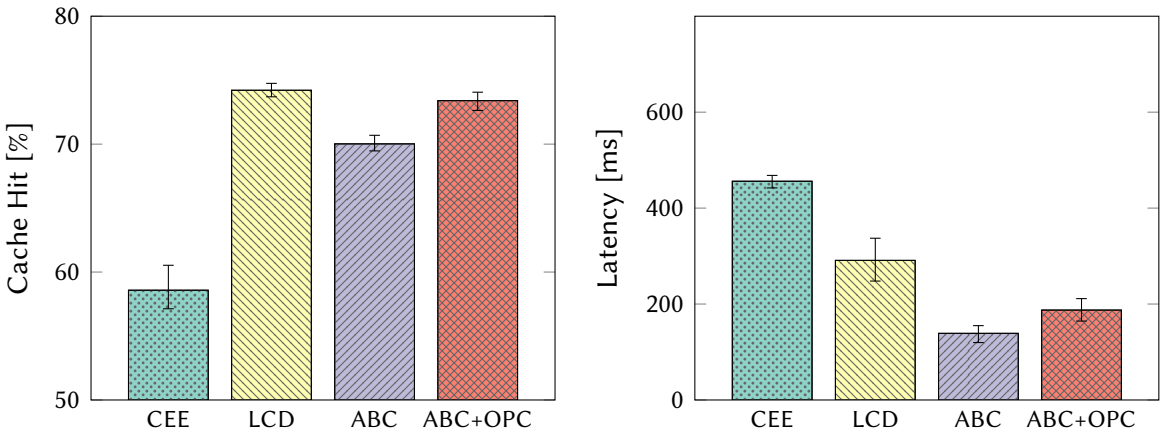
Extending *ABC* with OPC does not drastically alter the energy consumption profile. The overall shape of consumption remains the same, with a stark distinction between leaf and non-leaf nodes. However, there is a slight reduction of energy consumption levels at anchor nodes specifically, indicating the slightly decreased strain on these nodes. The difference, however, is fairly minute.

5.4.2 Caching Performance Metrics

While it is clear that OPC reduces the load of *ABC* across all metrics, it is important to ensure that this does not come at the cost of the other advantages *ABC* provides, namely its strong performance in terms of content delivery. In particular, since OPC will lead to some contents being cached at less central nodes, it would be reasonable to expect an increase in the average number of hops it takes to hit a cached copy of a content chunk, and therefore an increase in content delivery latency. The question is how much of an impact this has. Therefore, further evaluation is needed that compares *ABC+OPC* to non-OPC caching using some of the approaches discussed in previous chapters. To this end, *ABC+OPC* was compared to *ABC* (without OPC), *LCD*, and the default caching strategy *Cache Everything Everywhere (CEE)* in terms of the caching performance metrics used previously in this thesis: cache hit ratio, content delivery latency, and hop reduction rate. This evaluation is presented in the following sections.

Cache Hit Rate

The cache hit rate for each strategy was calculated in the same manner as done previously in Sections 3.2 and 4.3.4.



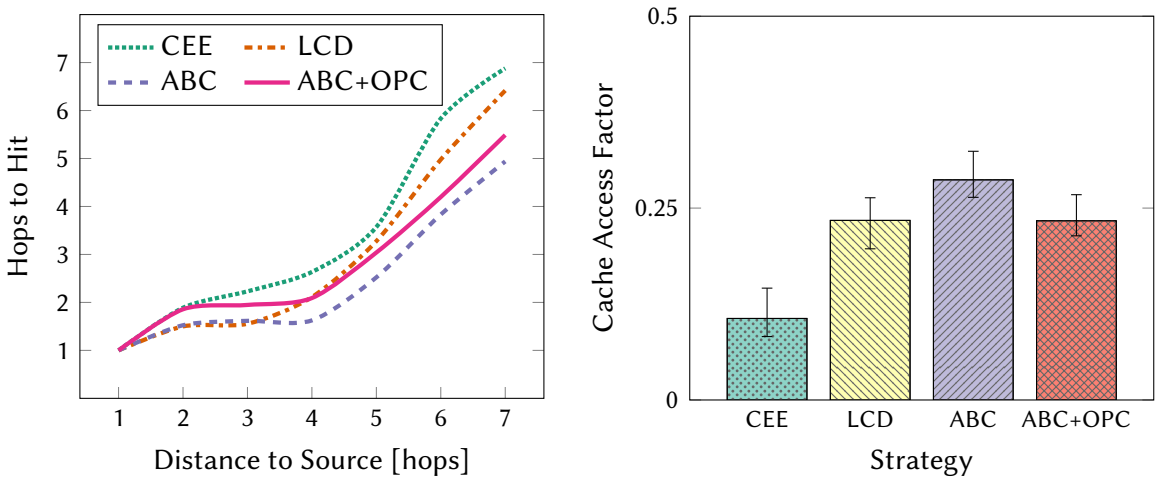
(a) Cache Hit Rate

(b) Content Delivery Latency

Figure 5.18.: Caching performance: *ABC+OPC* vs. other strategies

In Section 4.3.4, it was shown that the cache hit rate (and other performance metrics) of *LCD* were highly dependent on network topology, with the strategy posting strong results in core topologies and very weak results in edge topologies (cf. Section 2.2.4) due to the tendency to keep contents close to the core. The topology used in this experiment, which is closer to a realistic setup than the two extreme cases discussed previously, has elements of both topology types, with both a strongly connected core as well as anchor nodes with multiple branching leaf nodes.

As can be seen in Figure 5.18(a), this topology allows *LCD* to perform reasonably well while not outclassing other strategies entirely as was the case in a full core topology. *CEE*'s cache hit ratio stays at the expected level of just under 60% (*CEE* is largely unaffected by network topology). As would be expected from results in Chapter 4, *ABC* outclasses *CEE*, but it is not able to reach quite the same cache hit ratio as *LCD*, staying around 70%. However, once the strategy is extended by the OPC component, its cache hit rate increases to a level comparable to that of *LCD*. This means that OPC is indeed successful in not only reducing load, but also in making better use of available caching resources, allowing a greater number of content objects to be cached than before.



(a) Average hops to hit

(b) Cache access factor

Figure 5.19.: Hop reduction and cache access factor

Content Delivery Latency

While *LCD* has a better cache hit rate than *ABC* in this experiment topology, this does not directly translate to lower overall latency, as *LCD* keeps contents closer to the core while *ABC* places them at the most accessible location. For *ABC+OPC*, however, Figure 5.18(b) confirms the expectation that the OPC extension would result in a slight performance loss in terms of content delivery latency. Because some contents are now stored at nodes that are not as central and thus less easily accessible by other nodes, the delivery path lengths for some contents will increase. This increase, however, still results in lower latencies than those of *LCD*, which when taken in conjunction with the comparable cache hit rates in Figure 5.18(a), paints a very promising picture.

Hop Reduction

The increased content delivery latency shown in the previous section already implies a general trend for average path lengths in *ABC+OPC*. Nonetheless, it is helpful to further break this down into a ratio of *hops to hit* by *distance to source*. As explained in detail in Section 4.3.5, this ratio contrasts the distance in hops between a consumer and a producer of a given content chunk with the actual average number of hops it takes for that content chunk to reach the consumer. The more efficient a caching strategy is at making use of caching space and placing contents in accessible locations, the lower that strategy's *hops to hit* value.

While the shape of the *hops to hit* curves for the different strategies in Figure 5.19(a) differ slightly from those shown in Section 4.3.5 due to the different network topology, the relative differences between the strategies remain comparable. All strategies show some reduction in the *hops to hit* count, particularly for path lengths of around 4 hops. As expected given previous results so far, *CEE* performs the worst while *ABC* performs the best. *LCD* is not able to outperform the other strategies like it did in the core topology in Section 4.3.5, as was already implied by its slightly higher content delivery latency in the previous section.

This metric reveals some more interesting details about the behaviour of *ABC+OPC*. It starts off worse than *LCD* and *ABC* without OPC, but then manages to outperform *LCD* at path lengths of 5 or more hops. This means that the benefits of caching off the content delivery path become more apparent the longer that delivery path is. More intermediate hops means (in most cases) more one-hop neighbours to offload contents to, which increases the cache hit rate. While it never quite reaches the reduction rate of *ABC*, *ABC+OPC* is clearly a solid choice for networks with long delivery paths.

Cache Access Factor

The final performance metric to consider before coming to a conclusion on the effectiveness of *ABC+OPC* is its cache access factor. As defined in Section 4.3.7, the cache access factor combines the cache hit rate and the hop reduction rate into one metric to provide a weighted valuation of the expected gains in caching performance.

Figure 5.19(b) shows the cache access factors of the different strategies. With the topology being somewhere near the middle of the core-edge spectrum, the differences between the strategies are not quite as pronounced as in Section 4.3.7. *ABC*'s strong hop reduction rate, especially for shorter delivery paths, earns it the top spot. Interestingly, *ABC+OPC* and *LCD* end up sharing the same cache access factor, even though it might be expected that *LCD* would beat the new strategy thanks to its stronger performance at shorter path lengths, which tend to dominate the path length distribution. However, it appears that the fact that *ABC+OPC* matches *LCD* in cache hit rate and outperforms it in hop reduction at longer path lengths is enough to place it on the same level.

5.4.3 Summary of *ABC+OPC* Results

As anticipated, the introduction of OPC has resulted in measurable improvements to the load experienced by the network. Just as load is expressed in multiple ways as described in Section 5.1, the effects of OPC on the different load metrics is varied. Some aspects, such as the PIT load, see an absolute decrease as a more balanced use of available caching space results in more possible request paths and thus in Interests being spread more evenly, while others, such as the CS load, stay at a comparable level but are more evenly distributed, with previously underutilised nodes storing more contents and overloaded nodes experiencing some relief. The better distribution of load can also be seen in the reduction in thrashing, which is still present to some extent purely due to the fact that some nodes are more connected than others, but does not exhibit the same extremes as before. Consequently, average content age is also positively affected, with the average content chunk being cached for longer than before.

The tradeoff from the introduction of OPC comes, as expected, in the form of an increase in average content delivery latency, as some contents are now stored in less readily accessible locations. However, the increase is small, and *ABC+OPC* still manages to outperform *LCD* in the experiment topology.

5.5 Conclusions

This chapter has demonstrated that while load imbalance is indeed an issue for *ABC* as feared in Chapter 4, its negative side effects can be mostly mitigated by employing a simple off-path caching strategy on top of the existing caching decision strategy. This has beneficial effects across almost all metrics save for a tolerable increase in content delivery latency.

This chapter has introduced a nuanced way of understanding load in an information-centric IoT deployment. Rather than relying on a single metric, a range of factors affecting different components of the network was examined, and nodes were categorised according to their roles in the topology. This helps predict which nodes are most likely to suffer from which expression of load, and allows the design of a comprehensive strategy to combat performance drains caused by load imbalance.

An approach to off-path caching tailored towards centrality-based caching strategies for information-centric IoT was presented. This OPC approach follows the same design philosophy as *ABC*, in that it is intended to be easy to understand and implement and is designed with the severely limited hardware of the IoT in mind. That means no access to global knowledge and a minimisation of computational, memory, and communications overhead. Other OPC strategies, such as those discussed in Section 2.2.6, may theoretically deliver even better results, but are not feasible for deployment in a realistic IoT scenario.

The proposed OPC solution was compared against its non-OPC counterpart in terms of the load metrics introduced previously, as well as other familiar caching strategies across relevant performance metrics. The results show a reduction in load across almost all dimensions, with only a slight increase in energy use due to the additional communication introduced by the offloading operation, and an overall acceptable loss of performance that mostly concerns slightly increased content delivery times in exchange for more evenly balanced load and more stable cache hit rates.

Overall, *ABC+OPC* is a promising extension to the *ABC* caching algorithm, which maintains the lightweight approach of *ABC* while reducing the potential drawbacks of uneven load on load-sensitive IoT hardware. Chapter 4 concluded that if the network topology is well-known and immutable, it is very likely that a caching strategy can be explicitly tailored to be

optimal in that particular topology. However, if there is a need for a universally applicable caching strategy, for example if the topology is unknown or subject to change, a strategy such as *ABC* that performs reasonably well for all topology types is a viable option. The same considerations are true for *ABC+OPC*.

While the OPC extension presented in this chapter was developed and tested in conjunction with *ABC*, it is not exclusive to that strategy and could provide benefits for any centrality-based strategy. It may even be worth considering as an extension to other on-path caching strategies in case they struggle with similar load balancing issues.

CHAPTER 6

Increasing Content Delivery Reliability by Introducing QoS Constraints

While the previous chapters have focused on reducing latency, many IoT scenarios have an additional constraint: the reliability of content delivery. Reliable delivery is particularly important in areas such as industrial control systems (where high precision is required and often depends on not only low-latency but also lossless communications), disaster management and emergency alert systems (where every single message may be of critical importance), or autonomous driving. The inherent unreliability of IoT communication poses a significant hurdle to its pervasive application in these scenarios. However, as opposed to traditional solutions, where opportunities for Quality of Service (QoS) improvements are thought to be limited [24, 30, 31, 105] and essentially only extend to managing forwarding resources [155], ICN includes further resources, such as in-network caches, that can be exploited for QoS purposes.

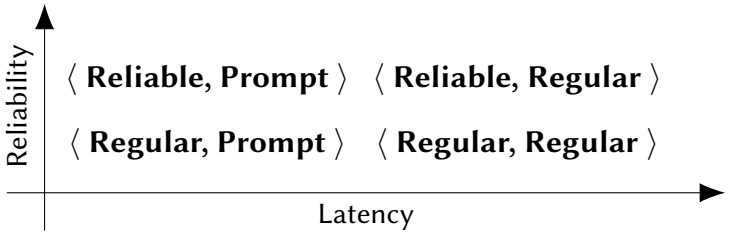


Figure 6.1.: QoS Service Levels [64]

This chapter presents a simple and straightforward approach to integrate QoS considerations into information-centric IoT. This is achieved by introducing the traffic service classes *prompt* and *reliable*. Traffic marked as *prompt* is given higher priority in terms of delivery latency, while *reliable* traffic receives higher priority in terms of in-network caching. A flow can have one, both, or neither of these classes. This chapter shows how a few simple adjustments to ICN primitives are enough to realise these service classes and presents an experiment that demonstrates the effectiveness of the proposed approach.

The contributions presented in this chapter are part of a larger set of contributions that are the result of fruitful collaboration with researchers from the Hamburg University of Applied Sciences, the Free University of Berlin, and Safety io, LLC. This chapter focuses on the caching aspect of QoS; the full scope of the proposed QoS improvements can be found in the publications resulting from this collaboration [62–64].

6.1 QoS Service Levels and Flow Classification

In order to achieve service differentiation in terms of both latency and reliability, the two quality dimensions *prompt* and *reliable* are proposed, as shown in Figure 6.1. For the sake of simplicity, only a binary distinction is made, but more fine-grained distinctions are possible. Traffic can then be assigned a service class using prefix matching. An example is shown in Table 6.1. Consider a simple building monitoring system with sensors that measure room occupancy and temperature and actuators for an alarm system, as well as any number of additional, miscellaneous sensors and actuators. Any traffic under the `/ICN/monitoring` prefix is by default treated as <*Regular, Regular*>, i.e. without any special QoS considerations. Then,

Table 6.1.: Examples of prefix to service class mappings [64]

Prefix	Service Class
/ICN/monitoring	<Regular, Regular>
/ICN/monitoring/occupancy	<Reliable, Regular>
/ICN/monitoring/temp	<Regular, Prompt>
/ICN/monitoring/alarm	<Reliable, Prompt>

the rules are extended to assign different service classes to certain subclasses of the monitoring traffic. Traffic from occupancy sensors is required to be *reliable* (i.e. this information should be guaranteed to arrive at the sink) but does not need to be delivered in a timely fashion. Conversely, temperature readings should be *prompt* because they are only useful within a limited time window, but dropped temperature packets are acceptable. Finally, packets that trigger an alarm need to be both *prompt* and *reliable*, i.e. they need to be delivered as fast as possible and never dropped.

This method of marking QoS service levels using the content prefix expands the role of the prefix beyond its influence on the forwarding plane and into further aspects of the NDN stack, such as the Pending Interest Table (PIT) and the Content Store (CS). Consequently, it introduces some new constraints regarding namespace design. The hierarchy of the namespace needs to be constructed with QoS service levels in mind in order to be able to benefit from the differentiated services introduced in this chapter. However, in most deployments, this should be easy enough to implement by extending content prefixes with appropriate service level markers.

6.2 QoS-Enabled In-Network Caching

The introduction of traffic flow priorities adds an additional dimension to the caching decision strategy. Regardless of which specific approach is used, content marked as *reliable* should always be cached, as it is imperative that this content is available throughout the network. Thus, reception of *reliable* content should not trigger the caching decision strategy;

instead, control should be handed directly to the cache replacement policy (see below). The question whether *prompt* content should be cached with a higher priority than content with *regular* latency requirements is not as clear-cut. Caching *prompt* content with higher priority would have a positive effect on future transmissions of that content object (either by retransmission of the original request or by new requests) and thus have a positive effect on latency, although the potential gain in this aspect is dependent on path length. Any content that is marked as *regular* in both QoS dimensions should be treated as normal; in other words, the caching decision strategy is consulted.

After a node has decided to cache a new content object, the cache replacement policy is typically consulted if the CS is at capacity. As explained in Section 2.2.3, in most cases, CS contents will be replaced using a simple heuristic such as *Least Recently Used (LRU)*. However, once again the introduction of traffic flow priorities adds an additional dimension to this decision.

In general, incoming content should not replace content of a higher priority. Therefore, content with *regular* latency requirements should not replace *prompt* content and content with *regular* reliability should not replace *reliable* content. When it comes to the correlation between latency and reliability, the primary goal of the CS should be to ensure content availability, which places a stronger emphasis on the reliability aspect. Thus, *reliable* content with *regular* latency should be able to replace *prompt* content if no other content is eligible to be replaced. If all content is of the same priority class, regular replacement rules (e.g. *LRU*) should apply.

In probabilistic caching, as introduced in Section 2.2.2, each node caches incoming content according to a certain probability p. Regardless of how exactly p is determined (whether statically or dynamically), the probabilistic approach may be refined by differentiating between two separate probabilities p_{reg} for regular content and p_{rel} for reliable content, with $p_{rel} > p_{reg}$. This has the effect that a CS at each node will have different contents, thus contributing to CS diversity across the network by making a larger range of content available as cached copies, while giving consideration to service classes ensures that higher-priority content is still treated preferentially.

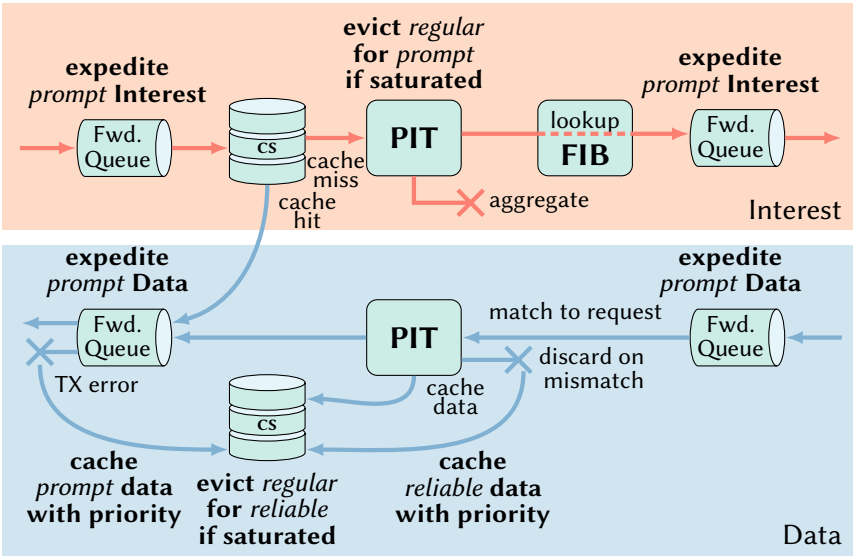


Figure 6.2.: Flow description for Interest and Data messages in a QoS enabled NDN forwarder [63]

Figure 6.2 shows the complete flow of ICN decisions with QoS service levels enabled, as an extension of the flow diagram shown previously in Figure 2.4. This includes alterations made to the forwarding queues and the PIT, which are described in detail by Gündoğan *et al* [62–64].

6.3 Starvation and Fairness

Introducing QoS considerations to ICN caching creates further open questions that need to be addressed. If CS contents can only be replaced by new contents and unprioritised content may not replace prioritised content according to Section 6.2, then it is easy to construct a scenario in which the CS is fully saturated with prioritised content and unprioritised content is starved. In an extreme case, the entire caching space of the network could be poisoned by an initial burst of prioritised content. Then, if all subsequent traffic is unprioritised, none of it would be cached given the simple rules stated above. This further implies that unprioritised

content would not even be able to rely on intermediate caches as retransmit buffers, potentially causing transmission to fail entirely. Therefore, fairness measures are needed for QoS enabled caching. The following sections will identify two specific starvation-related issues and introduce potential countermeasures to address them.

6.3.1 Cache Blocking

Cache blocking occurs when prioritised content takes up all cache space in a CS, preventing unprioritised content from being cached. This can be prevented by introducing a priority decay time τ, which specifies how long a piece of content may be treated as prioritised. Any prioritised content chunk that has been in the CS for a time $t \geq \tau$ is reclassified as unprioritised and thus free to be replaced by newer unprioritised content. The value of τ depends on the application, but a practical value would be the expected time of cache utility, i.e. how long on average a given content item will be valid or useful.

6.3.2 Cache Preemption

When prioritised content arrives at such a rate that unprioritised content can never be stored in a CS for long enough to make an impact on performance, this is called *cache preemption*. The priority decay time τ introduced above can not prevent this if new prioritised content arrives faster than the priority decay rate. In this case, any unprioritised content in the cache will be evicted as soon as new prioritised content arrives. The preemption problem could be countered by reserving a certain proportion of the CS for unprioritised content (which may be problematic if CS capacity is extremely limited) or by modifying τ. However, these solutions are highly dependent on scenario-specific traffic patterns and cannot be generalised. Therefore, a simpler solution utilising probabilistic caching may be more appropriate.

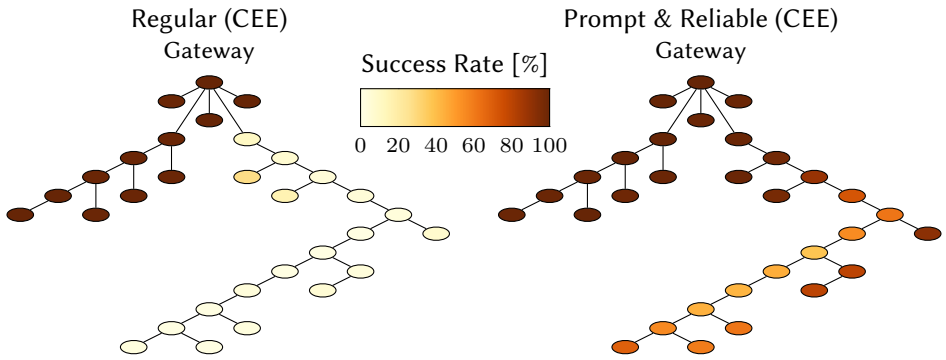


Figure 6.3.: Nodal success rates for *Scenario 1* using *regular* traffic and *reliable* actuator traffic [62]

6.4 Evaluation

As in previous chapters, a series of evaluation experiments was conducted on the **FIT IoT-LAB** [3] using the same constrained hardware as before (see Section 3.2 for details). The **RIOT-OS/CCN-lite** stack on the nodes was extended with the QoS management scheme described above.

