Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Efficient Data Replication in Distributed IoT Environments

Christoph Putz

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Efficient Data Replication in Distributed IoT Environments

Effiziente Datenreplikation in verteilten IoT Umgebungen

Author:	Christoph Putz
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Stefan Liebald, M. Sc.
	Dr. Marc-Oliver Pahl
Date:	April 15, 2019

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, April 15, 2019

Location, Date

Signature

Abstract

The Internet of Things (IoT) is a research area that got a lot of attention during the last years. The great interest comes from people's need for more and more omnipresent connectivity on the one side and from the easy availability of emerging new technologies on the other side. The Internet of Things nowadays is capable of connecting nearly anything everywhere. It can be expected that in a few years the majority of households deploy IoT solutions. For proper acceptance an IoT system needs to be fast, reliable and flexible to adapt to varying deployment scenarios. The most promising way for improving the performance and reliability of such systems is proper implementation of replication. Unfortunately, most of the current IoT-based proposals either concentrate on performance or reliability but not both. In this work existing replication strategies and mechanisms are analysed, compared and evaluated to identify the key requirements for an efficient replication strategy that satisfies the performance goal as well as the reliability goal. Based on the advantages and shortcoming of existing approaches a new replication strategy is proposed that has the potential of achieving the desired goals. The maximum benefits can presumably be achieved by intertwining algorithms for performance enhancement with mechanisms for reliability increase quite deeply instead of complementing one kind with pluggable representatives of the other kind. Furthermore, the resulting replication strategy should be configurable regarding different points to allow for adoption in versatile IoT deployment domains.

Zusammenfassung

Internet of Things (IoT) beschreibt ein Forschungsgebiet, dem in den letzten Jahren sehr viel Aufmerksamkeit zu Teil geworden ist. Das große Interesse kommt einerseits von dem Wunsch der Menschen nach immer mehr und mehr allgegenwärtiger Vernetztheit und zum anderen durch die steigende Verfügbarkeit von neuen und fortschrittlichen Technologien. Durch das Internet of Things können heutzutage bereits nahezu alle Dinge überall vernetzt werden und es ist davon auszugehen, dass bereits in wenigen Jahren der Großteil aller Haushalte mit IoT-Systemen ausgestattet sein wird. Um angemessen akzeptiert zu werden muss ein IoT-System schnell, zuverlässig und flexibel sein, um verschiedensten Einsatzszenarien zu trotzen. Der vielversprechendste Weg die Leistung und Zuverlässigkeit eines solchen Systems zu verbessern scheint der passende Einsatz von Replikation zu sein. Leider fokussieren sich die meisten aktuellen IoT-Ansätze entweder auf die Leistung oder auf die Zuverlässigkeit, aber nicht auf beides. In dieser Arbeit werden verschiedene existierende Replikationsstrategien und -mechanismen analysiert, verglichen und bewertet um die Hauptanforderungen an einen Replikationsmechanismus zu identifizieren, der sowohl die Leistungssteigerung, als auch die Verbesserung der Zuverlässigkeit erfüllt. Basierend auf den Stärken und Schwächen der analysierten Ansätze soll eine neue Replikationsstrategie entwickelt werden, die die gewünschten Ziele erreichen kann. Es ist zu vermuten, dass das Optimum dadurch zu erreichen ist, dass existierende Algorithmen zur Verbesserung der Leistung mit denen zur Verbesserung der Zuverlässigkeit von Grund auf miteinander vereint werden, anstatt Vertreter des einen mit vorgefertigten Versionen des anderen zu ergänzen. Des Weiteren sollte die resultierende Strategie in mehreren Punkten konfigurierbar sein, um für den Einsatz in vielseitigen IoT-Szenarien geeignet zu sein.

Contents

List of Tables

Chapter 1

Introduction

1.1 Motivation

The dream of automated homes, offices and other environments becomes reality more and more due to the rise of the Internet of Things (IoT). The rapid development branches out in different directions which gives IoT research topics a high popularity but on the other side the research at many fronts leads to a lack of standards and a lack of uniform and compatible approaches. In general, the performance and reliability of a system can be improved with replication. This leads to the question how replication can be utilized to maximize the benefits of IoT solutions. Ultimately an approach is needed to find a common solution for replication in distributed IoT environments that takes to most promising key technologies of modern IoT scenarios into consideration.