6.4.1 Experiment Parameters

The experiments were conducted with two different caching decision strategies — *CEE* and *Prob(p)* — and with *LRU* as the cache replacement policy. *CEE* functions almost exactly the same as in a non-QoS enabled system, except that incoming *reliable* Data with no corresponding PIT entry (which would normally be discarded) is cached opportunistically as described in Section 6.2. *Prob(p)* is extended to use the two probabilities p_{reg} and p_{rel} for regular and reliable content. They are set to 30% and 70% respectively.

6.4.2 Experiment Topology

An experiment topology of 31 M3 nodes was constructed in the *Grenoble* site[1]. The topology (shown in Figure 6.3) is a Destination Oriented Directed Acyclic Graph (DODAG) with 30 nodes forming two main branches with path lengths of up to 12 hops connected to a single gateway. The two branches correspond to two corridors in the IoT-LAB *Grenoble* site.

6.4.3 Experiment Scenarios

Three experiment scenarios were designed using this topology in order to show the impact of QoS provisioning on the performance of the different ICN resources.

Scenario 1: PITs and Forwarding

The first of these experiments focused on the impact on the forwarding queues and the PITs. The results have been published previously [62, 63]. They are not discussed here because PIT and forwarding queue considerations are not closely related to the caching questions discussed in this thesis. However, Figure 6.3 gives an impression of how forwarding efficiency is improved by the introduction of QoS measures.

Scenario 2: Caching

The second experiment scenario was designed to measure the impact of QoS enabled caching on network performance. In this scenario, the gateway requests temperature readings with increasing sequence numbers from the other 30 nodes every 8–12 seconds. Furthermore, the non-gateway nodes send actuator requests with increasing content IDs to the gateway every 4–6 seconds. Traffic is thus bidirectional and divided into *sensor traffic* and *actuator traffic*.

[1] https://www.iot-lab.info/deployment/grenoble/

The nodes are randomly assigned to one of five *actuator groups*. Nodes in the same group receive identical commands from the gateway, meaning that actuator traffic can be cached. For sensor traffic, the CS only functions as a retransmission buffer because sensor requests are device-specific and sequence numbers do not repeat.

In this scenario, the rate of successful deliveries, the content delivery latency, and the cache hit ratio are measured.

Scenario 3: Starvation and Fairness

The third experiment scenario is identical to the second scenario, except that the priorities of sensor and actuator traffic are inverted. The sensor traffic, which due to its sequential, node-specific nature can not benefit from caching, is prioritised, while the actuator traffic is unprioritised. Furthermore, CS capacity is limited to 5 throughout. This should help shine a light on starvation effects.

In this scenario, the average CS utilisation as well as the delivery success and cache hit rates are measured.

6.4.4 Results

Scenario 2

Figure 6.4 shows that the success rate of content delivery for *prompt* and *reliable* traffic is almost 100% across all configurations. With a sufficiently large CS, delivery of *reliable* content becomes virtually guaranteed. Severely limited CS sizes (capacity 5) as used in previous experiments in this thesis still result in unreliable sensor traffic at high path lengths, but even an increase to 10 entries is enough to ensure success rates above 80%. Contrast this performance with that of *regular* traffic, which at large delivery distances can not exceed 70% even with a cache capacity of 30. As would be expected given the conclusions from Chapter 3, the probabilistic caching strategy performs better than *CEE* thanks to its increased cache diversity. However, the difference is only significant with small caches and long delivery paths.

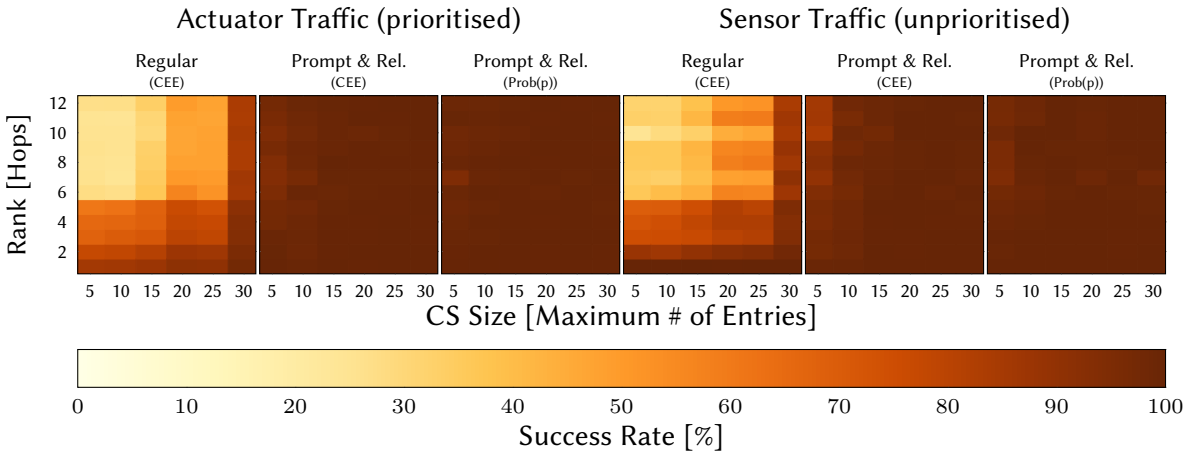


Figure 6.4.: Success rates per rank for *Scenario 2* using varying CS sizes [62]

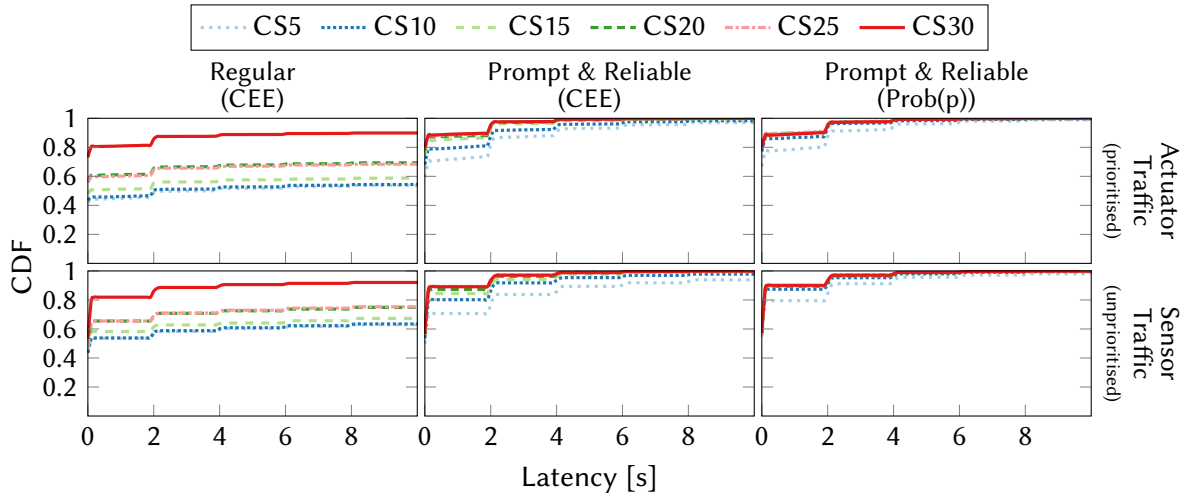


Figure 6.5.: Content delivery latencies for *Scenario 2* [63]

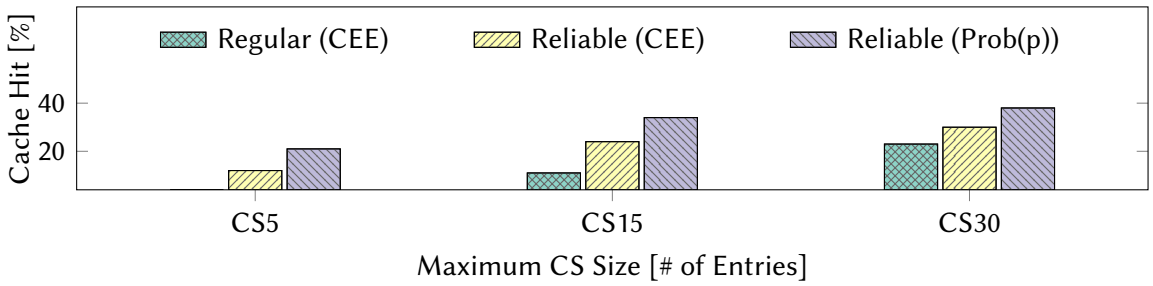


Figure 6.6.: Cache hit rate for actuator traffic in *Scenario 2* [62]

Figure 6.5 shows that the introduction of QoS service classes has a positive impact on content delivery latencies. Again, *Prob(p)* outperforms *CEE*, although for this metric the difference is not as strong and mostly affects smaller cache capacities. Furthermore, the differences in CS capacity are less significant when QoS classes are observed. However, this is most likely due to the adaptations regarding prioritised forwarding rather than caching alterations. *Prompt* traffic is prioritised and therefore less affected by cache replacement effects. On the other hand, for *regular* traffic, not even a CS capacity of 30 is enough for full reliability.

The cache hit rate as shown in Figure 6.6 further supports the previous observations. The experiment scenario, with its extremely long paths, was designed not to be conducive to caching efficiency. This is reflected in the poor performance of the *regular* traffic. The QoS enabled traffic, on the other hand, shows significant improvement for both caching strategies even with minimum cache capacity. This is because the *reliable* actuator traffic is virtually guaranteed to be found in a cache on the path thanks to its prioritisation. As in Chapter 3, probabilistic caching further increases the cache hit rate.

Scenario 3

To highlight the impact of starvation, the average CS utilisation over time for a specific node is shown in Figure 6.7. This node is the direct successor of the gateway node on the right-hand side of Figure 6.3. It is the root of a large subtree with 18 nodes and thus expected to route large amounts of both prioritised and unprioritised traffic in both directions.

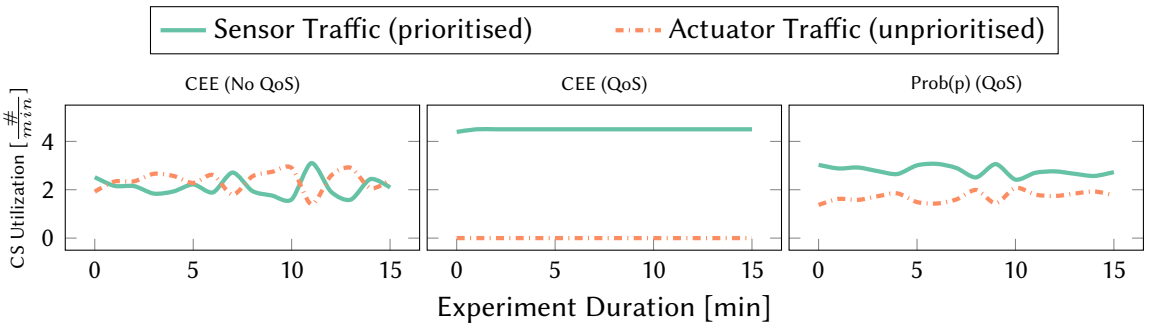


Figure 6.7.: CS utilisation in a starvation-prone node in *Scenario 3* [63]

It is clear that the unprioritised actuator content is indeed prevented from being cached when using *CEE*. At no point during the experiment run is there a significant amount of unprioritised content in the node's CS. Contrast this with the scenario in which QoS is disabled. In that scenario, the ratio of sensor to actuator traffic stays constant on average and only swings in one or the other direction for short amounts of time. On the other hand, when using probabilistic caching (with $p_{reg} = 0.3$ and $p_{rel} = 0.7$ as established above), the effects of starvation are diminished. While prioritised content is still more prevalent, there is always some room for unprioritised content (about 30% of the available space on average). This reflects the desired CS composition, as prioritised content is supposed to be afforded more of the available caching space, but not at the cost of shutting out unprioritised content entirely.

Figure 6.8 shows the success and cache hit rates for the third scenario with QoS disabled and enabled for *CEE*, as well as with QoS enabled for *Prob(p)*. We can see that enabling QoS greatly increases the success rate of the prioritised sensor traffic while having a smaller impact on the unprioritised actor traffic[2]. This is true for both caching strategies. In contrast, while actuator traffic profits from caching with QoS disabled due to the fact that it uses groups as described in Section 6.4.3, these cache hits are prevented entirely when QoS is enabled. Instead, caches are poisoned with prioritised sensor traffic, which does not profit from caching due to its

[2]Even unprioritised traffic profits from enabling QoS because it significantly reduces retransmissions, which has a positive effect on overall network load. This is explored in more detail in *Scenario 1* in previously published work [62, 63].

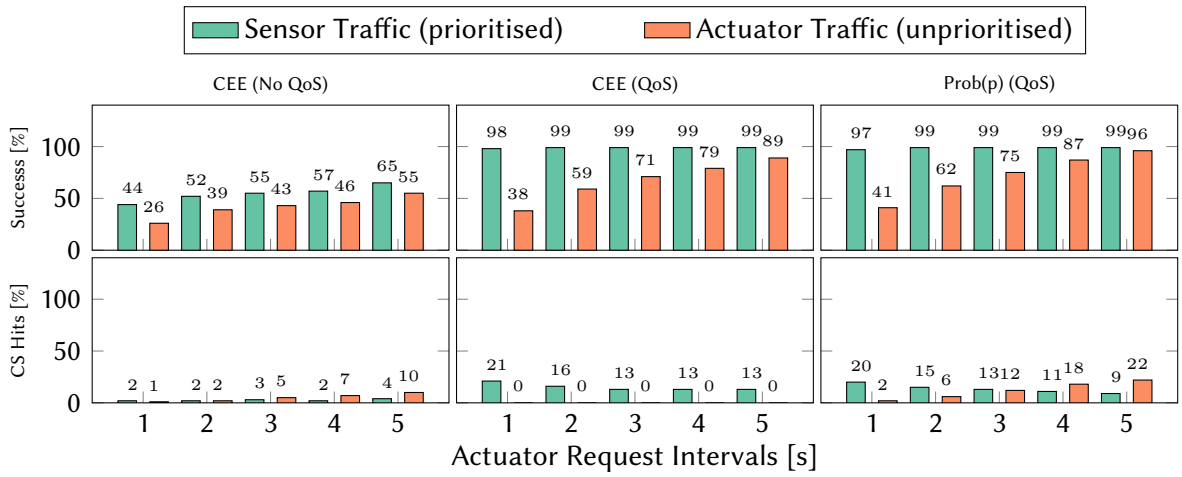


Figure 6.8.: Success rates and cache hits for different actuator request intervals in *Scenario 3* [63]

strictly increasing sequence numbers. Using probabilistic caching brings the cache hit rate of unprioritised content back up while allowing for both types of traffic to be cached. At longer actuator request intervals, actuator cache hits outperform those of the prioritised content because the cached contents are valid for longer and are replaced less frequently.

6.5 Conclusions

The QoS extensions discussed in this chapter are easy to implement in information-centric ICN and show promising results. The interconnected resources specific to ICN are exploited to realise a simple QoS system based on service classes. The role of caching in QoS-enabled ICN is significant, and particular care must be taken to ensure fairness and prevent starvation. The evaluations presented here show that enabling QoS can improve the content delivery performance of the network as a whole, not just for prioritised traffic flows. They also prove that ensuring fairness in a lightweight and non-intrusive manner is possible while still maintaining QoS guarantees.

CHAPTER 7

Conclusions and Future Work

Information-Centric Networking is a promising new approach to networking that warrants further attention. Particularly in the Internet of Things, where blank-slate approaches such as ICN are much more realistic to deploy than in the global Internet, it offers a range of tangible benefits such as a drastically compressed network stack and a content-centric philosophy that meshes well with the type of traffic IoT applications typically produce.

Nevertheless, there is a range of issues and open questions raised by the prospect of bringing ICN to the IoT owing to the idiosyncratic nature of the hardware used in that domain. Many of the assumptions that underlie the variations on ICN proposed for traditional networks simply do not hold true in the IoT space. Network participants are extremely heterogeneous, with some having severe limitations in terms of processing power and/or available memory. This means that the ubiquitous caching that is fundamental to ICN can not be realised in the IoT. Furthermore, ICN generally assumes stable, wired links between network participants, something IoT can not generally guarantee. This means that further research is needed in order to bring the advantages of ICN to the IoT in a way that actually makes it a worthwhile competitor to the current state of the art.

To that end, the question of how and where to cache content was identified as one of the most important open research questions in the field at the time of writing. Although some research had been conducted into this and some promising approaches had been developed, there had been no general attempt to fully characterise the adaptations needed to make full use of ICN's caching philosophy in the IoT. There was no comprehensive study on how existing caching strategies actually behaved in realistic IoT scenarios with real hardware, and very little research in the way of developing solutions that would actually work well on such hardware. This brings us to the contributions of this thesis.

7.1 Review

The contents of this thesis are summarised below:

- Chapter 3 presented a series of experiments intended to characterise the behaviour and performance of a range of potential solutions for caching in information-centric IoT. Useful metrics for such a performance analysis were introduced, including the new *cache access factor* metric that combines the cache hit and hop reduction ratios into a single metric. A large number of caching decision and cache replacement strategies (previously introduced in Chapter 2) were compared across these metrics using real IoT hardware. The first important finding was that in general, the choice of cache replacement strategy appears to have little impact on caching performance, while there are significant performance differences between the different caching decision policies. These findings were expanded upon in further experiments broadly focused on the efficiency of content delivery, in particular the reduction of delivery path lengths and thus content retrieval latency. These experiments demonstrated that topology effects account for a significant proportion of the caching behaviour of some of the more prominent caching strategies, an effect that, while not entirely unknown, had not previously been considered in the design of new caching strategies.

- The previous chapter's findings about the effects of network topology on caching efficiency led naturally to the development of a new caching strategy for information-centric IoT that would take these effects into account explicitly. This new caching strategy is called *Approximate Betweenness Centrality (ABC)* and comprises the main contribution of Chapter 4. By leveraging the concept of betweenness centrality, a caching algorithm was designed that relies exclusively on local information to store contents at the most accessible point in any given network, regardless of its topology. This approach was compared to several other state-of-the-art caching strategies using real IoT hardware, and while it did not outperform strategies that relied on full network information (which is unfeasible in a full IoT setup), it did exhibit very comparable performance while fully adhering to the restrictions imposed by the limited IoT hardware.
- Chapter 5 expanded upon the *ABC* caching strategy introduced in the previous chapter by characterising its main drawback: the uneven load it potentially places on particular nodes in the network. The chapter presents a comprehensive discussion of how load can be quantified in an information-centric IoT context, which serves as a basis to tackle the issue. The chapter addresses *ABC*'s load problem by extending the strategy with an Off-Path Caching (OPC) component designed to offset the potential load balancing issues implied by the centrality-based approach to caching. The new extension builds on the foundation of *ABC* while remaining modular and able to be applied to any other strategy. The new combined strategy, called *ABC+OPC*, performs satisfactorily and is able to successfully mitigate the negative load effects exhibited by pure *ABC* while only suffering from minor increases in content delivery latency.
- Chapter 6 discussed an intuitive approach to enabling Quality of Service (QoS) service classes in information-centric ICN. Particular attention is given to the impact of caching on such a solution. Reasonable modifications to existing caching decision strategies and cache replacement policies are discussed. Furthermore, potential caveats in the form of starvation are addressed with workable solutions. A series of experiments is presented that shows that enabling QoS in information-centric ICN is feasible and beneficial while not breaking with the overarching goal of providing lightweight solutions for constrained devices.

7.2 Contributions

The contributions of the thesis are summarised below:

- A comprehensive study of caching performance metrics in information-centric IoT was designed and carried out entirely on real, physical IoT hardware. At the time of writing, this is the first published study to cover such a wide range of caching strategies and the first overall study to examine caching in IoT on real hardware.
- A further study was presented that characterised content delivery latency and identified hitherto undiscovered topology effects that had a significant impact on the performance of caching strategies in information-centric IoT.
- The new *cache access factor* metric was proposed. This metric combines the cache hit rate and the hop reduction rate into a single weighted metric.
- *ABC* was proposed, a new, lightweight centrality-based caching strategy for information-centric IoT that is explicitly aimed at mitigating the topology effects discovered previously while being optimised towards deployment on limited IoT hardware.
- *ABC+OPC* was proposed, an extension to *ABC* that improves its performance and mitigates load issues observed in the vanilla implementation.
- QoS extensions were proposed that enable differentiated treatment of traffic flows of varying priorities. The proposed scheme is simple but effective and leads to notable performance increases.

7.3 Limitations and Future Work

The presented work still has a number of limitations that present opportunities for future research. In several places, only one of multiple possible avenues of research was considered, where ideally all possible solutions should be explored for their viability. This includes using other measures of centrality such as degree or eigenvector for the caching decision (see Section 4.1) and exploring alternative approaches to off-path caching. In fact, while the pre-

sented work focuses heavily on the centrality-based approach to topology-agnostic caching, the range of possible caching solution extends far beyond this narrow slice of strategies. Ideally, all of the families of caching strategies shown in the taxonomy in Figure 2.5 should be afforded the same level of scrutiny in order to find the best possible solution.

As briefly stated in Section 5.5, while the presented off-path caching solution was designed to address the load-balancing issues inherent to *ABC*, it is in no way limited to it and could be applied to any other caching strategy. Exploring the interactions of this solution with other strategies may shed light on other caching and load balancing phenomena that were not considered in this work.

The rest of this section will give an overview of several avenues of future research that are left open by this thesis.

- **Evaluating other approaches to off-path caching.** Section 5.3 briefly touched on the fact that OPC could be implemented using a number of different approaches. In the end, just as when developing *ABC*, the simplest approach was chosen for this thesis in order to keep overheads as low as possible and to show that satisfactory results can be achieved in information-centric IoT with comparatively little effort. However, this is not to say that performance improvements are out of the question. Even when keeping to the *Targeted Offloading (TO)* approach, the heuristics for which neighbour to choose for offloading could be expanded to include more factors, such as relative content popularities, request distributions, knowledge about neighbours' cache contents, or energy (see above). Furthermore, the feasibility of other, more cooperative offloading implementations should be explored, such as the idea mentioned in Section 5.3 that would allow the offloading node's neighbours to decide whether to cache the content.
- **Integrating energy considerations into *ABC*.** One of the negative side effects of uneven load in a battery-powered IoT is that individual nodes may run out of power at very different rates. A node that is receiving, processing, and forwarding significantly more packets than its neighbours will have to have its batteries replaced much sooner. Depending on the environment the network is deployed in, battery replacement may pose a significant effort. Therefore, there may be significant financial incentive to avoid situations where only a small number of nodes is at critical battery levels, as opposed

to being able to schedule regular replacements of all batteries at longer intervals. While this problem is partially addressed by *ABC+OPC*'s equalisation of load, *ABC* itself could also be extended to include energy considerations directly in its caching decisions. For example, instead of simply always caching at the node with the highest centrality, a combined metric of centrality and remaining energy could be introduced that aims to cache content at the "healthiest" nodes while still maintaining content accessibility through centrality. Furthermore, OPC might also be extended to include energy considerations, where offloading targets are not simply chosen by centrality but also by energy levels, thereby directing traffic to the healthiest nodes in the neighbourhood and equalising the overall power consumption.