1.2 Goals

This thesis aims to provide a conceptual approach for an efficient data replication mechanism tailored for distributed IoT environments. The major benefits are expected in the form of improved performance of such distributed IoT systems and the parallel enhancement of reliability or at least the maintenance of existing system performance accompanied by improved resilience in case of node failure.

To achieve the previously mentioned goal several questions and problems have to be addressed. Different kinds of replication strategies have to be observed in order to determine their applicability in distributed IoT scenarios. These strategies need to include what data should be replicated and where it should be stored inside the distributed system. Furthermore, the mechanisms for the dissemination of data need to

work in a distributed manner. Centralized approaches are probably not suitable due to the distributed and volatile nature of IoT environments. After the identification of promising replication strategies concrete mechanisms and algorithms need to be found. These algorithms again need to be examined for their suitability inside distributed IoT environments.

In the end this thesis should introduce a possibility to improve existing IoT frameworks. In this context existing requirements of such IoT systems as security, scalability or others should also be considered and at least be maintained.

1.3 Outline

In chapter 2 an extensive analysis of a variety of bricks required for different replication algorithms is conducted. This includes a closer view on IoT properties, Information Centric Networking as promising base for IoT systems, algorithms concerning cache localization, cache replacement and cache coherency. Further, existing replication approaches in Peer-to-Peer systems and Database Management systems are examined and insights in different selection criteria for which content to replicate are given. At last existing replica discovery algorithms are analysed before the requirements for the desired strategy are gathered from the previous analysis. In chapter 3 five implementations of existing replication mechanisms are closely observed. Therefore, the functionality of each approach is described and an evaluation is presented. After that the proposed mechanisms are compared against the previously gathered requirements to identify potential shortcomings. In the end the five strategies are shortly compared with respect to each other and the desired requirements. Chapter 4 reasons about which bricks from the analysis are suitable to build an efficient replication mechanism for IoT environments with respect to the gathered requirements. In chapter 5 a few thoughts for a potential prototype implementation are expressed, before chapter 6 suggests expressive evaluation metrics. Chapter 7 concludes the paper and at last in chapter 8 a way-marker for additional work in the near future is given.

Chapter 2

Analysis

In the scope of this work efficient data replication is aiming at two main goals. The first one is the improvement of the performance of a system and the second one is the enhanced reliability of such a system. The type of systems at stake of this thesis are IoT networks, which are distributed systems to different extents by their nature. In this chapter existing caching and replication mechanisms will be analysed and their advantages and disadvantages regarding distributed IoT environments are under observation. The targeted IoT systems might be employed on Information Centric Networking (ICN) architectures for best performance. The ICN approach is a new content-centric networking architecture that could replace the traditional IP-based internet architecture in the future. First the general IoT and ICN approaches are introduced with their chances and challenges before the for this thesis relevant aspects of the Virtual State Layer (VSL) approach are introduced. The VSL is a middleware designed for easy orchestration of IoT environments and it offers a potential framework for implementing and testing the data replication approach. Afterwards the already mentioned caching and replication mechanisms will be examined in detail with respect to IoT and ICN applicability.

2.1 Internet of Things

The term Internet of Things (IoT) describes a paradigm in which countless objects all around the globe are connected in a global network. These objects are commonly referred to as smart objects or devices and need to have certain basic capabilities as defined in [15]. According to the authors smart devices always need to have physical bodies of varying size or shape (e.g. there must be some hardware) and they need to implement at least elementary communication mechanisms. Every smart device has its

own unique identifier, a commonly human-readable name and an address that can be used for communication. Furthermore, basic computing skills are required and optionally capabilities for sensing physical phenomena or triggering actions with effects on the physical world can be possessed. Additionally, the key challenges of IoT systems have been identified as large device heterogeneity, scalability, ubiquitous data exchange, energy-optimized solutions, localization capabilities, self-organization capabilities, semantic interoperability and embedded security and privacy mechanisms.