- **Other open issues facing information-centric IoT**. While caching on constrained devices is a major open question, it is not the only hurdle facing the implementation of ICN solutions in the IoT space. Another way in which IoT communications differ from the traditional Internet is the fact that the packet MTU is usually much smaller. While the Ethernet standard defines a fixed MTU of 1500 bytes, common IoT link layer technologies typically have MTU sizes of less than a tenth of this (e.g. 127 bytes for IEEE 802.15.4 [1] or even just 27 bytes for Bluetooth Low Energy up to protocol version 4.1 [26]). This has several implications for ICN functionality. The first is that the hierarchical, human-readable names traditionally used in ICN may not be feasible in IoT as a single packet might not be able to fit a whole name. Furthermore, long names can result in a processing bottleneck as string comparisons with variable lengths are typically among the most costly operations, especially on constrained hardware [21]. Therefore, a rethinking of the naming approach towards compressed, machine-readable names may be in order. Furthermore, the small MTU may necessitate fragmentation, particularly of Data packets. This raises further questions that can be connected back to caching: How can efficient caching be achieved if content chunks are fragmented across different content stores?
- **Testing and evaluation in real-world IoT deployments**. While the IoT-LAB testbed used for the experiments in this thesis provides very realistic testing conditions, it still does not cover the whole range of possible IoT scenarios. There are deployments that have to deal which much more prohibitive external factors, such as battery-powered

nodes in locations that make physical access almost impossible. Deployments in hostile environments, such as underwater or space networks, also exist or are conceivable. While it may never be possible to find an ICN solution that fits all deployments, there is definitely more room for research and development encompassing more complex scenarios.

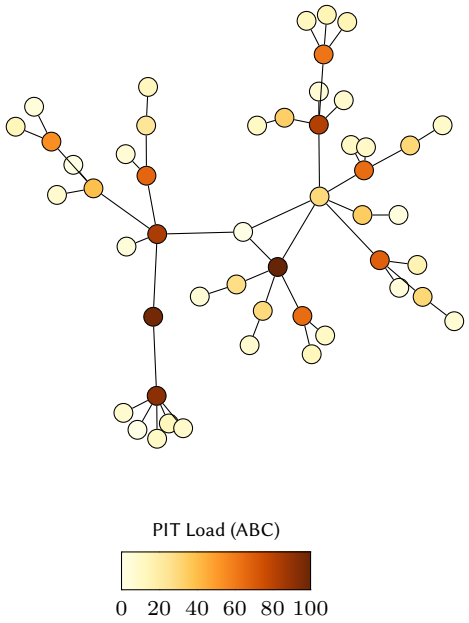
APPENDIX A

Alternative Experiment Scenarios for Load Characteristics and Off-Path Caching

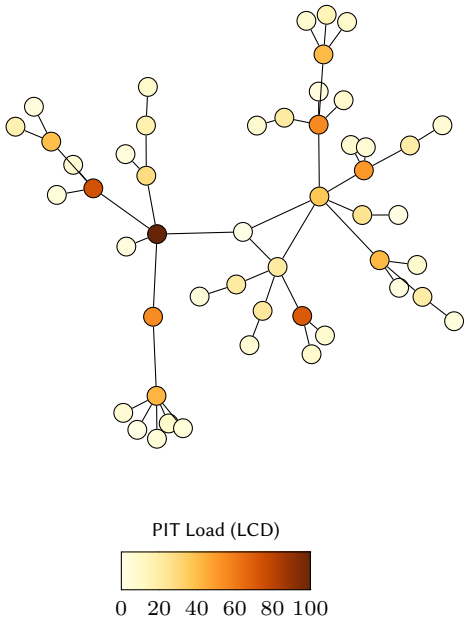
In Chapter 5, experiments to determine the load characteristics of *ABC* and the performance improvements of *ABC+OPC* were performed on a specific topology with specific experiment parameters. The main scenario featured a full mesh between the five most central nodes, with five nodes being randomly chosen as producers for each experiment run.

An alternative topology was considered in which the core was not fully meshed. Instead, all connections between the subtrees of the topology would run through the logical centre of the topology. This would severely increase the load on that central node, potentially exacerbating the effects discussed in Section 5.2. However, since it would be easier to approach this problem by meshing the core than by introducing OPC, it was decided to perform the main evaluation on a topology that already featured a meshed core.

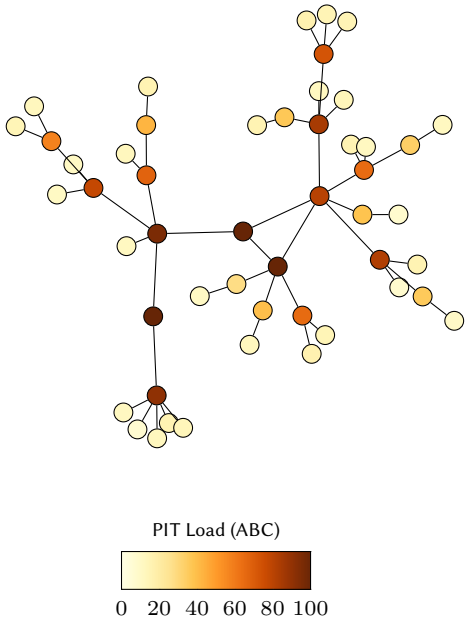
An alternative experiment setup was further considered in which there was only a single producer, chosen randomly for each experiment run. This would have the effect of consolidating content request paths across the network, as all FIBs would point towards the same node. However, the diversity of content requests would be decreased. Although these alternative experiment scenarios were not discussed in Chapter 5 for the reasons stated above, their results are shown here for the sake of completeness and comparison.



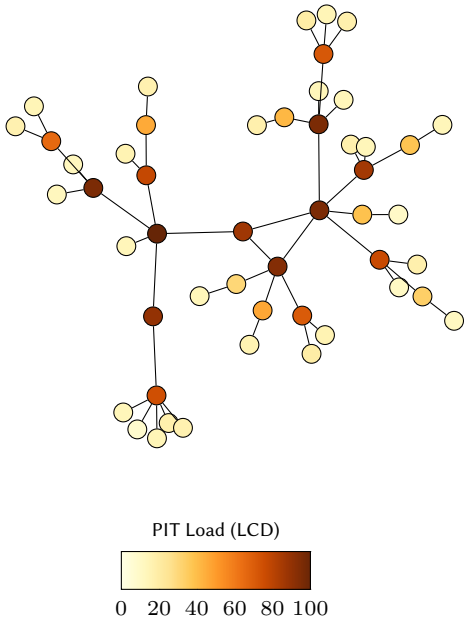
(a) ABC, Non-Mesh Topology, Single Producer



(b) LCD, Non-Mesh Topology, Single Producer

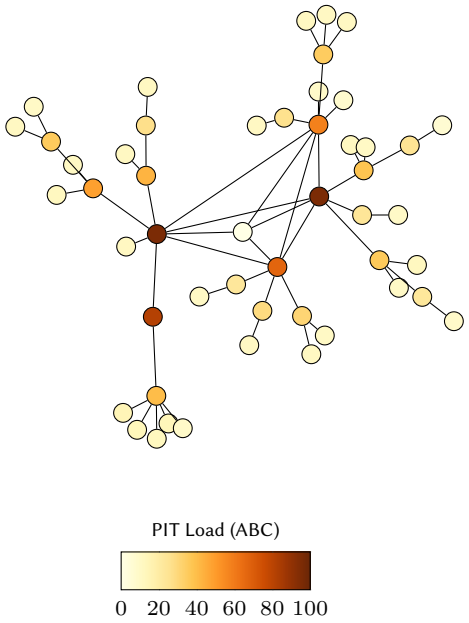


(c) ABC, Non-Mesh Topology, Multi Producer

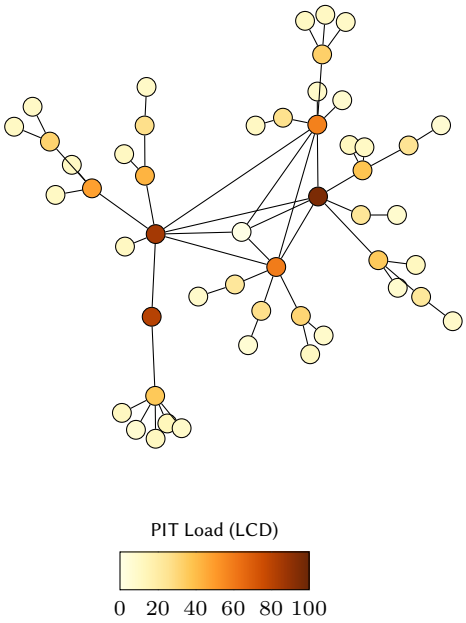


(d) LCD, Non-Mesh Topology, Multi Producer

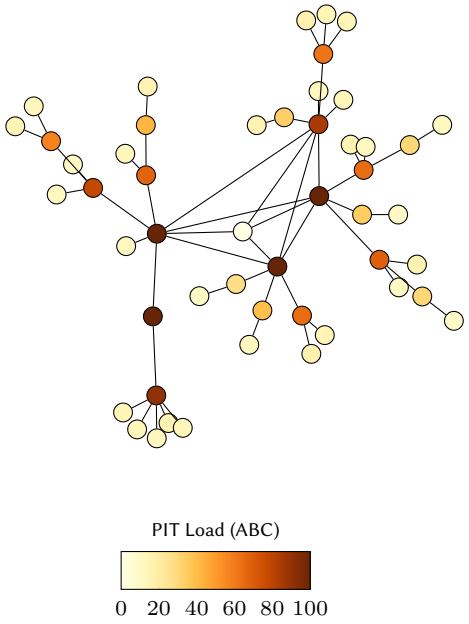
Figure A.1.: PIT Load: Non-Mesh topology



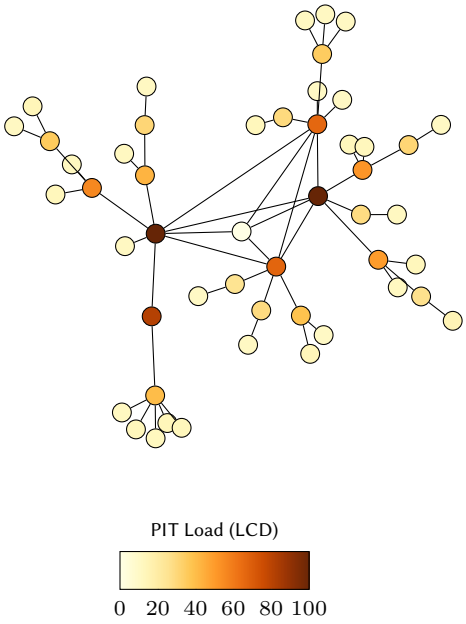
(a) ABC, Mesh Topology, Single Producer



(b) LCD, Mesh Topology, Single Producer

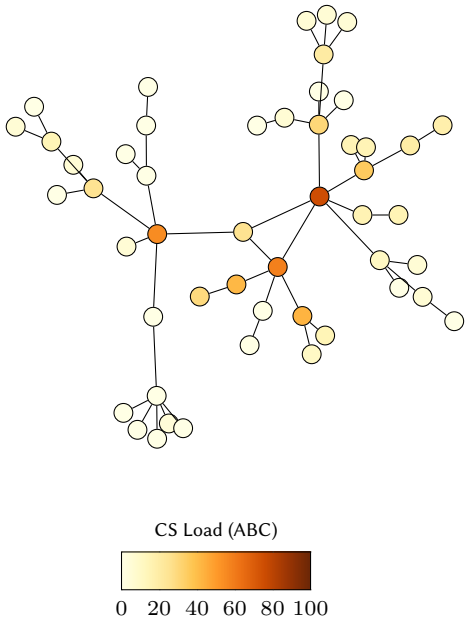


(c) ABC, Mesh Topology, Multi Producer

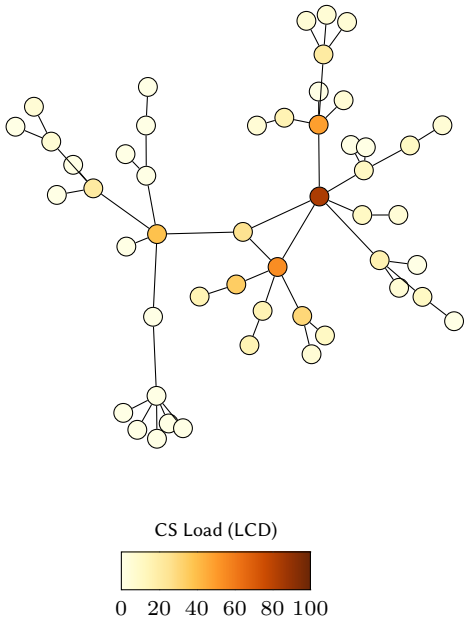


(d) LCD, Mesh Topology, Multi Producer

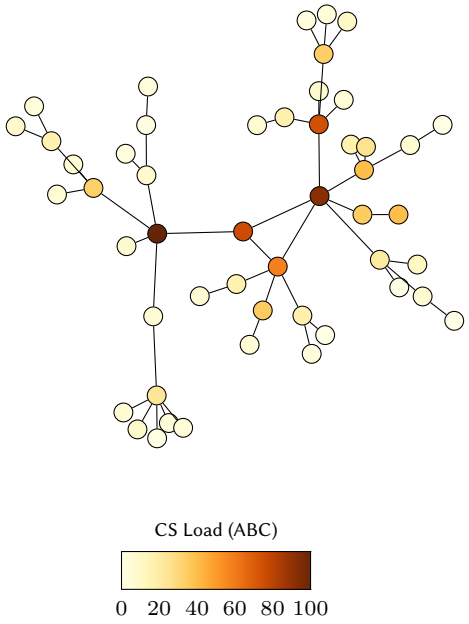
Figure A.2.: PIT Load: Mesh topology



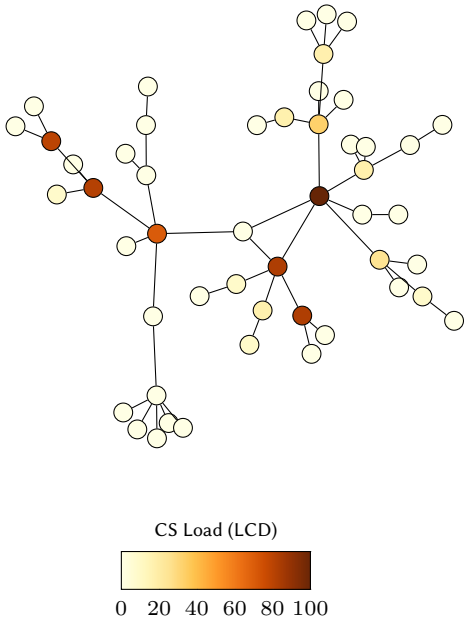
(a) ABC, Non-Mesh Topology, Single Producer



(b) LCD, Non-Mesh Topology, Single Producer

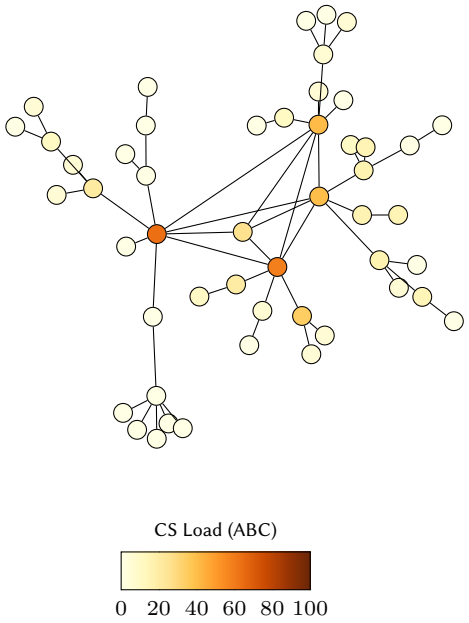


(c) ABC, Non-Mesh Topology, Multi Producer

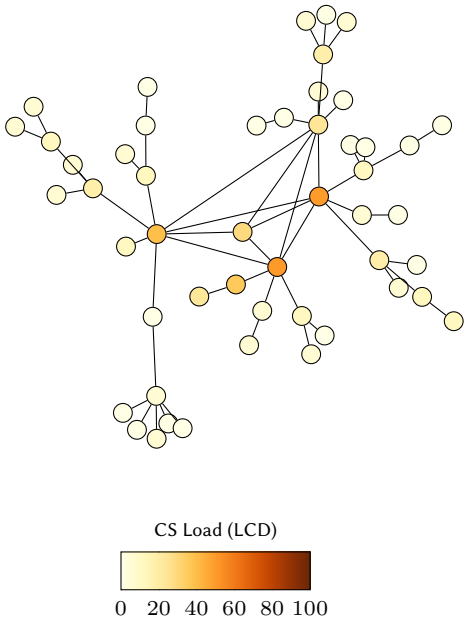


(d) LCD, Non-Mesh Topology, Multi Producer

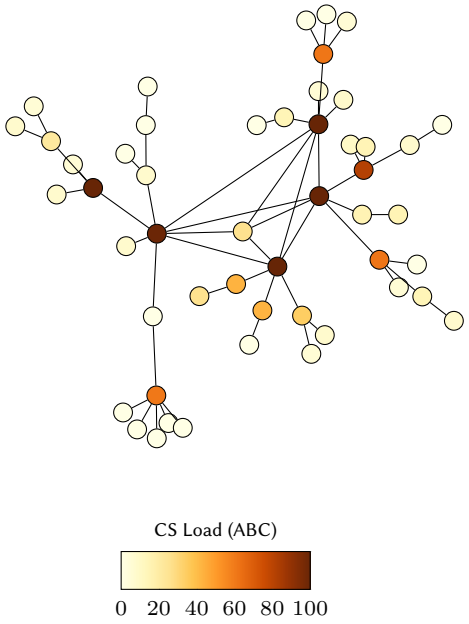
Figure A.3.: CS Load: Non-Mesh topology



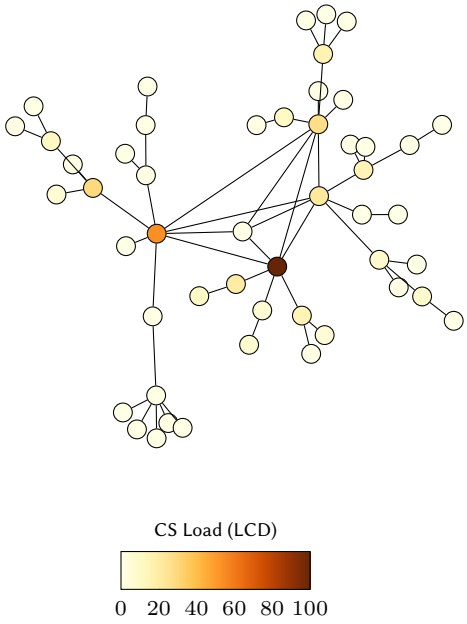
(a) ABC, Mesh Topology, Single Producer



(b) LCD, Mesh Topology, Single Producer

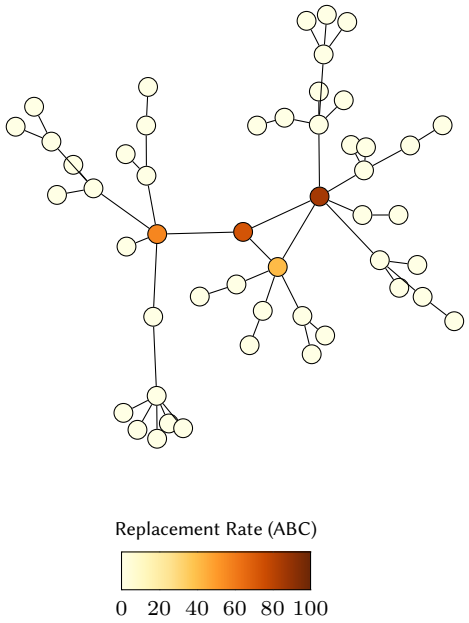


(c) ABC, Mesh Topology, Multi Producer

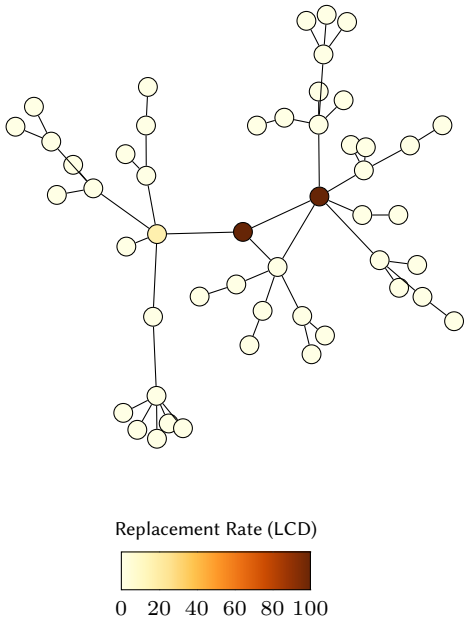


(d) LCD, Mesh Topology, Multi Producer

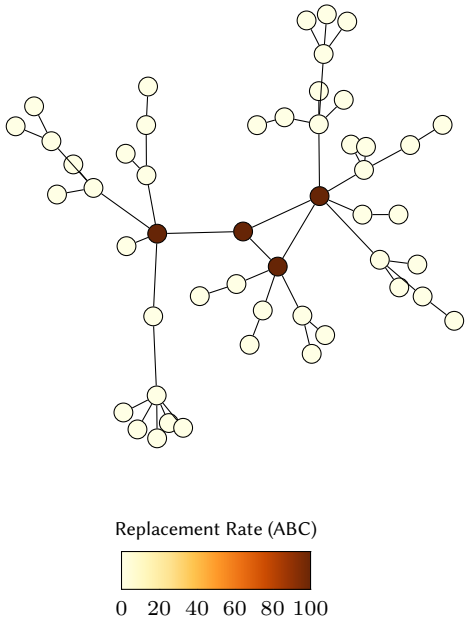
Figure A.4.: CS Load: Mesh topology



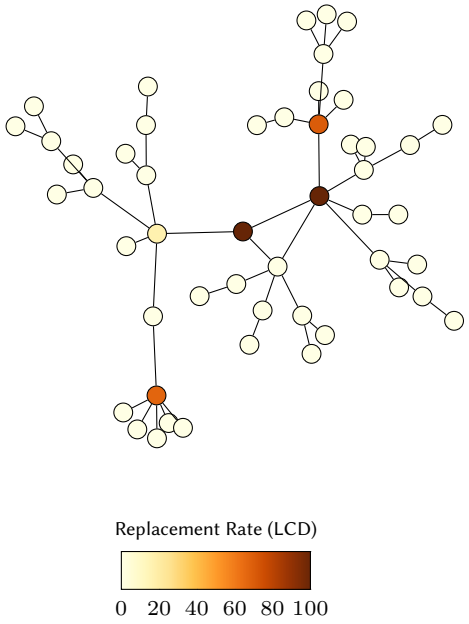
(a) ABC, Non-Mesh Topology, Single Producer



(b) LCD, Non-Mesh Topology, Single Producer

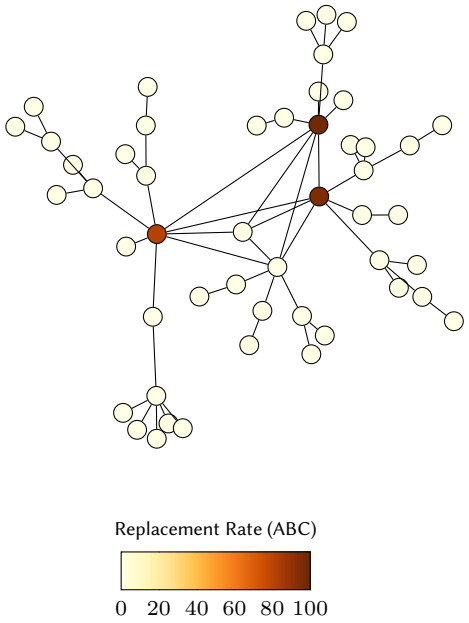


(c) ABC, Non-Mesh Topology, Multi Producer

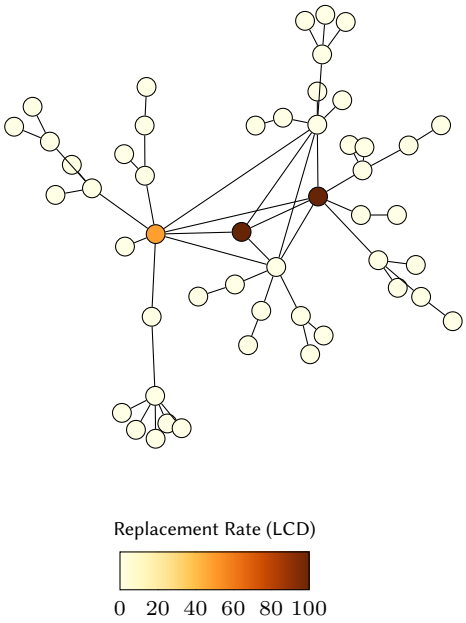


(d) LCD, Non-Mesh Topology, Multi Producer

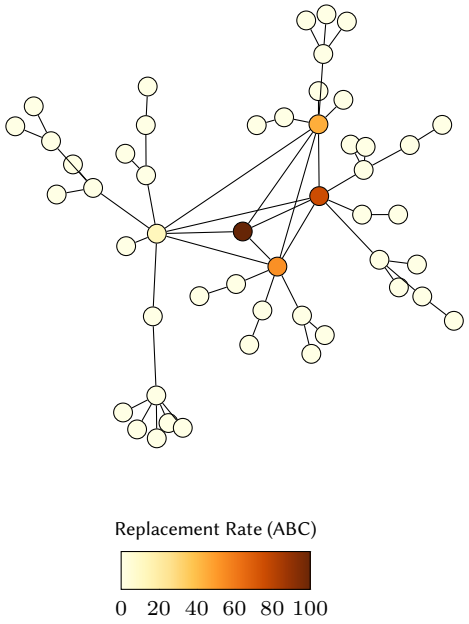
Figure A.5.: CS Replacement Rate: Non-Mesh topology



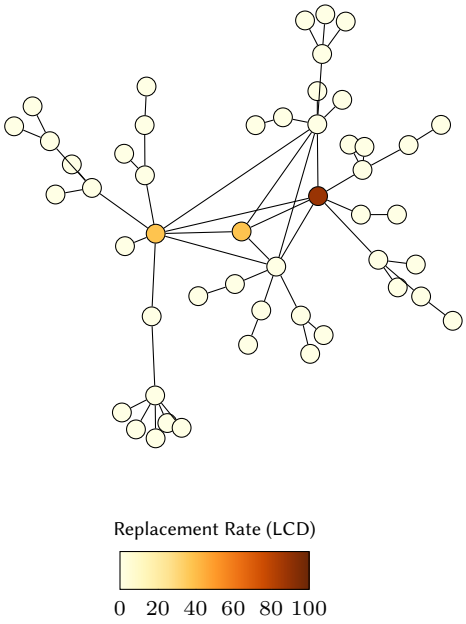
(a) ABC, Mesh Topology, Single Producer



(b) LCD, Mesh Topology, Single Producer

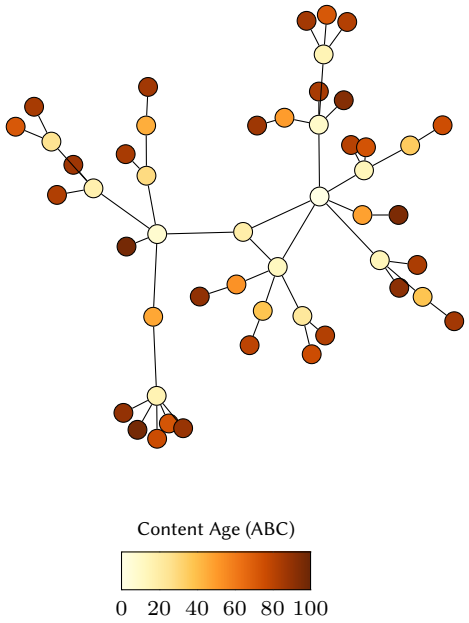


(c) ABC, Mesh Topology, Multi Producer

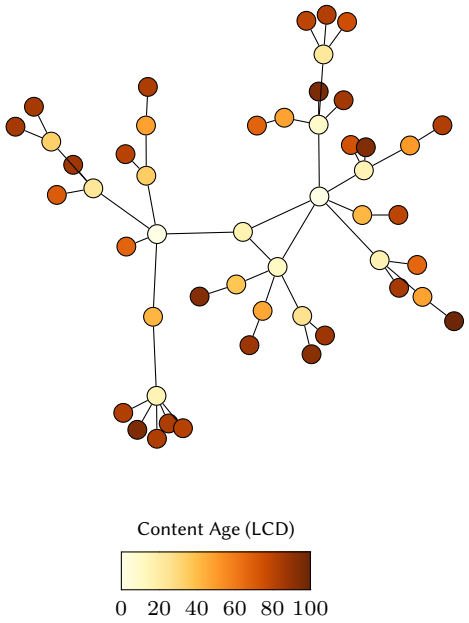


(d) LCD, Mesh Topology, Multi Producer

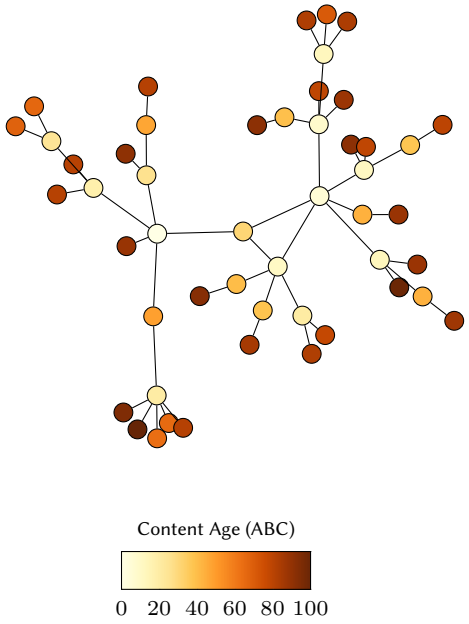
Figure A.6.: CS Replacement Rate: Mesh topology



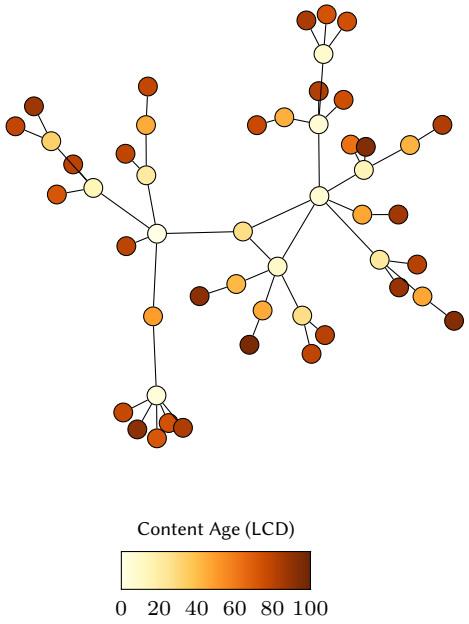
(a) ABC, Non-Mesh Topology, Single Producer



(b) LCD, Non-Mesh Topology, Single Producer

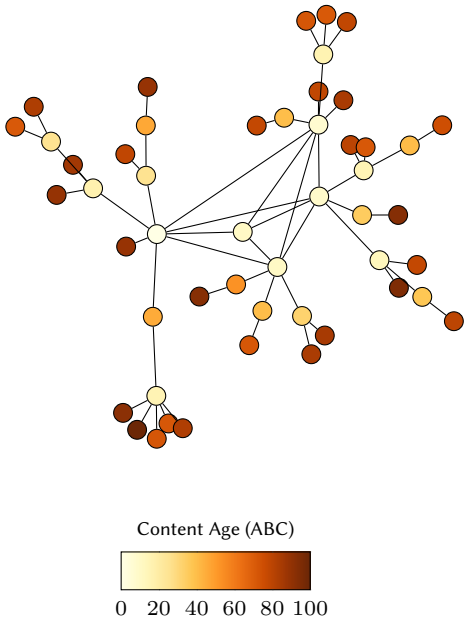


(c) ABC, Non-Mesh Topology, Multi Producer

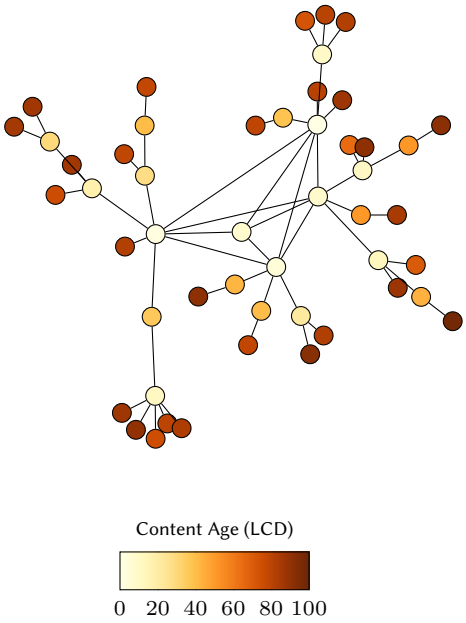


(d) LCD, Non-Mesh Topology, Multi Producer

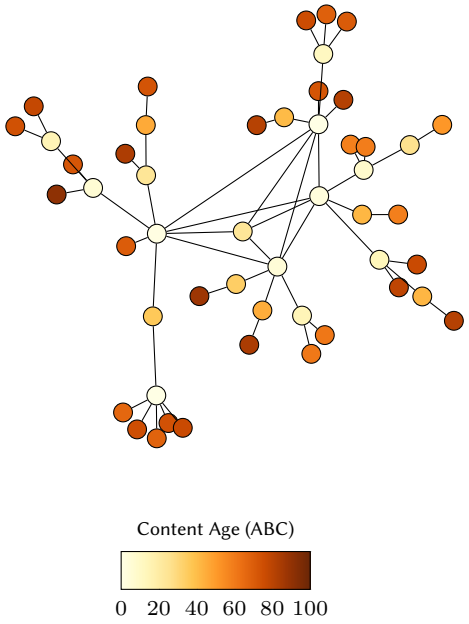
Figure A.7.: Average Content Age: Non-Mesh topology



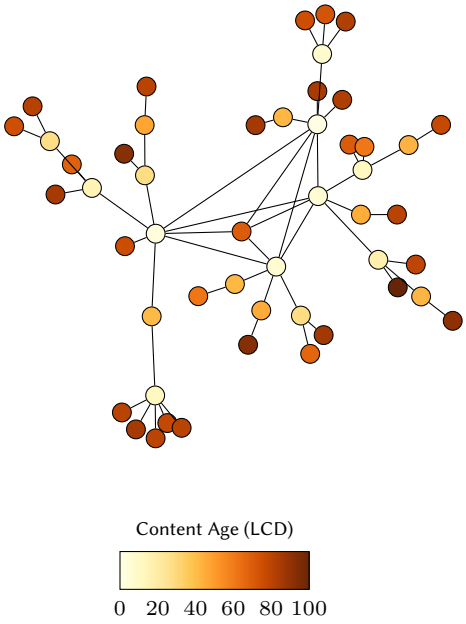
(a) ABC, Mesh Topology, Single Producer



(b) LCD, Mesh Topology, Single Producer

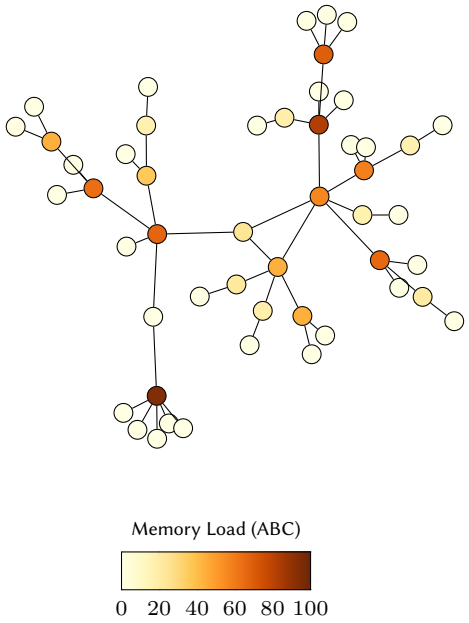


(c) ABC, Mesh Topology, Multi Producer

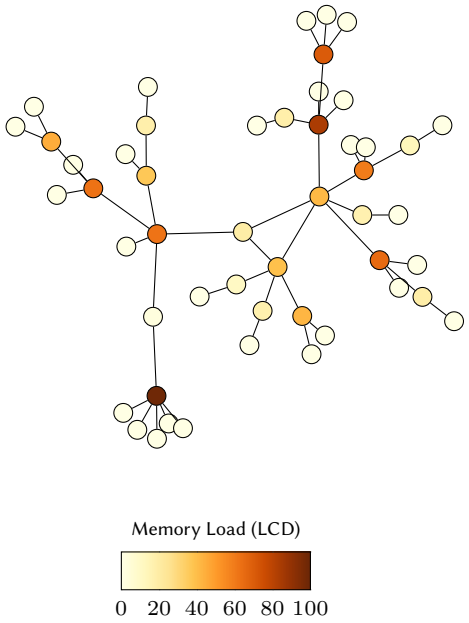


(d) LCD, Mesh Topology, Multi Producer

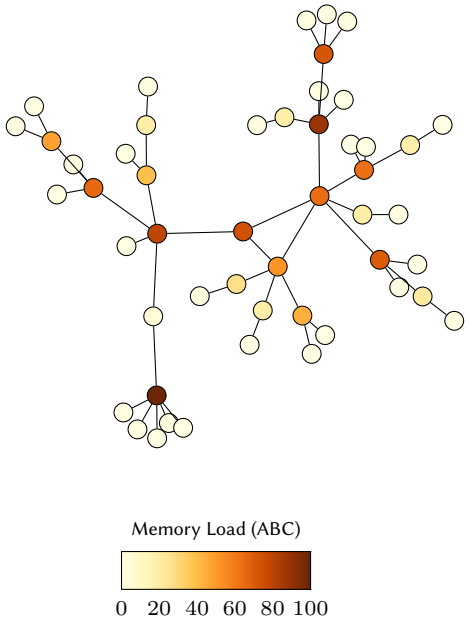
Figure A.8.: Average Content Age: Mesh topology



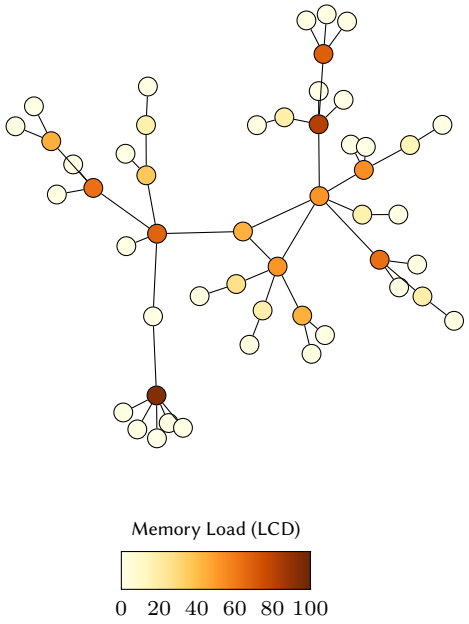
(a) ABC, Non-Mesh Topology, Single Producer



(b) LCD, Non-Mesh Topology, Single Producer

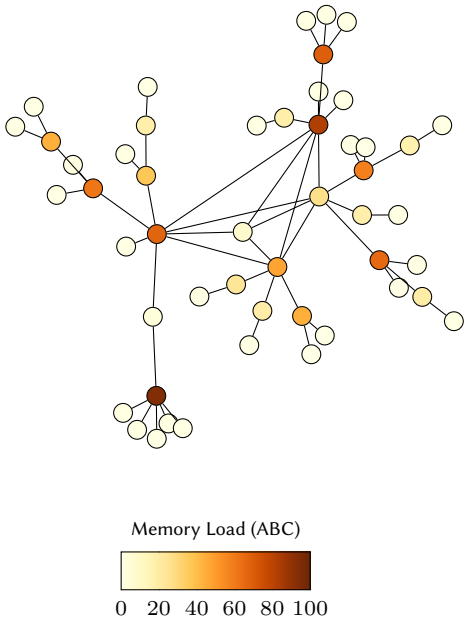


(c) ABC, Non-Mesh Topology, Multi Producer

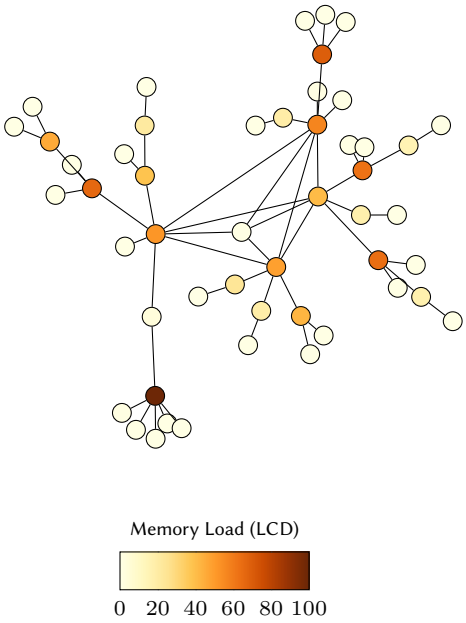


(d) LCD, Non-Mesh Topology, Multi Producer

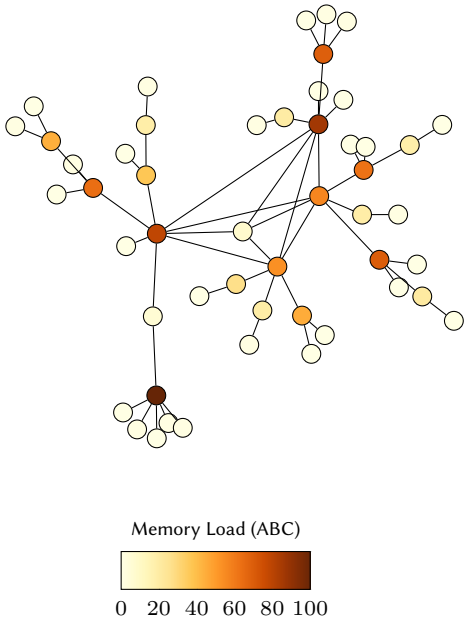
Figure A.9.: Memory Load: Non-Mesh topology



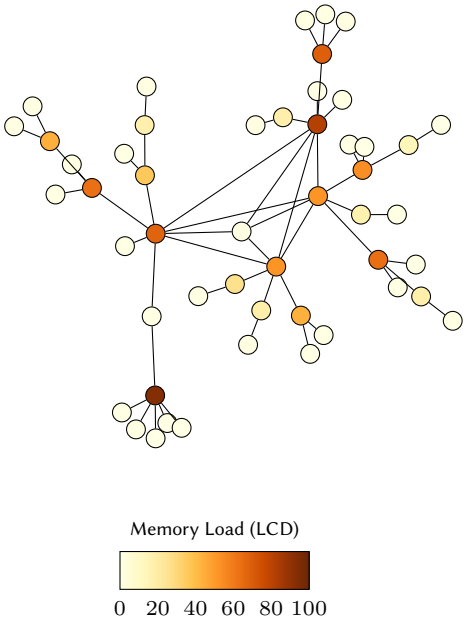
(a) ABC, Mesh Topology, Single Producer



(b) LCD, Mesh Topology, Single Producer

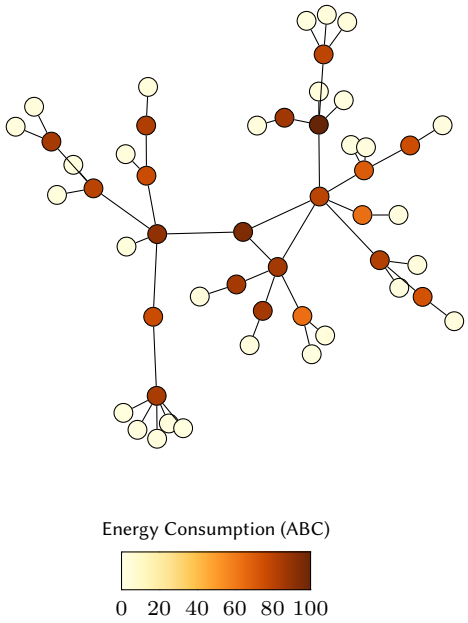


(c) ABC, Mesh Topology, Multi Producer

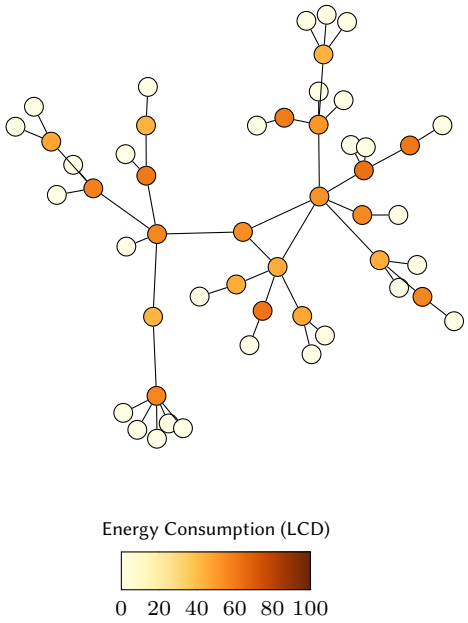


(d) LCD, Mesh Topology, Multi Producer

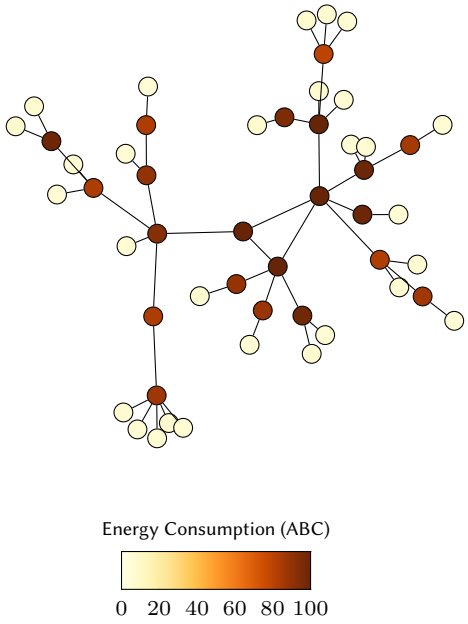
Figure A.10.: Memory Load: Mesh topology



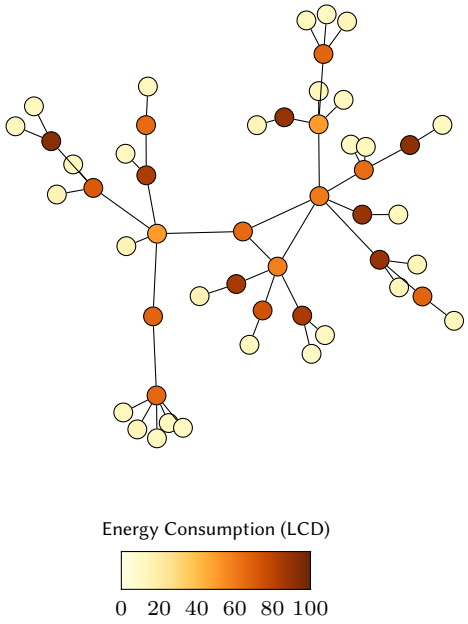
(a) ABC, Non-Mesh Topology, Single Producer



(b) LCD, Non-Mesh Topology, Single Producer

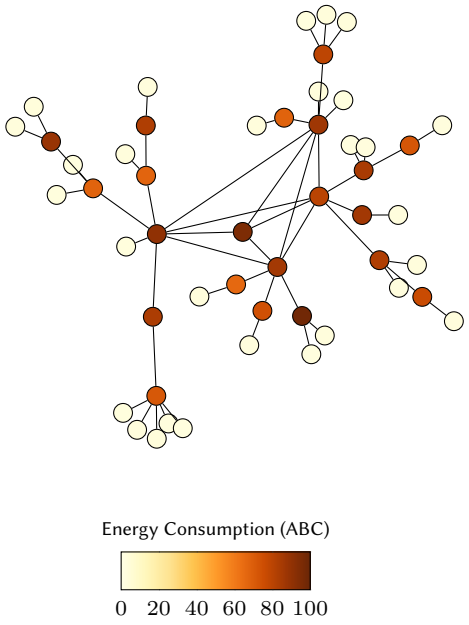


(c) ABC, Non-Mesh Topology, Multi Producer

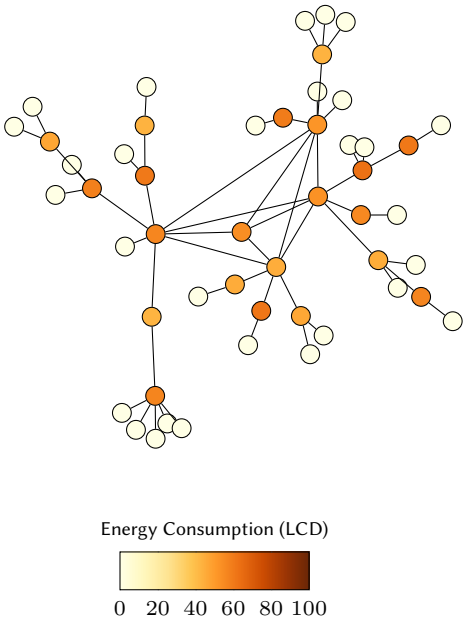


(d) LCD, Non-Mesh Topology, Multi Producer

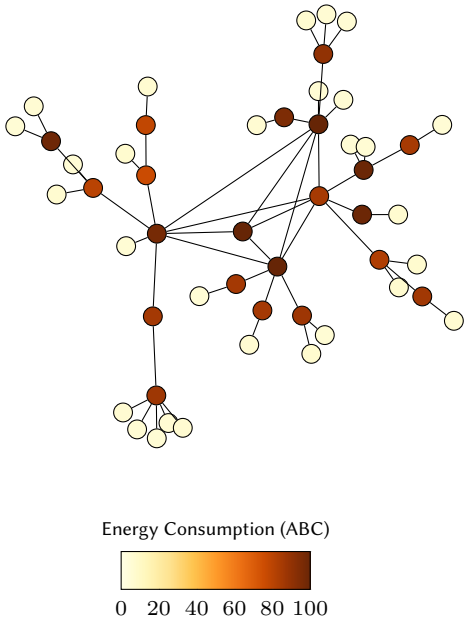
Figure A.11.: Energy consumption: Non-Mesh topology



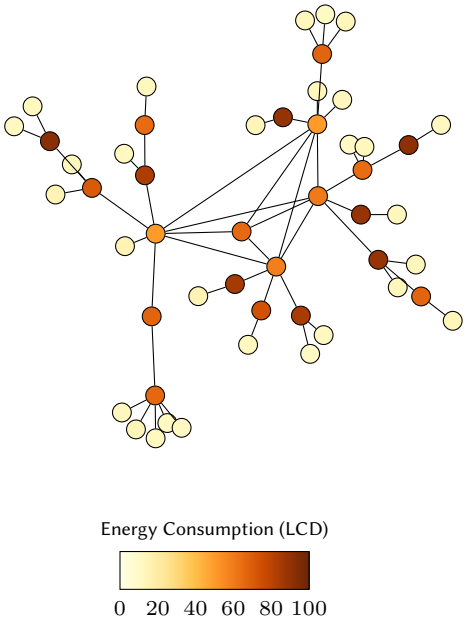
(a) ABC, Mesh Topology, Single Producer



(b) LCD, Mesh Topology, Single Producer

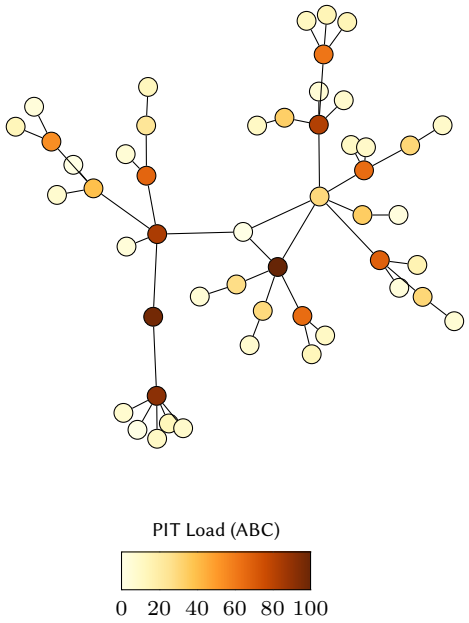


(c) ABC, Mesh Topology, Multi Producer

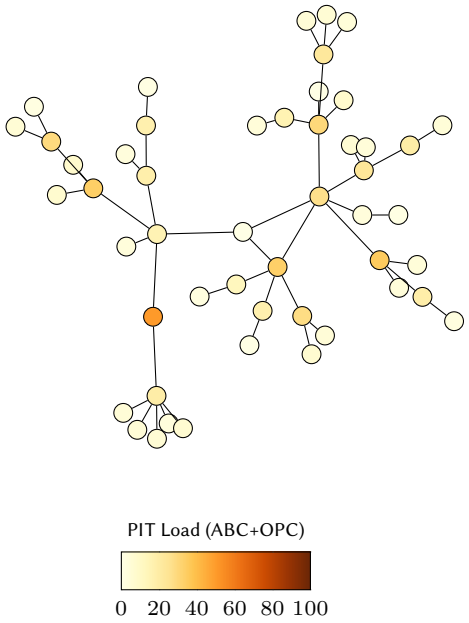


(d) LCD, Mesh Topology, Multi Producer

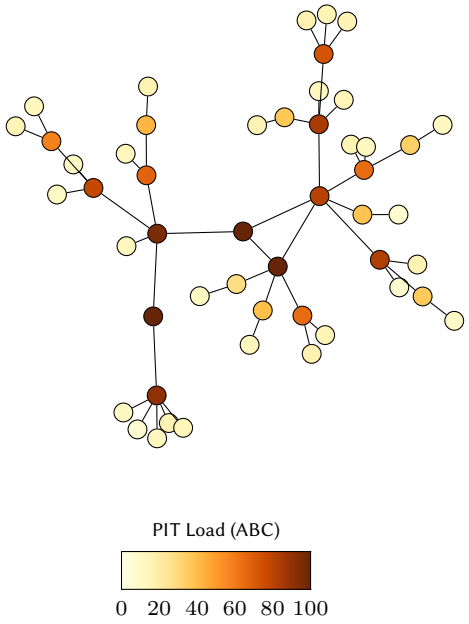
Figure A.12.: Energy consumption: Mesh topology



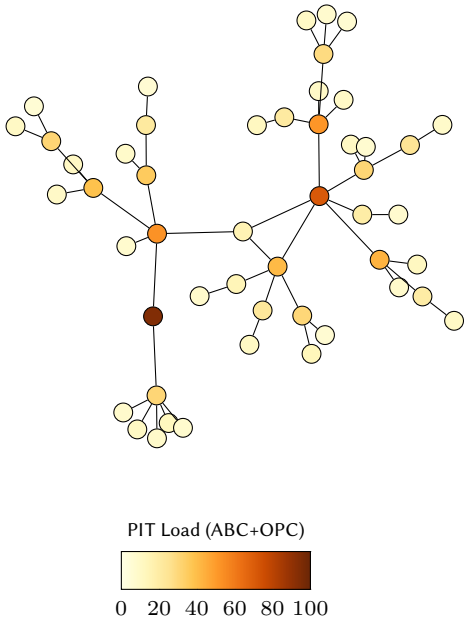
(a) ABC, Non-Mesh Topology, Single Producer



(b) ABC+OPC, Non-Mesh Topology, Single Producer

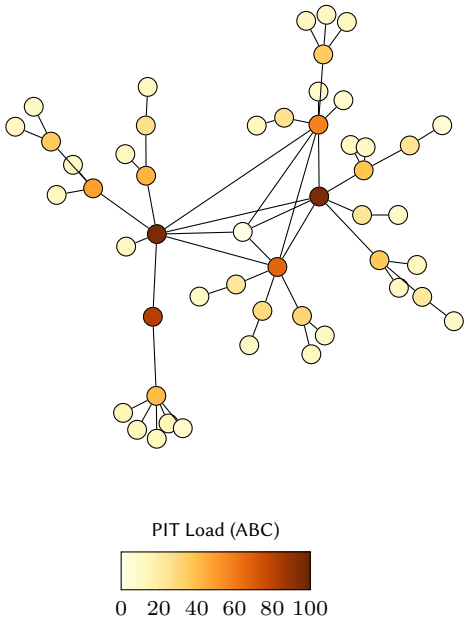


(c) ABC, Non-Mesh Topology, Multi Producer

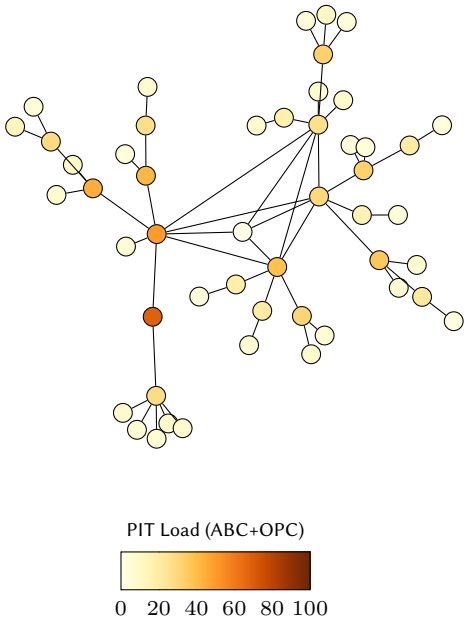


(d) ABC+OPC, Non-Mesh Topology, Multi Producer

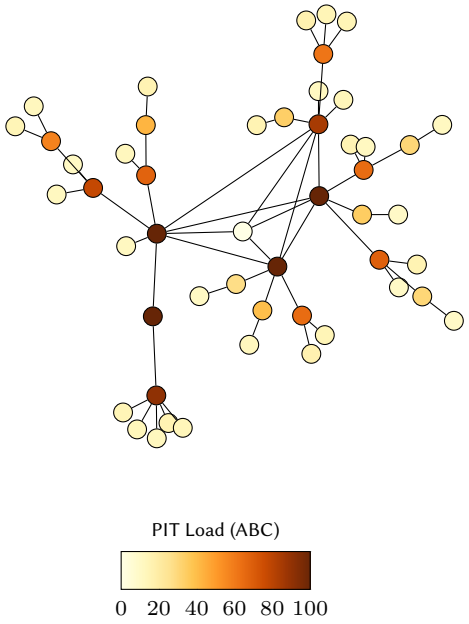
Figure A.13.: PIT load: ABC vs. ABC+OPC, Non-mesh topology



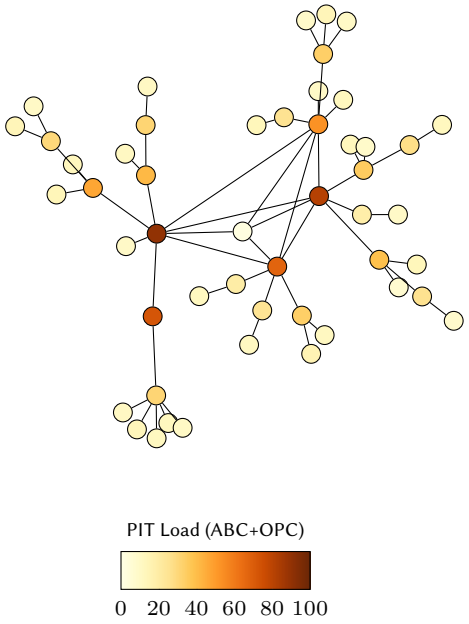
(a) ABC, Mesh Topology, Single Producer



(b) ABC+OPC, Mesh Topology, Single Producer

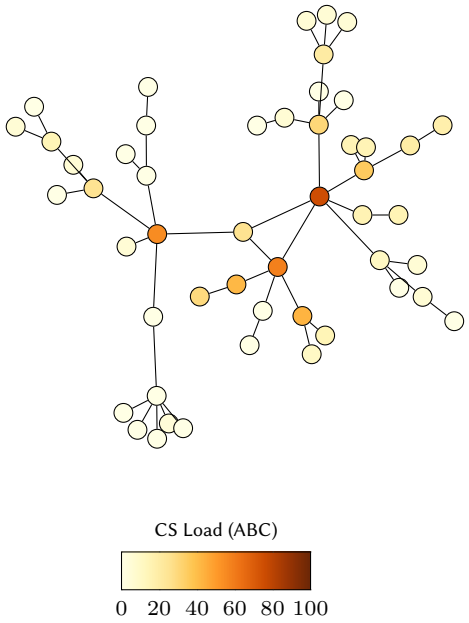


(c) ABC, Mesh Topology, Multi Producer

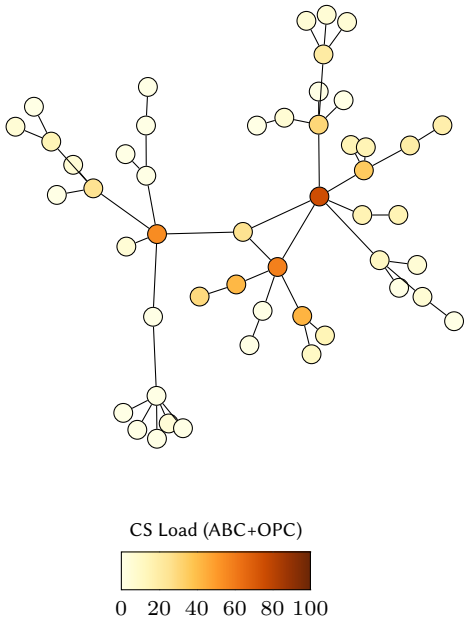


(d) ABC+OPC, Mesh Topology, Multi Producer

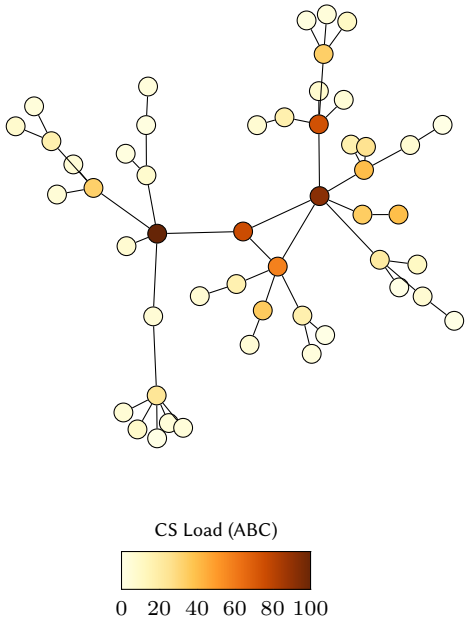
Figure A.14.: PIT load: ABC vs. ABC+OPC, Mesh topology



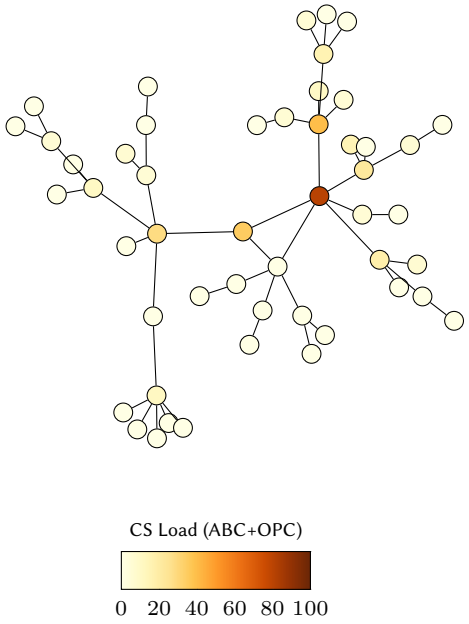
(a) ABC, Non-Mesh Topology, Single Producer



(b) ABC+OPC, Non-Mesh Topology, Single Producer

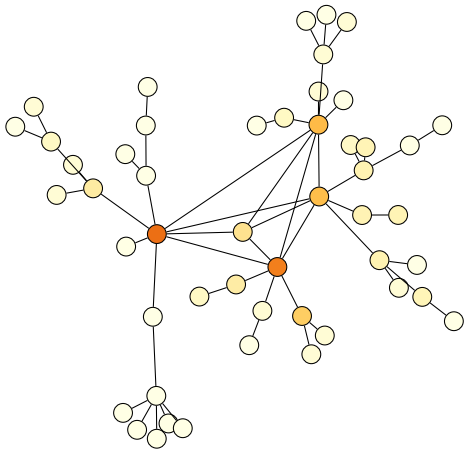


(c) ABC, Non-Mesh Topology, Multi Producer

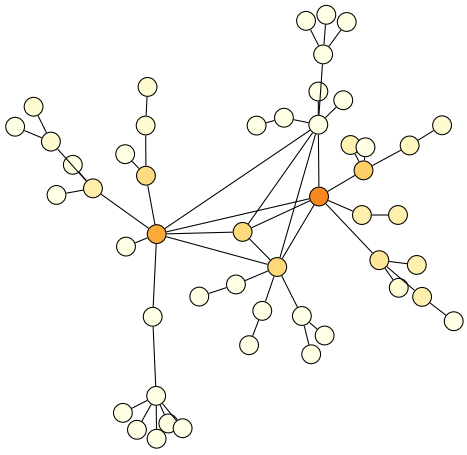


(d) ABC+OPC, Non-Mesh Topology, Multi Producer

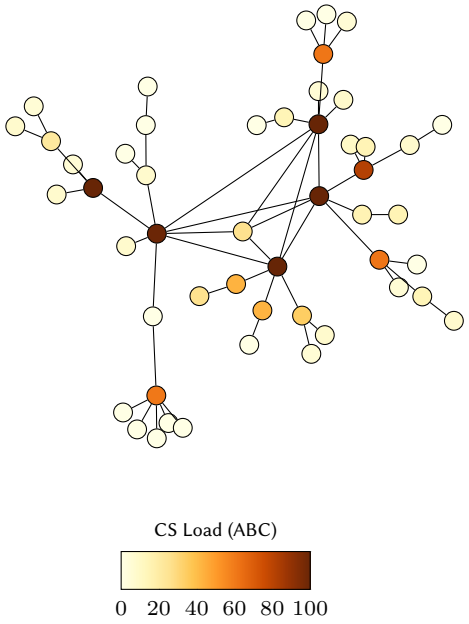
Figure A.15.: CS load: ABC vs. ABC+OPC, Non-mesh topology



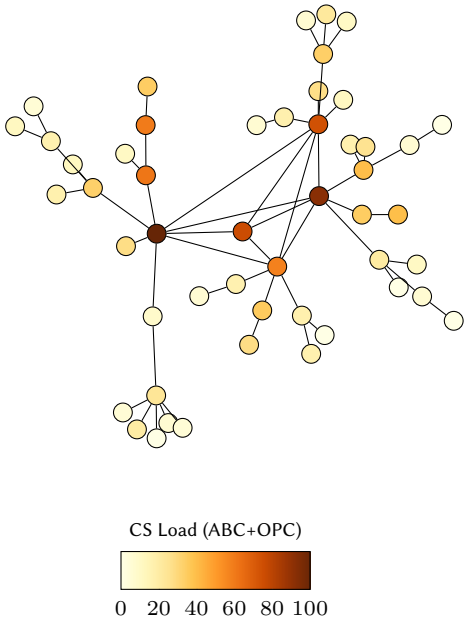
(a) ABC, Mesh Topology, Single Producer



(b) ABC+OPC, Mesh Topology, Single Producer

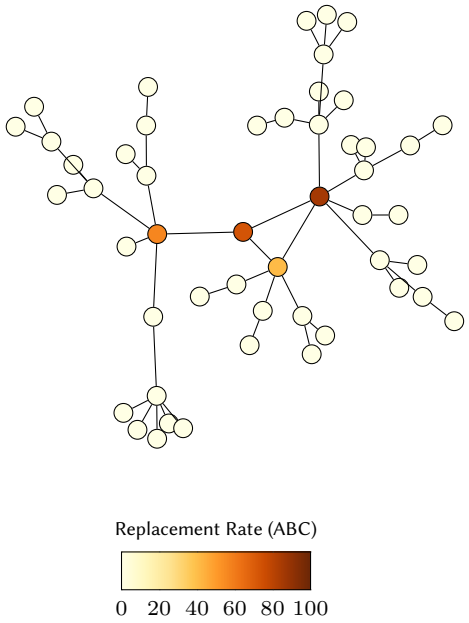


(c) ABC, Mesh Topology, Multi Producer

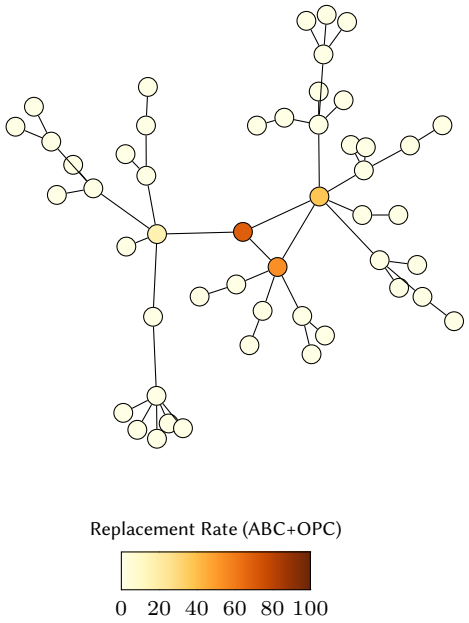


(d) ABC+OPC, Mesh Topology, Multi Producer

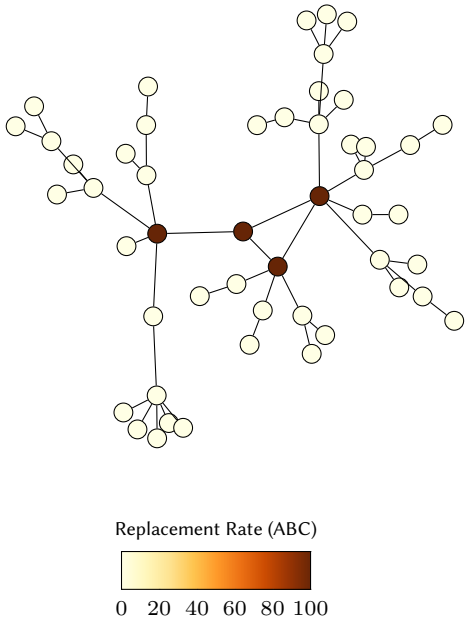
Figure A.16.: CS load: ABC vs. ABC+OPC, Mesh topology



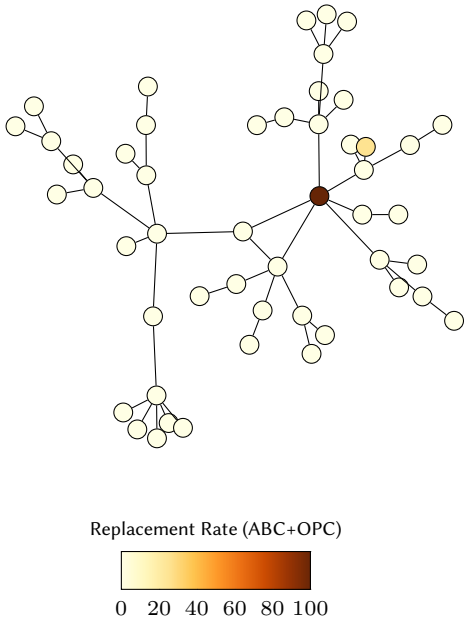
(a) ABC, Non-Mesh Topology, Single Producer



(b) ABC+OPC, Non-Mesh Topology, Single Producer

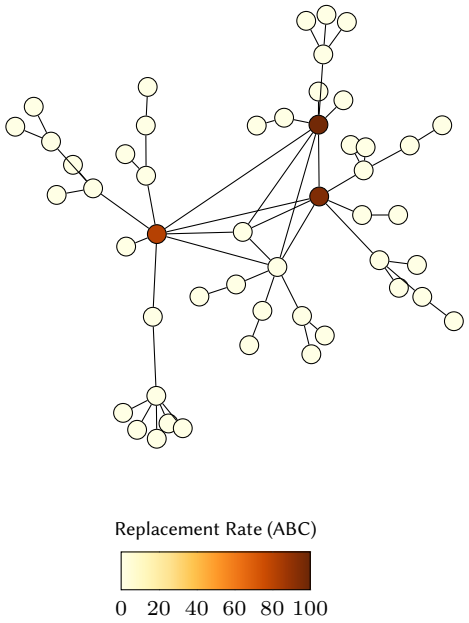


(c) ABC, Non-Mesh Topology, Multi Producer

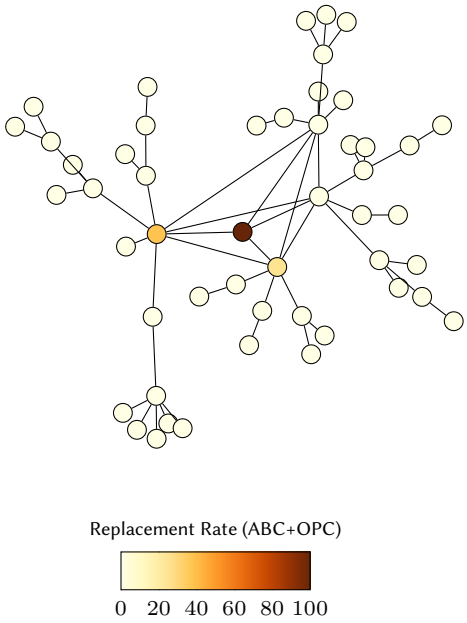


(d) ABC+OPC, Non-Mesh Topology, Multi Producer

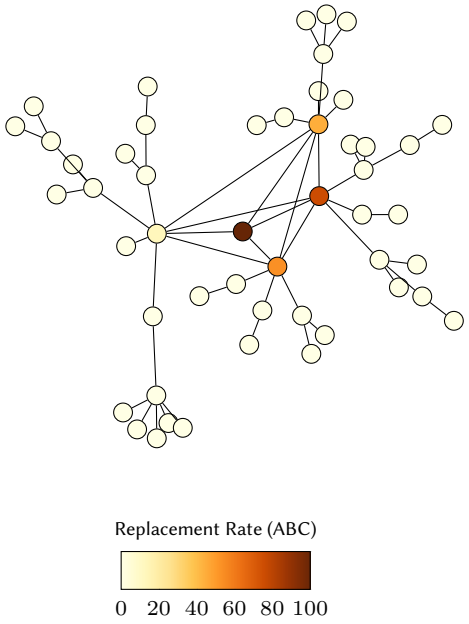
Figure A.17.: Replacement rate: ABC vs. ABC+OPC, Non-mesh topology



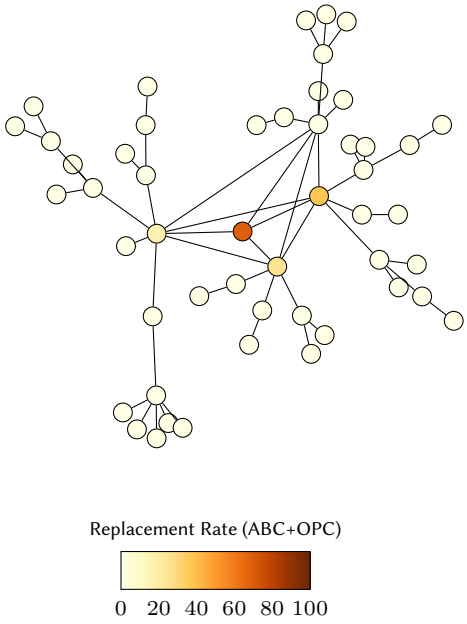
(a) ABC, Mesh Topology, Single Producer



(b) ABC+OPC, Mesh Topology, Single Producer

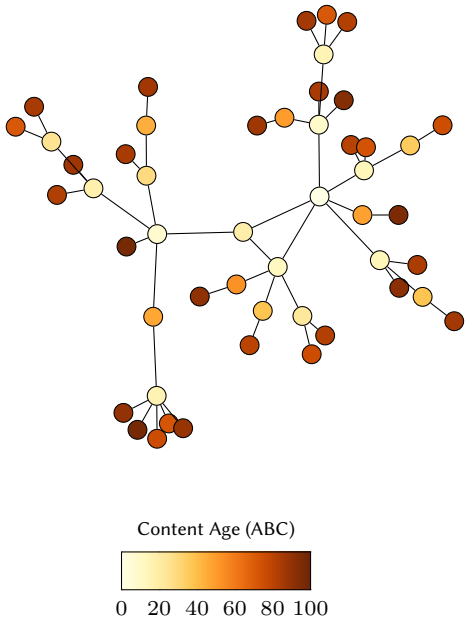


(c) ABC, Mesh Topology, Multi Producer

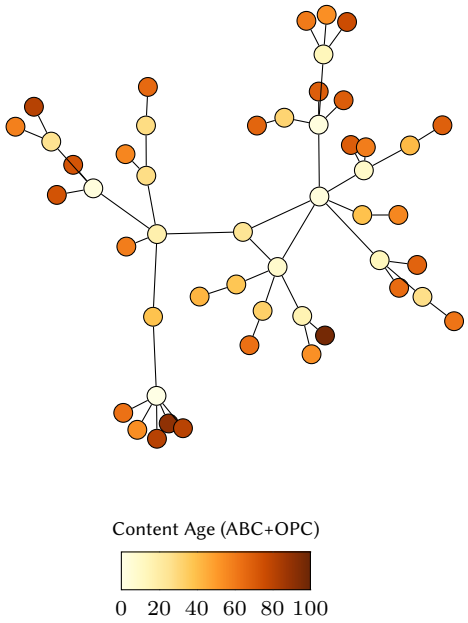


(d) ABC+OPC, Mesh Topology, Multi Producer

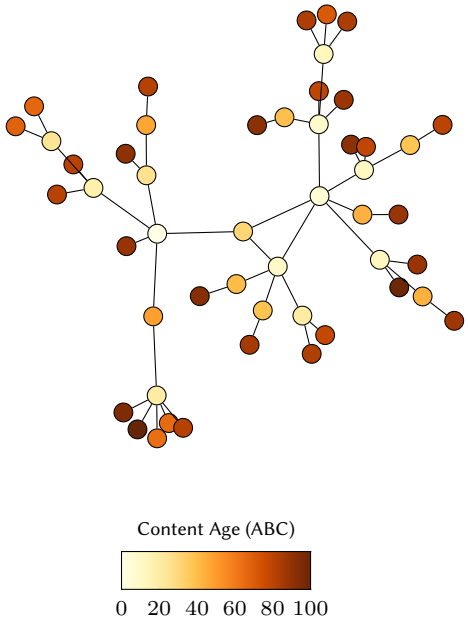
Figure A.18.: Replacement rate: ABC vs. ABC+OPC, Mesh topology



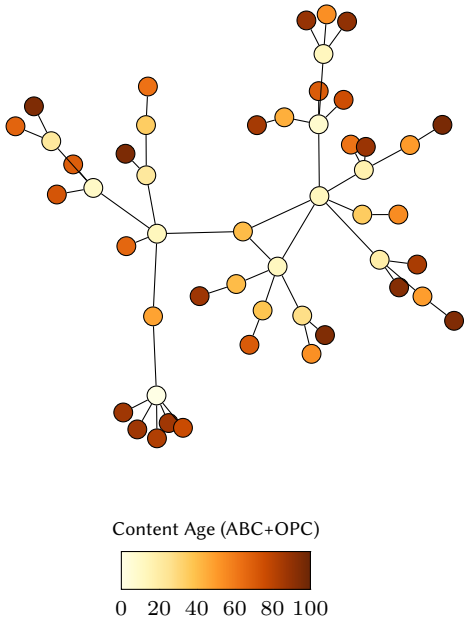
(a) ABC, Non-Mesh Topology, Single Producer



(b) ABC+OPC, Non-Mesh Topology, Single Producer

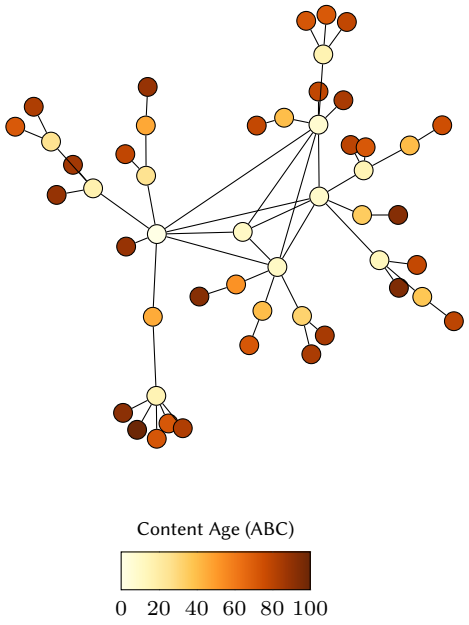


(c) ABC, Non-Mesh Topology, Multi Producer

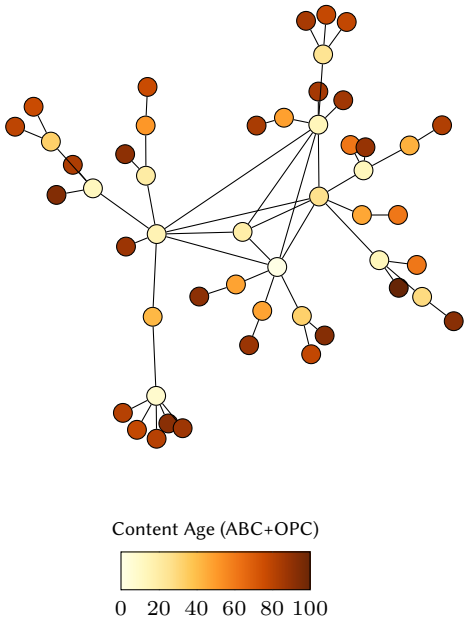


(d) ABC+OPC, Non-Mesh Topology, Multi Producer

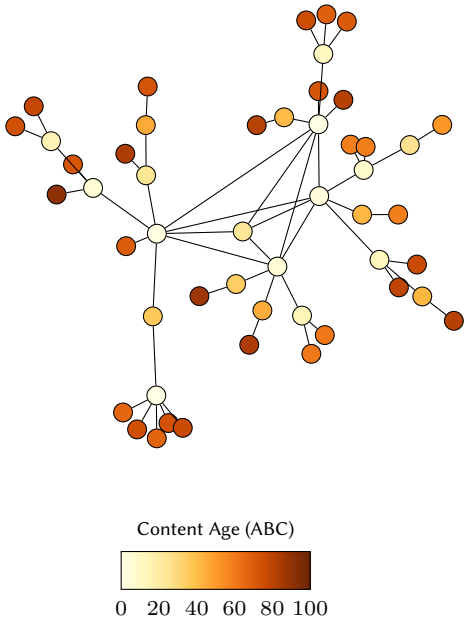
Figure A.19.: Average content age: ABC vs. ABC+OPC, Non-mesh topology



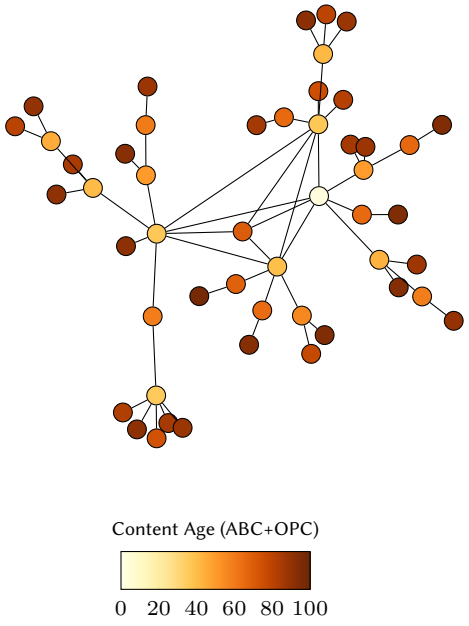
(a) ABC, Mesh Topology, Single Producer



(b) ABC+OPC, Mesh Topology, Single Producer

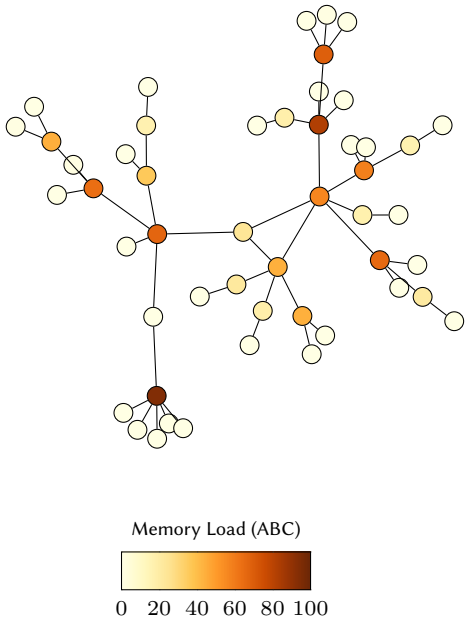


(c) ABC, Mesh Topology, Multi Producer

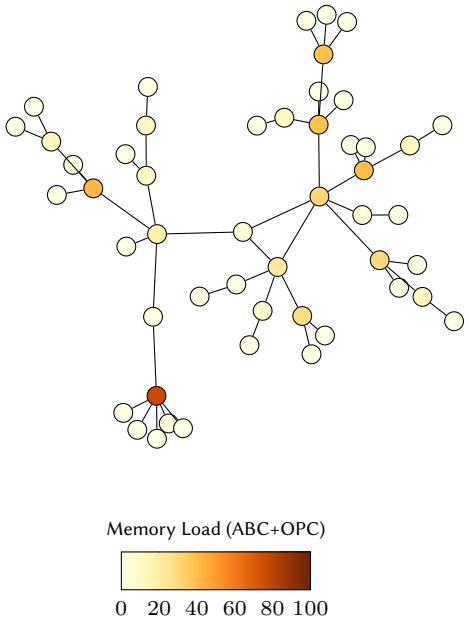


(d) ABC+OPC, Mesh Topology, Multi Producer

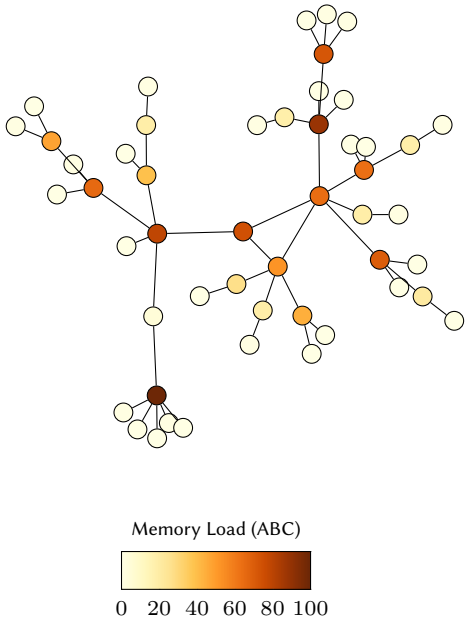
Figure A.20.: Average content age: ABC vs. ABC+OPC, Mesh topology



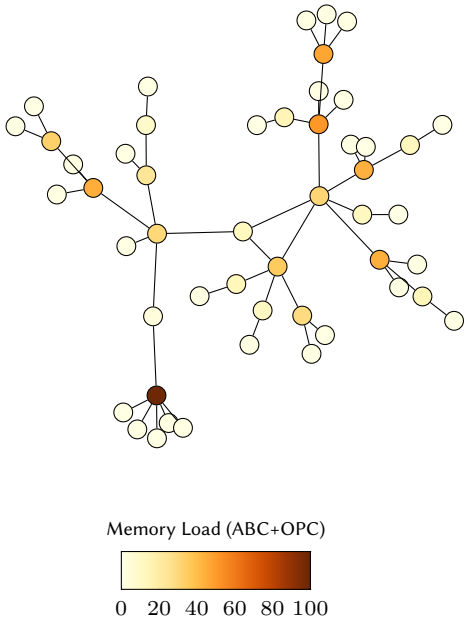
(a) ABC, Non-Mesh Topology, Single Producer



(b) ABC+OPC, Non-Mesh Topology, Single Producer

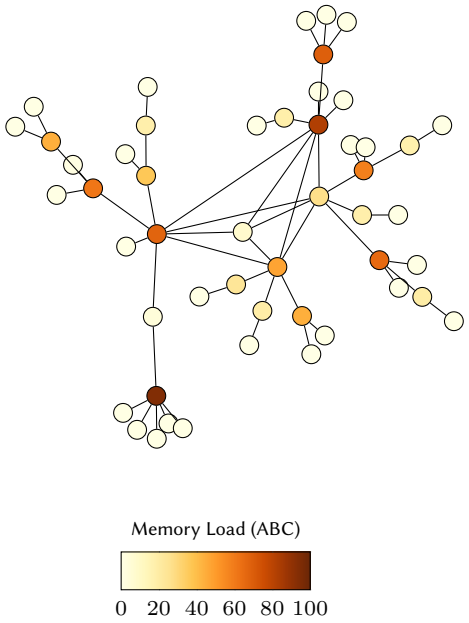


(c) ABC, Non-Mesh Topology, Multi Producer

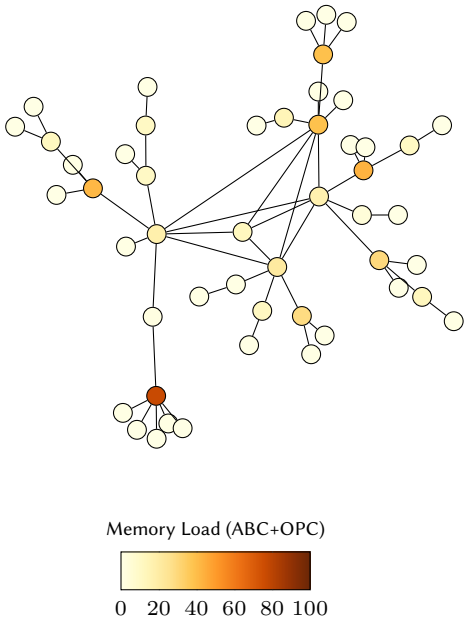


(d) ABC+OPC, Non-Mesh Topology, Multi Producer

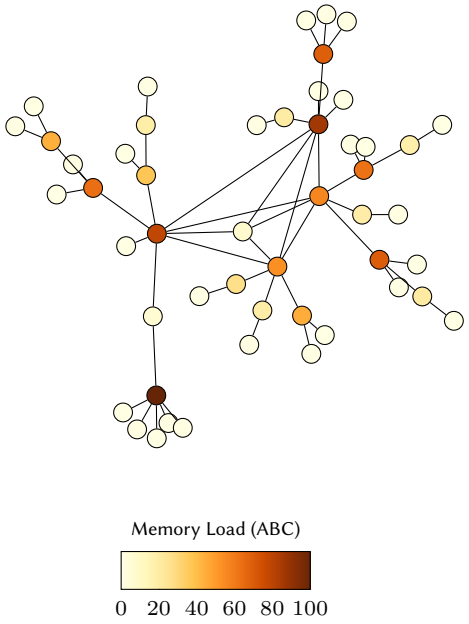
Figure A.21.: Memory load: ABC vs. ABC+OPC, Non-mesh topology



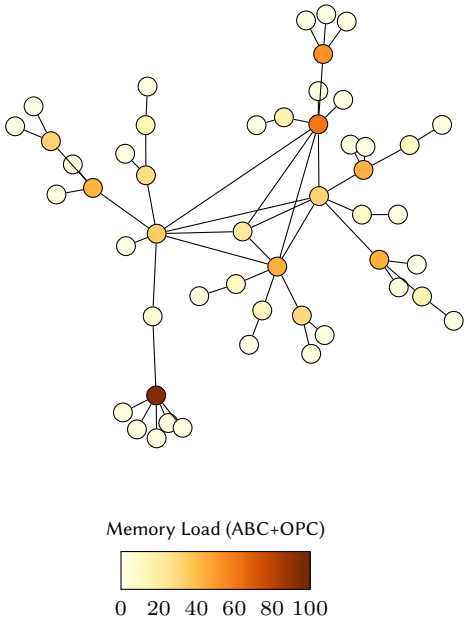
(a) ABC, Mesh Topology, Single Producer



(b) ABC+OPC, Mesh Topology, Single Producer

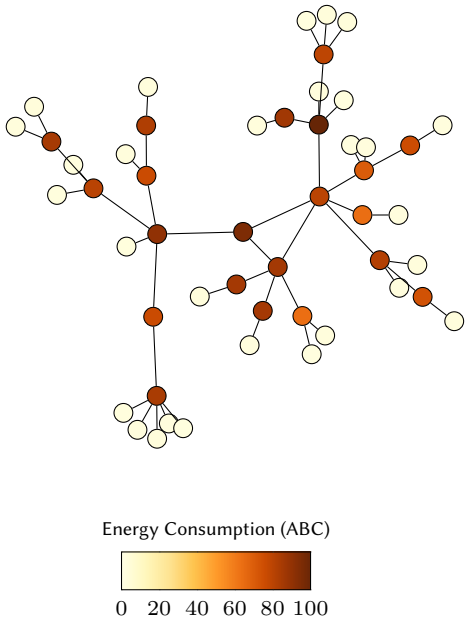


(c) ABC, Mesh Topology, Multi Producer

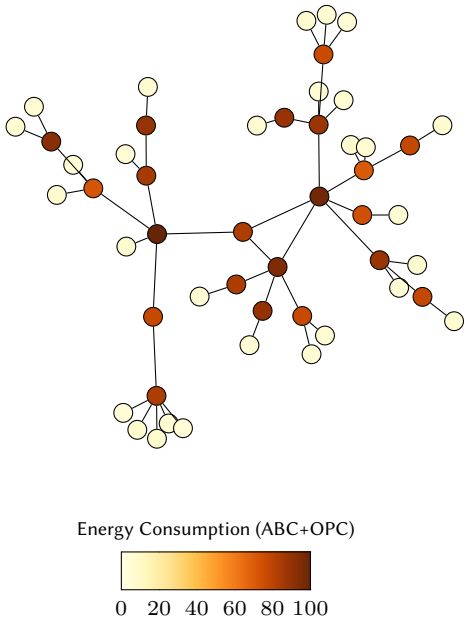


(d) ABC+OPC, Mesh Topology, Multi Producer

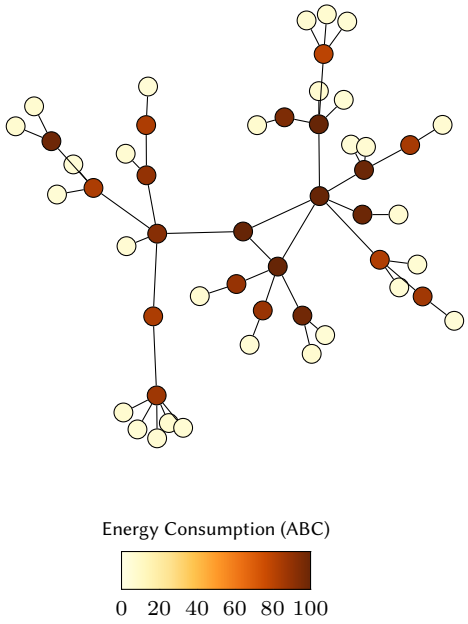
Figure A.22.: Memory load: ABC vs. ABC+OPC, Mesh topology



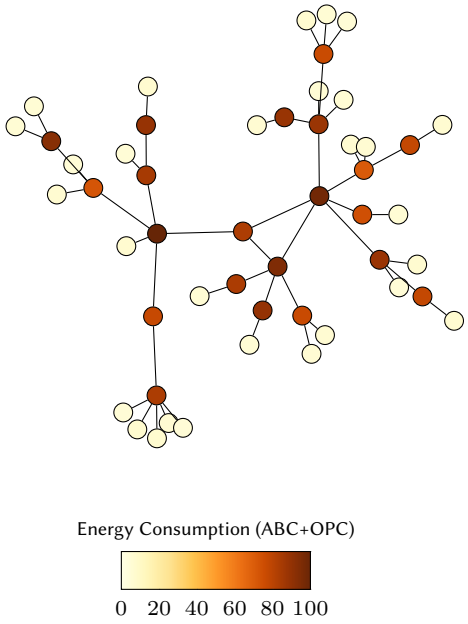
(a) ABC, Non-Mesh Topology, Single Producer



(b) ABC+OPC, Non-Mesh Topology, Single Producer

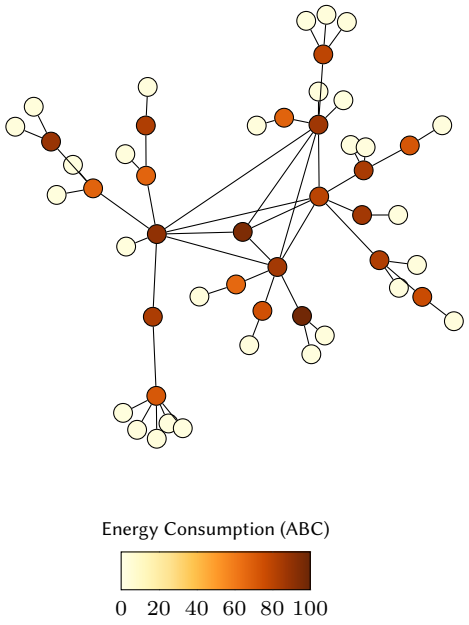


(c) ABC, Non-Mesh Topology, Multi Producer

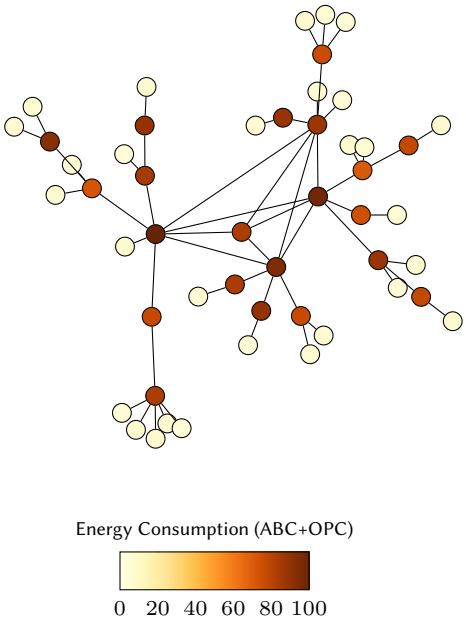


(d) ABC+OPC, Non-Mesh Topology, Multi Producer

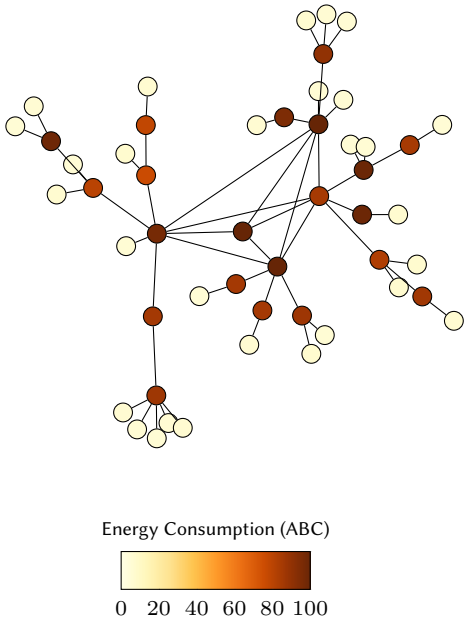
Figure A.23.: Energy consumption: ABC vs. ABC+OPC, Non-mesh topology



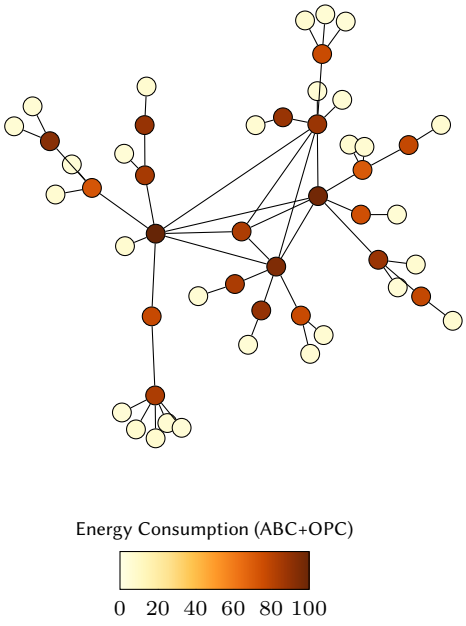
(a) ABC, Mesh Topology, Single Producer



(b) ABC+OPC, Mesh Topology, Single Producer



(c) ABC, Mesh Topology, Multi Producer



(d) ABC+OPC, Mesh Topology, Multi Producer

Figure A.24.: Energy consumption: ABC vs. ABC+OPC, Mesh topology

Bibliography

[1] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (Apr. 2016), 1–709.

[2] Adhatarao, S. S., Arumaithurai, M., Kutscher, D., and Fu, X. ISI: Integrate Sensor Networks to Internet With ICN. *IEEE Internet of Things Journal 5*, 2 (Apr. 2018), 491–499.

[3] Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J., and others. 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, 14-16 Dec 2015), pp. 459–464.

[4] Afanasyev, A., Mahadevan, P., Moiseenko, I., Uzun, E., and Zhang, L. Interest Flooding Attack and Countermeasures in Named Data Networking. In *IFIP Networking Conference* (2013), IEEE, pp. 1–9.

[5] Afanasyev, A., Shi, J., Wang, L., Zhang, B., and Zhang, L. Packet Fragmentation in NDN: Why NDN Uses Hop-by-Hop Fragmentation. *Tech. Rep.* (2015).

[6] Ahmed, S. H., Bouk, S. H., and Kim, D. RUFS: RobUst Forwarder Selection in Vehicular Content-Centric Networks. *IEEE Communications Letters 19*, 9 (Sept. 2015), 1616–1619.

[7] Ali, Z., Shah, M. A., Almogren, A., Ud Din, I., Maple, C., and Khattak, H. A. Named Data Networking for Efficient IoT-based Disaster Management in a Smart Campus. *Sustainability 12*, 8 (Jan. 2020), 3088. Publisher: Multidisciplinary Digital Publishing Institute.

[8] Amadeo, M., Campolo, C., Iera, A., and Molinaro, A. Named Data Networking for IoT: An Architectural Perspective. In *European Conference on Networks and Communications (EuCNC)* (2014), IEEE, pp. 1–5.

[9] Amadeo, M., Campolo, C., and Molinaro, A. Multi-Source Data Retrieval in IoT Via Named Data Networking. In *Proceedings of the 1st International Conference on Information-Centric Networking (ICN)* (Paris, France, 2014), 24-26 Sep, pp. 67–76.

[10] Amadeo, M., Campolo, C., Molinaro, A., and Mitton, N. Named Data Networking: A Natural Design for Data Collection in Wireless Sensor Networks. In *Wireless Days (WD), IFIP 2013* (2013), IEEE, pp. 1–6.

[11] Amadeo, M., Campolo, C., Molinaro, A., and Ruggeri, G. Content-Centric Wireless Networking: A Survey. *Computer Networks 72* (2014), 1–13.

[12] Amadeo, M., Campolo, C., Ruggeri, G., Lia, G., and Molinaro, A. Caching Transient Contents in Vehicular Named Data Networking: A Performance Analysis. *Sensors 20*, 7 (Jan. 2020), 1985. Publisher: Multidisciplinary Digital Publishing Institute.

[13] Arshad, S., Azam, M. A., Ahmed, S. H., and Loo, J. Towards Information-Centric Networking (ICN) Naming for Internet of Things (IoT): The Case of Smart Campus. In *Proceedings of the International Conference on Future Networks and Distributed Systems* (2017), ICFNDS '17, ACM, pp. 41:1–41:6. Cambridge, United Kingdom.

[14] Arshad, S., Azam, M. A., Rehmani, M. H., and Loo, J. Information-Centric Networking Based Caching and Naming Schemes for Internet of Things: A Survey and Future Research Directions. *arXiv preprint arXiv:1710.03473* (2017).

[15] Arshad, S., Azam, M. A., Rehmani, M. H., and Loo, J. Recent Advances in Information-Centric Networking-Based Internet of Things (ICN-IoT). *IEEE Internet of Things Journal 6*, 2 (Apr. 2019), 2128–2158.

[16] Arshad, S., Shahzaad, B., Azam, M. A., Loo, J., Ahmed, S. H., and Aslam, S. Hierarchical and Flat-Based Hybrid Naming Scheme in Content-Centric Networks of Things. *IEEE Internet of Things Journal 5*, 2 (Apr. 2018), 1070–1080.

[17] Asmat, H., Din, I. U., Ullah, F., Talha, M., Khan, M., and Guizani, M. ELC: Edge Linked Caching for Content Updating in Information-Centric Internet of Things. *Computer Communications 156* (Apr. 2020), 174–182.

[18] Atmel. AT86rf231 Low Power 2.4 GHz Transceiver for ZigBee, IEEE 802.15.4, 6LoWPAN, RF4CE, SP100, WirelessHART, and ISM Applications, 2009.

[19] Baccelli, E., Gündoğan, C., Hahm, O., Kietzmann, P., Lenders, M. S., Petersen, H., Schleiser, K., Schmidt, T. C., and Wählisch, M. 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.

[20] Baccelli, E., Hahm, O., Gunes, M., Wählisch, M., and Schmidt, T. C. RIOT OS: Towards an OS for the Internet of Things. In *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2013), IEEE, pp. 79–80.

[21] Baccelli, E., Mehlis, C., Hahm, O., Schmidt, T. C., and Wählisch, M. Information Centric Networking in the IoT: Experiments with NDN in the Wild. In *Proceedings of the 1st International Conference on Information-Centric Networking* (2014), ACM, pp. 77–86.

[22] Bavelas, A. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America 22*, 6 (1950), 725–730. Publisher: Acoustical Society of America.

[23] Bernardini, C., Silverston, T., and Festor, O. Socially-Aware Caching Strategy for Content Centric Networking. In *2014 IFIP Networking Conference* (June 2014), pp. 1–9.

[24] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W. An Architecture for Differentiated Services. RFC 2475, IETF, December 1998.

[25] Bloom, B. H. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Communications of the ACM 13*, 7 (July 1970), 422–426.

[26] Bluetooth, S. I. G. Bluetooth Core Specification Version 4.0. *Specification of the Bluetooth System 1* (2010), 7.

[27] Bonacich, P. Factoring and Weighting Approaches to Status Scores and Clique Identification. *Journal of Mathematical Sociology 2* (1972), 113–120. Publisher: Taylor & Francis Group.

[28] Borgatti, S. P. Centrality and Network Flow. *Social Networks 27*, 1 (Jan. 2005), 55–71.

[29] Bormann, C., Ersue, M., and Keranen, A. Terminology for Constrained-Node Networks. Tech. rep., Internet Engineering Task Force (IETF), 2014.

[30] Braden, R., Clark, D., and Shenker, S. Integrated Services in the Internet Architecture: an Overview. RFC 1633, IETF, June 1994.

[31] Braden, R., Zhang, L., Berson, S., Herzog, S., and Jamin, S. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, IETF, September 1997.

[32] Breslau, L., Cao, P., Fan, L., Phillips, G., and Shenker, S. Web Caching and Zipf-Like Distributions: Evidence and Implications. In *Proceedings of the IEEE INFOCOM* (New York, NY, USA, 21-25 Mar 1999), pp. 126–134.

[33] Burke, J., Gasti, P., Nathan, N., and Tsudik, G. Securing Instrumented Environments Over Content-Centric Networking: The Case of Lighting Control and NDN. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2013), IEEE, pp. 394–398.

[34] Carofiglio, G., Gallo, M., Muscariello, L., and Perino, D. Modeling Data Transfer in Content-Centric Networking. In *Proceedings of the 23rd International Teletraffic Congress* (San Francisco, CA, USA, 6-8 Sep 2011), pp. 111–118.

[35] Carofiglio, G., Gehlen, V., and Perino, D. Experimental Evaluation of Memory Management in Content-Centric Networking. In *Proceedings of the IEEE International Conference on Communications (ICC)* (Kyoto, Japan, 5-9 Jun 2011), pp. 1–6.

[36] Carofiglio, G., Mekinda, L., and Muscariello, L. FOCAL: Forwarding and Caching with Latency Awareness in Information-Centric Networking. In *2015 IEEE Globecom Workshops (GC Wkshps)* (Dec. 2015), pp. 1–7.

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

[38] Carofiglio, G., Mekinda, L., and Muscariello, L. Analysis of Latency-Aware Caching Strategies in Information-Centric Networking. In *Proceedings of the 1st Workshop on Content Caching and Delivery in Wireless Networks* (Heidelberg, Germany, 2016), CCDWN '16, ACM, pp. 5:1–5:7.

[39] Carofiglio, G., Mekinda, L., and Muscariello, L. Joint Forwarding and Caching with Latency Awareness in Information-Centric Networking. *Computer Networks 110* (Dec. 2016), 133–153.

[40] Carzaniga, A., Papalini, M., and Wolf, A. L. Content-Based Publish/Subscribe Networking and Information-Centric Networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking* (2011), ACM, pp. 56–61.

[41] Chai, W. K., He, D., Psaras, I., and Pavlou, G. Cache "Less for More" in Information-Centric Networks (Extended Version). *Computer Communications 36*, 7 (2013), 758–770.

[42] Chen, B., Liu, L., Zhang, Z., Yang, W., and Ma, H. BRR-CVR: A Collaborative Caching Strategy for Information-Centric Wireless Sensor Networks. In *12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)* (2016), IEEE, pp. 31–38.

[43] Chiocchetti, R., Rossi, D., Rossini, G., Carofiglio, G., and Perino, D. 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, 17 Aug 2012), pp. 7–12.

[44] Chun, B.-G., Chaudhuri, K., Wee, H., Barreno, M., Papadimitriou, C. H., and Kubiatowicz, J. Selfish Caching in Distributed Systems: A Game-Theoretic Analysis. In *Proceedings of the Twenty-Third Annual Acm Symposium on Principles of Distributed Computing* (St. John's, Newfoundland, Canada, July 2004), PODC '04, Association for Computing Machinery, pp. 21–30.

[45] Dan, G. Cache-to-Cache: Could ISPs Cooperate to Decrease Peer-to-Peer Content Distribution Costs? *IEEE Transactions on Parallel and Distributed Systems 22*, 9 (Sept. 2011), 1469–1482.

[46] Dannewitz, C., Kutscher, D., Ohlman, B., Farrell, S., Ahlgren, B., and Karl, H. Network of Information (NetInf) – an Information-Centric Networking Architecture. *Computer Communications 36*, 7 (Apr. 2013), 721–735.

[47] Detti, A., Blefari Melazzi, N., Salsano, S., and Pomposini, M. CONET: A Content Centric Inter-Networking Architecture. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking* (New York, NY, USA, Aug. 2011), ICN '11, Association for Computing Machinery, pp. 50–55.

[48] Din, I. U., Hassan, S., Khan, M. K., Guizani, M., Ghazali, O., and Habbal, A. Caching in Information-Centric Networking: Strategies, Challenges, and Future Research Directions. *IEEE Communications Surveys Tutorials 20*, 2 (2018), 1443–1474.

[49] Djama, A., Djamaa, B., and Senouci, M. R. Information-Centric Networking Solutions for the Internet of Things: A Systematic Mapping Review. *Computer Communications* (May 2020), 37–59.

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

[51] Dräxler, M., and Karl, H. Efficiency of On-Path and Off-Path Caching Strategies in Information Centric Networks. In *IEEE International Conference on Green Computing and Communications (GreenCom)* (2012), IEEE, pp. 581–587.

[52] Elayoubi, S.-E., and Roberts, J. Performance and Cost Effectiveness of Caching in Mobile Access Networks. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking* (San Francisco, California, USA, Sept. 2015), ACM-ICN '15, Association for Computing Machinery, pp. 79–88.

[53] Everett, M., and Borgatti, S. P. Ego Network Betweenness. *Social Networks 27*, 1 (2005), 31–38.

[54] Fang, C., Yu, F. R., Huang, T., Liu, J., and Liu, Y. A Survey of Energy-Efficient Caching in Information-Centric Networking. *IEEE Communications Magazine 52*, 11 (2014), 122–129.

[55] Fayazbakhsh, S. K., Lin, Y., Tootoonchian, A., Ghodsi, A., Koponen, T., Maggs, B., Ng, K. C., Sekar, V., and Shenker, S. Less Pain, Most of the Gain: Incrementally Deployable ICN. In *ACM SIGCOMM Computer Communication Review* (2013), vol. 43, ACM, pp. 147–158.

[56] Fotiou, N., Nikander, P., Trossen, D., and Polyzos, G. C. Developing Information Networking Further: From PSIRP to PURSUIT. In *Broadband Communications, Networks, and Systems* (Berlin, Heidelberg, 2012), I. Tomkos, C. J. Bouras, G. Ellinas, P. Demestichas, and P. Sinha, Eds., Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer, pp. 1–13.

[57] Fotiou, N., and Polyzos, G. C. Realizing the Internet of Things Using Information-Centric Networking. In *10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine)* (2014), IEEE, pp. 193–194.

[58] García, G., Beben, A., Ramón, F. J., Maeso, A., Psaras, I., Pavlou, G., Wang, N., Śliwiński, J., Spirou, S., Soursos, S., and Hadjioannou, E. COMET: Content Mediator Architecture for Content-Aware Networks. In *2011 Future Network Mobile Summit* (June 2011), pp. 1–8.

[59] Guan, J., Yan, Z., Yao, S., Xu, C., and Zhang, H. GBC-Based Caching Function Group Selection Algorithm for Sinet. *Journal of Network and Computer Applications 85* (May 2017), 56–63.

[60] Gündoğan, C., Kietzmann, P., Lenders, M., Petersen, H., Schmidt, T. C., and Wählisch, M. NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT. In *Proceedings of the 5th ACM Conference on Information-Centric Networking* (2018), ICN '18, ACM, pp. 159–171. Boston, Massachusetts.

[61] Gündoğan, C., Kietzmann, P., Schmidt, T. C., and Wählisch, M. HoPP: Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things. In *Proc. of the 43rd IEEE Conference on Local Computer Networks (LCN)* (Piscataway, NJ, USA, Oct. 2018), IEEE Press, pp. 331–334.

[62] Gündoğan, C., Pfender, J., Frey, M., Schmidt, T. C., Shzu-Juraschek, F., and Wählisch, M. Gain More for Less: The Surprising Benefits of QoS Management in Constrained NDN Networks. In *Proceedings of the 6th ACM Conference on Information-Centric Networking* (Macao, China, Sept. 2019), ICN '19, Association for Computing Machinery, pp. 141–152.

[63] Gündoğan, C., Pfender, J., Kietzmann, P., Schmidt, T. C., and Wählisch, M. On the Impact of QoS Management in an Information-Centric Internet of Things. *Computer Communications 154* (Mar. 2020), 160–172.

[64] Gündoğan, C., Schmidt, T. C., Wählisch, M., Frey, M., Shzu-Juraschek, F., and Pfender, J. Quality of Service for ICN in the IoT. IRTF Internet Draft – work in progress 01, IRTF, July 2019.

[65] Gupta, D., Rani, S., Ahmed, S. H., and Hussain, R. Caching Policies in NDN-IoT Architecture. In *Integration of WSN and IoT for Smart Cities*, EAI/Springer Innovations in Communication and Computing. Springer International Publishing, 2020, pp. 43–64.

[66] Hahm, O., Baccelli, E., Schmidt, T., Wählisch, M., Adjih, C., and Massoulié, L. Low-Power Internet of Things with NDN & Cooperative Caching. In *Proceedings of the 4th ACM Conference on Information-Centric Networking (ICN)* (Berlin, Germany, 26-28 Sep 2017), pp. 98–108.

[67] Hahm, O., Baccelli, E., Schmidt, T. C., Wahlisch, M., and Adjih, C. A Named Data Network Approach to Energy Efficiency in IoT. In *Proceedings of the IEEE Globecom Workshops (GC Wkshps)* (Washington, DC, USA, 4-8 Dec 2016), pp. 1–6.

[68] Hail, M. A., Amadeo, M., Molinaro, A., and Fischer, S. Caching in Named Data Networking for the Wireless Internet of Things. In *International Conference on Recent Advances in Internet of Things (RIoT)* (2015), IEEE, pp. 1–6.

[69] Hail, M. A. M., Amadeo, M., Molinaro, A., and Fischer, S. 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)* (Malaga, Spain, 25-27 May 2015), Springer, pp. 313–326.

[70] Hoque, A. K. M., Amin, S. O., Alyyan, A., Zhang, B., Zhang, L., and Wang, L. NLSR: Named-Data Link State Routing Protocol. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-Centric Networking* (2013), ACM, pp. 15–20.

[71] Hua, Y., Guan, L., and Kyriakopoulos, K. G. A Fog Caching Scheme Enabled by ICN for IoT Environments. *Future Generation Computer Systems 111* (Oct. 2020), 82–95.

[72] Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. Directed Diffusion for Wireless Sensor Networking. *IEEE/ACM Transactions on Networking (ToN) 11*, 1 (2003), 2–16.

[73] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies* (2009), ACM, pp. 1–12.

[74] Jangam, A., suthar, P., and Stolic, M. QoS Treatments in ICN using Disaggregated Name Components. Internet-Draft – work in progress 02, IETF, March 2020.

[75] Khan, J. A., Westphal, C., Garcia-Luna-Aceves, J. J., and Ghamri-Doudane, Y. NICE: Network-Oriented Information-Centric Centrality for Efficiency in Cache Management. In *Proceedings of the 5th ACM Conference on Information-Centric Networking* (Boston, Massachusetts, Sept. 2018), ICN '18, Association for Computing Machinery, pp. 31–42.

[76] Khodaparas, S., Benslimane, A., and Yousefi, S. A Software-Defined Caching Scheme for the Internet of Things. *Computer Communications 158* (May 2020), 178–188.

[77] Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. A Data-Oriented (and Beyond) Network Architecture. In *ACM SIGCOMM Computer Communication Review* (2007), vol. 37, ACM, pp. 181–192.

[78] Król, M., Habak, K., Oran, D., Kutscher, D., and Psaras, I. RICE: Remote Method Invocation in ICN. In *Proceedings of the 5th ACM Conference on Information-Centric Networking* (Boston, Massachusetts, Sept. 2018), ICN '18, Association for Computing Machinery, pp. 1–11.

[79] Król, M., Mastorakis, S., Oran, D., and Kutscher, D. Compute First Networking: Distributed Computing meets ICN. In *Proceedings of the 6th ACM Conference on Information-Centric Networking* (Macao, China, Sept. 2019), ICN '19, Association for Computing Machinery, pp. 67–77.

[80] Król, M., and Psaras, I. NFaaS: Named Function as a Service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (Berlin, Germany, Sept. 2017), ICN '17, Association for Computing Machinery, pp. 134–144.

[81] Kurniawan, F. S., Yovita, L. V., and Wibowo, T. A. Modified-LRU Algorithm for Caching on Named Data Network. In *2019 International Conference on Electrical Engineering and Informatics (ICEEI)* (July 2019), pp. 438–443. ISSN: 2155-6830.

[82] Kushalnagar, N., and Montenegro, G. Transmission of IPv6 Packets over IEEE 802.15.4 Networks, 2007.

[83] Lai, J., Li, C., Chen, Z., Sun, Y., and Xiao, H. A Novel Content Popularity and Distance based Interval Caching Strategy for Named Data Mobile Ad-hoc Network (NDMANET). In *2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET)* (Aug. 2019), pp. 259–263.

[84] Laoutaris, N., Che, H., and Stavrakakis, I. The LCD Interconnection of LRU Caches and its Analysis. *Performance Evaluation 63*, 7 (July 2006), 609–634.

[85] Laoutaris, N., Telelis, O., Zissimopoulos, V., and Stavrakakis, I. Distributed Selfish Replication. *IEEE Transactions on Parallel and Distributed Systems 17*, 12 (Dec. 2006), 1401–1413.

[86] Lee, M., Song, J., Cho, K., Pack, S., Kwon, T. "., Kangasharju, J., and Choi, Y. Content Discovery for Information-Centric Networking. *Computer Networks 83* (June 2015), 1–14.

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

[88] Li, Z., Point, J.-C., Ciftci, S., Eker, O., Mauri, G., Savi, M., and Verticale, G. ICN Based Shared Caching in Future Converged Fixed and Mobile Network. In *16th International Conference on High Performance Switching and Routing (HPSR)* (2015), IEEE, pp. 1–6.

[89] Li, Z., and Simon, G. 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, 5-9 Jun 2011), pp. 1–6.

[90] Lindgren, A., Abdesslem, F. B., Ahlgren, B., Schelén, O., and Malik, A. M. Design Choices for the IoT in Information-Centric Networks. In *Proceedings of the 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (Las Vegas, NV, USA, 9-12 Jan 2016), pp. 882–888.

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

[92] Liu, Y., Zhu, D., and Ma, W. A Novel Cooperative Caching Scheme for Content Centric Mobile Ad Hoc Networks. In *IEEE Symposium on Computers and Communication (ISCC)* (2016), IEEE, pp. 824–829.

[93] MacGillivray, C., and Reinsel, D. Worldwide Global DataSphere IoT Device and Data Forecast, 2019–2023, May 2019.

[94] Meddeb, M., Dhraief, A., Belghith, A., Monteil, T., and Drira, K. How to Cache in ICN-Based IoT Environments? In *IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)* (2017), IEEE, pp. 1117–1124.

[95] Meddeb, M., Dhraief, A., Belghith, A., Monteil, T., Drira, K., and AlAhmadi, S. Cache Freshness in Named Data Networking for the Internet of Things. *The Computer Journal 61*, 10 (Oct. 2018), 1496–1511. Publisher: Oxford Academic.

[96] Meddeb, M., Dhraief, A., Belghith, A., Monteil, T., Drira, K., and Mathkour, H. Least Fresh First Cache Replacement Policy for NDN-Based IoT Networks. *Pervasive and Mobile Computing 52* (Jan. 2019), 60–70.

[97] Melvix, L., Lokesh, V., and Polyzos, G. C. Energy Efficient Context Based Forwarding Strategy in Named Data Networking of Things. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking* (2016), ACM, pp. 249–254.

[98] Moiseenko, I., and Oran, D. Flow Classification in Information Centric Networking. Internet-Draft – work in progress 05, IETF, January 2020.

[99] Moiseenko, I., and Zhang, L. Consumer-Producer API for Named Data Networking. In *Proceedings of the 1st International Conference on Information-Centric Networking* (2014), ACM, pp. 177–178.

[100] Naeem, M. A., Ali, R., Kim, B.-S., Nor, S. A., and Hassan, S. A Periodic Caching Strategy Solution for the Smart City in Information-Centric Internet of Things. *Sustainability 10*, 7 (July 2018), 2576.

[101] Naeem, M. A., Rehman, M. A. U., Ullah, R., and Kim, B.-S. A Comparative Performance Analysis of Popularity-Based Caching Strategies in Named Data Networking. *IEEE Access 8* (2020), 50057–50077.

[102] Nash, J. Two-Person Cooperative Games. *Econometrica 21*, 1 (1953), 128–140. Publisher: [Wiley, Econometric Society].

[103] Naz, S., Naveed Bin Rais, R., Shah, P. A., Yasmin, S., Qayyum, A., Rho, S., and Nam, Y. A Dynamic Caching Strategy for CCN-Based MANETs. *Computer Networks 142* (Sept. 2018), 93–107.

[104] Nelson, B. J. *Remote Procedure Call.* PhD thesis, Carnegie Mellon University, USA, 1981. AAI8204168.

[105] Nichols, K., Blake, S., Baker, F., and Black, D. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, IETF, December 1998.

[106] Nour, B., Sharif, K., Li, F., Biswas, S., Moungla, H., Guizani, M., and Wang, Y. A Survey of Internet of Things Communication Using ICN: A Use Case Perspective. *Computer Communications 142-143* (June 2019), 95–123.

[107] Nour, B., Sharif, K., Li, F., Moungla, H., Kamal, A. E., and Afifi, H. NCP: A near ICN Cache Placement Scheme for IoT-Based Traffic Class. In *2018 IEEE Global Communications Conference (GLOBECOM)* (Dec. 2018), pp. 1–6. ISSN: 1930-529X.

[108] Oran, D. Considerations in the Development of a QoS Architecture for CCNx-like ICN protocols. Internet-Draft – work in progress 04, IETF, February 2020.

[109] Oran, D. Maintaining CCNx or NDN Flow Balance with Highly Variable Data Object Sizes. Internet-Draft – work in progress 03, IETF, February 2020.

[110] Pacifici, V., and Dán, G. Selfish Content Replication on Graphs. In *2011 23rd International Teletraffic Congress (ITC)* (Sept. 2011), pp. 119–126.

[111] Pacifici, V., and Dán, G. Content-Peering Dynamics of Autonomous Caches in a Content-Centric Network. In *2013 Proceedings IEEE INFOCOM* (Apr. 2013), pp. 1079–1087. ISSN: 0743-166X.

[112] Pfender, J., Valera, A., and Seah, W. K. G. Performance Comparison of Caching Strategies for Information-Centric IoT. In *5th ACM Conference on Information-Centric Networking (ICN '18)* (Boston, MA, USA, 21-23 September 2018), ACM, pp. 43–53.

[113] Pfender, J., Valera, A., and Seah, W. K. G. Content Delivery Latency of Caching Strategies for Information-Centric IoT. *arXiv:1905.01011 [cs]* (May 2019).

[114] Pfender, J., Valera, A., and Seah, W. K. G. Easy as ABC: A Lightweight Centrality-Based Caching Strategy for Information-Centric IoT. In *Proceedings of the 6th ACM Conference on Information-Centric Networking* (Macao, China, Sept. 2019), ICN '19, Association for Computing Machinery, pp. 100–111.

[115] Priscilla, C. V., and Charulatha, A. A Comparative Study on Caching Strategies in Content Centric Networking for Mobile Networks. In *2019 11th International Conference on Advanced Computing (ICoAC)* (Dec. 2019), pp. 122–128.

[116] Psaras, I., Chai, W. K., and Pavlou, G. Probabilistic In-Network Caching for Information-Centric Networks. In *Proceedings of the Second Edition of the ICN Workshop on Information-Centric Networking* (Helsinki, Finland, 17 Aug 2012), pp. 55–60.

[117] Psaras, I., Clegg, R. G., Landa, R., Chai, W. K., and Pavlou, G. Modelling and Evaluation of CCN-Caching Trees. In *Proceedings of the 10th International IFIP TC 6 Conference on Networking* (Valencia, Spain, 9-13 May 2011), NETWORKING, pp. 78–91.

[118] Putra, M. A. P., Situmorang, H., and Syambas, N. R. Least Recently Frequently Used Replacement Policy Named Data Networking Approach. In *2019 International Conference on Electrical Engineering and Informatics (ICEEI)* (July 2019), pp. 423–427. ISSN: 2155-6830.

[119] Qin, J., Xue, K., Chen, Y., Wei, W., Liu, J., and Yue, H. STNDN: Link Aware Segmented Transmission for Named Data Networking. In *2019 2nd International Conference on Hot Information-Centric Networking (HotICN)* (Dec. 2019), pp. 50–55.

[120] Quevedo, J., Corujo, D., and Aguiar, R. A Case for ICN Usage in IoT Environments. In *IEEE Global Communications Conference (GLOBECOM)* (2014), IEEE, pp. 2770–2775.

[121] Quevedo, J., Corujo, D., and Aguiar, R. Consumer Driven Information Freshness Approach for Content Centric Networking. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2014), IEEE, pp. 482–487.

[122] Rao, A., Schelén, O., and Lindgren, A. Performance Implications for IoT Over Information Centric Networks. In *Proceedings of the Eleventh ACM Workshop on Challenged Networks* (2016), ACM, pp. 57–62.

[123] Rosensweig, E. J., Kurose, J., and Towsley, D. Approximate Models for General Cache Networks. In *Proceedings of the IEEE INFOCOM* (San Diego, CA, USA, 14-19 Mar 2010), pp. 1–9.

[124] Rossi, D., and Rossini, G. On Sizing CCN Content Stores by Exploiting Topological Information. In *2012 Proceedings IEEE INFOCOM Workshops* (Mar. 2012), pp. 280–285.

[125] Sabidussi, G. The Centrality Index of a Graph. *Psychometrika 31*, 4 (Dec. 1966), 581–603.

[126] Saha, S., Lukyanenko, A., and Ylä-Jääski, A. Cooperative Caching Through Routing Control in Information-Centric Networks. In *Proceedings of the IEEE INFOCOM* (Turin, Italy, 14-19 Apr 2013), pp. 100–104.

[127] Sarvotham, S., Riedi, R., and Baraniuk, R. Connection-Level Analysis and Modeling of Network Traffic. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), IMW '01, ACM, pp. 99–103. San Francisco, California, USA.

[128] Seah, W. K. G., Eu, Z. A., and Tan, H.-P. Wireless Sensor Networks Powered by Ambient Energy Harvesting (WSN-HEAP) - Survey and Challenges. In *1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE)* (2009), IEEE, pp. 1–5.

[129] Seskar, I., Nagaraja, K., Nelson, S., and Raychaudhuri, D. MobilityFirst Future Internet Architecture Project. In *Proceedings of the 7th Asian Internet Engineering Conference* (New York, NY, USA, Nov. 2011), AINTEC '11, Association for Computing Machinery, pp. 1–3.

[130] Shang, W., Ding, Q., Marianantoni, A., Burke, J., and Zhang, L. Securing Building Management Systems Using Named Data Networking. *IEEE Network 28*, 3 (2014), 50–56.

[131] Shang, W., Thompson, J., Cherkaoui, M., Burkey, J., and Zhang, L. NDN. JS: A JavaScript Client Library for Named Data Networking. In *Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2013), IEEE, pp. 399–404.

[132] Shekhawat, V. S., Vineet, A., and Gautam, A. Efficient Content Caching for Named Data Network Nodes. In *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services* (Houston, Texas, Nov. 2019), MobiQuitous '19, Association for Computing Machinery, pp. 11–19.

[133] Shi, J., and Zhang, B. NDNLP: A Link Protocol for NDN. *The University of Arizona, Tucson, AZ, NDN Technical Report NDN-0006* (2012).

[134] Song, Y., Ma, H., and Liu, L. TCCN: Tag-Assisted Content Centric Networking for Internet of Things. In *16th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (2015), IEEE, pp. 1–9.

[135] Sun, Y., Zhang, T., Wang, R., Feng, W., and Chen, P. Topology Potential Based Probability Caching Strategy for Content Centric Ad Hoc Networks. *Journal of Residuals Science & Technology 13*, 6 (2016).

[136] Tarkoma, S., Ain, M., and Visala, K. The Publish/Subscribe Internet Routing Paradigm (PSIRP): Designing the Future Internet Architecture. *Towards the Future Internet: A European Research Perspective* (2009), 102–111. Publisher: IOS PRESS.

[137] Tarnoi, S., Suksomboon, K., Kumwilaisak, W., and Ji, Y. 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, 8-11 Oct 2014), pp. 99–106.

[138] Tschudin, C., and Sifalakis, M. Named Functions and Cached Computations. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th* (2014), IEEE, pp. 851–857.

[139] Tsilopoulos, C., and Xylomenos, G. Supporting Diverse Traffic Types in Information Centric Networks. In *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking* (Toronto, Ontario, Canada, 2011), ICN '11, ACM, pp. 13–18.

[140] Vasilakos, X., Siris, V. A., Polyzos, G. C., and Pomonis, M. 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, 17 Aug 2012), pp. 61–66.

[141] Vural, S., Navaratnam, P., Wang, N., Wang, C., Dong, L., and Tafazolli, R. In-Network Caching of Internet-of-Things Data. In *International Conference on Communications (ICC)* (2014), IEEE, pp. 3185–3190.

[142] Vural, S., Wang, N., Navaratnam, P., and Tafazolli, R. Caching Transient Data in Internet Content Routers. *IEEE/ACM Transactions on Networking 25*, 2 (2017), 1048–1061.

[143] Wählisch, M., Schmidt, T. C., and Vahlenkamp, M. Backscatter from the Data Plane - Threats to Stability and Security in Information-Centric Network Infrastructure. *Computer Networks 57*, 16 (2013), 3192–3206.

[144] Wallerich, J., Dreger, H., Feldmann, A., Krishnamurthy, B., and Willinger, W. A Methodology for Studying Persistency Aspects of Internet Flows. *SIGCOMM Comput. Commun. Rev. 35*, 2 (Apr. 2005), 23–36.

[145] Waltari, O., and Kangasharju, J. Content-Centric Networking in the Internet of Things. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)* (Jan. 2016), pp. 73–78. ISSN: 2331-9860.

[146] Wang, H., Hernandez, J. M., and Van Mieghem, P. Betweenness Centrality in a Weighted Network. *Physical Review E 77*, 4 (Apr. 2008), 046105. Publisher: American Physical Society.

[147] Wang, L., Bayhan, S., and Kangasharju, J. Cooperation Policies for Efficient in-Network Caching. *ACM SIGCOMM Computer Communication Review 43*, 4 (Aug. 2013), 533–534.

[148] Wang, L., Bayhan, S., and Kangasharju, J. Effects of Cooperation Policy and Network Topology on Performance of In-Network Caching. *IEEE Communications Letters 18*, 4 (Apr. 2014), 680–683.

[149] Wang, L., Hoque, A., Yi, C., Alyyan, A., and Zhang, B. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. *University of Memphis and University of Arizona, Tech. Rep* (2012).

[150] Wang, L., Kangasharju, J., and Crowcroft, J. You Really Need A Good Ruler to Measure Caching Performance in Information-Centric Networks. *arXiv:1603.05963 [cs]* (Aug. 2016).

[151] Wang, L., Tyson, G., Kangasharju, J., and Crowcroft, J. FairCache: Introducing Fairness to ICN Caching. In *24th International Conference on Network Protocols (ICNP), 2016* (2016), IEEE, pp. 1–10.

[152] Wang, L., Tyson, G., Kangasharju, J., and Crowcroft, J. Milking the Cache Cow With Fairness in Mind. *IEEE/ACM Transactions on Networking 25*, 5 (Oct. 2017), 2686–2700.

[153] Wang, W., Sun, Y., Guo, Y., Kaafar, D., Jin, J., Li, J., and Li, Z. CRCache: Exploiting the Correlation Between Content Popularity and Network Topology Information for ICN Caching. In *2014 IEEE International Conference on Communications (ICC)* (June 2014), pp. 3191–3196. ISSN: 1938-1883.

[154] Wasserman, S., and Faust, K. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Nov. 1994. Google-Books-ID: CAm2DpIqRUIC.

[155] Welzl, M. *Network Congestion Control: Managing Internet Traffic*. John Wiley & Sons, Dec. 2005. Google-Books-ID: vfXmqrrjGD0C.

[156] Xu, X., Feng, C., Shan, S., Zhang, T., and Loo, J. Proactive Edge Caching in Content-Centric Networks With Massive Dynamic Content Requests. *IEEE Access 8* (2020), 59906–59921.

[157] Xylomenos, G., Ververidis, C. N., Siris, V. A., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K. V., and Polyzos, G. C. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials 16*, 2 (2014), 1024–1049.

[158] Yan, Z., Zeadally, S., and Park, Y. A Novel Vehicular Information Network Architecture Based on Named Data Networking (NDN). *IEEE Internet of Things Journal 1*, 6 (Dec. 2014), 525–532.

[159] Yi, C., Afanasyev, A., Moiseenko, I., Wang, L., Zhang, B., and Zhang, L. A Case for Stateful Forwarding Plane. *Computer Communications 36*, 7 (2013), 779–791.

[160] Yi, C., Afanasyev, A., Wang, L., Zhang, B., and Zhang, L. Adaptive Forwarding in Named Data Networking. *ACM SIGCOMM Computer Communication Review 42*, 3 (2012), 62–67.

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

[162] Zhang, G., Li, Y., and Lin, T. Caching in Information Centric Networking: A Survey. *Computer Networks 57*, 16 (2013), 3128–3141.

[163] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., Crowley, P., Papadopoulos, C., Wang, L., Zhang, B., and others. Named Data Networking. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 66–73.

[164] Zhang, L., Zhao, J., and Shi, Z. LF: A Caching Strategy for Named Data Mobile Ad Hoc Networks. In *Proceedings of the 4th International Conference on Computer Engineering and Networks* (2015), Springer, pp. 279–290.

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

[166] Zhang, T., Xu, X., Jiang, a. X., and Loo, J. Cache Space Efficient Caching Scheme for Content-Centric Mobile Ad Hoc Networks. *IEEE Systems Journal 13*, 1 (Mar. 2019), 530–541.

[167] Zhang, Y., Breslau, L., Paxson, V., and Shenker, S. On the Characteristics and Origins of Internet Flow Rates. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2002), SIGCOMM '02, ACM, pp. 309–322. Pittsburgh, Pennsylvania, USA.

[168] Zhang, Y., Raychadhuri, D., Grieco, L., Baccelli, E., Burke, J., Ravindran, R., Wang, G., Ahlgren, B., and Schelen, O. Requirements and Challenges for IoT over ICN. Internet-draft – work in progress, IETF, April 2016.

[169] Zhang, Y., Raychadhuri, D., Ravindran, R., and Wang, G. ICN Based Architecture for IoT. *IRTF contribution, October* (2013).

[170] Zhang, Z., Lung, C.-H., Lambadaris, I., and St-Hilaire, M. IoT Data Lifetime-Based Cooperative Caching Scheme for ICN-IoT Networks. In *2018 IEEE International Conference on Communications (ICC)* (May 2018), pp. 1–7. ISSN: 1938-1883.

[171] Zhao, C., Liu, J., Sheng, M., and Dai, Y. Efficient Betweenness Based Content Caching and Delivery Strategy in Wireless Networks. *arXiv:2005.03814 [cs, math]* (May 2020).

[172] Zhou, L., Zhang, T., Xu, X., Zeng, Z., and Liu, Y. Broadcasting Based Neighborhood Cooperative Caching for Content Centric Ad Hoc Networks. In *IEEE/CIC International Conference on Communications in China (ICCC)* (2015), IEEE, pp. 1–5.

[173] Zhu, H., Cao, Y., Wei, X., Wang, W., Jiang, T., and Jin, S. Caching Transient Data for Internet of Things: A Deep Reinforcement Learning Approach. *IEEE Internet of Things Journal 6*, 2 (Apr. 2019), 2074–2083.

[174] Zhu, Z., Bian, C., Afanasyev, A., Jacobson, V., and Zhang, L. Chronos: Serverless Multi-User Chat Over NDN. *NDN, Technical Report NDN-0008* (2012).

[175] Ziegler, S., Crettaz, C., Ladid, L., Krco, S., Pokric, B., Skarmeta, A. F., Jara, A., Kastner, W., and Jung, M. IoT6–Moving to an IPv6-Based Future IoT. In *The Future Internet Assembly* (2013), Springer, pp. 161–172.

[176] Ziegler, S., Kirstein, P., Ladid, L., Skarmeta, A., and Jara, A. *The Case for IPv6 as an Enabler of the Internet of Things.* IEEE Internet of Things, 2015.

[177] Ziegler, S., Kirstein, P., Ladid, L., Skarmeta, A., and Jara, A. Understanding IPv6's Potential for IoT: The IoT6 Research Project. *Newsletter* (2015).