Self-organization capabilities are quite important, because in most of the cases the user only gives an initial configuration which should then be implemented and sustained by the system autonomously. Even if the user changes the configuration this is only one change once in a while whereas IoT devices have to communicate all the time to ensure that the desired behaviour is satisfied. This is easily reasonable by looking at an example. There is a temperature sensor in a room and an actuator at the heater. In the beginning the user configures the desired temperature at the sensor and this was all about the interaction concerning the human. After that the sensor is measuring the temperature and if it differs from the configured value the actuator at the heating is requested to heat more or less. The self-organization ability is also important concerning the scalability of a system. Autonomous managing of added network nodes, devices or services is a key feature for scalable IoT systems. [19] [21]

The enormous heterogeneity of devices in the Internet of Things is mainly challenging for two reasons. First of all, there are hundreds or thousands of vendors for hardware devices and most of them have their own implementations and interfaces. Standardization efforts are a big issue in the research community to unify interfaces and APIs for groups of devices with similar functionalities. On success of these efforts the integration and coupling of arbitrary devices should be easily possible. [32] On the other side the device heterogeneity has to be looked at in terms of devices with very different functionalities. In this context devices may range from quite powerful computing nodes with lots of memory and power to very little devices with hardly computational power and memory and maybe power supply based on batteries. The latter devices are commonly referred to as constrained devices. [9]

Security and privacy of IoT systems are such huge and extensive fields and focus of countless research projects that a detailed examination would go beyond the scope of this thesis. Therefore, these aspects will only be preserved in terms of existing security mechanisms of used methods and algorithms during this work and it will be ensured that no major security leak will be exposed. For further information about security and privacy in IoT environments feel free to look at [31] or [1].

A distributed IoT environment has different characteristics than the established internet architecture which favours connections from a single mostly human user to a dedicated server. In these classic communication patterns the user requests something and a server responses to satisfy that specific request. In IoT networks devices are mainly communicating with each other and human users are involved on the network edges only. This means the vast majority of network traffic is generated through machine-to-machine (M2M) communication in which no human source or destination is involved. This means smart devices produce billions of small communication transactions while interacting among each other as host or router or both. [6]

In [30] the contents of traditional internet applications and IoT applications are differentiated. While in the past mostly large and static files were requested via the internet, IoT contents are typically small and often subject to changes. Further the generation of new content in IoT networks is drastically higher than in the traditional internet (due to M2M communication). Data is produced and requested either periodically or even continuous, as applications might only be interested in the newest environmental data or monitoring applications need a continuous live feed of certain data. Of course, there are still lots of scenarios where data is only requested once in a traditional like manner. The challenge is to support all of these communication patterns at the same time.

The authors of [5] have identified significant advantages concerning smart devices, especially sensor and actuator technologies, in terms of reduced size, growing versatility and of course affordability during the last years. This progress enables IoT technologies for a wide variety of applications including transportation and logistics systems, healthcare, smart environments, personal and social systems and futuristic applications. How exactly such an IoT system is set up finally depends strongly on the application domain. The extant of such cyber-physical environments may range from small local networks inside a room over larger networks connecting several buildings to huge global networks connecting everything and everyone.

2.2 Information Centric Networking

The evolving and altering usage of the internet during the last years has already triggered researchers to reconsider the current internet architecture. Dedicated host-server connections are still needed in certain cases, but nowadays one-to-one connections are not satisfying most of the requirements of everyday internet users. In fact, one-to-many or even many-to-many communication is needed, as most websites offer certain content which is accessed by many people. Or online contents get regularly input from different sources and these contents are requested by many users. [42]

This change in the usage of networks led to a differently structured approach known as Information Centric Networking (ICN). There is a lot of research about different ICN approaches, but the essential aspects are the same. Instead of addressing a host or server directly via end-to-end connections, uniquely named data is addressed and accessed independently of its source and location. The location independence of data is enabled by hop-wise replication of content throughout the network and in-network caching. Besides the arbitrary localization of content ICN relaxes the urge of permanent connectivity between all nodes of the network. In fact, ICN seems to be a very promising approach for the deployment of distributed IoT environments. [26]

For IoT scenarios the authors of [55] have shown that ICN-based architectures are superior to IP-based architectures as several simulations resulted in a significant reduction of network traffic and energy consumption for the ICN approach. They also described ICN as an architecture that favours multi-party communication and decouples senders and receivers. This is a very promising characteristic for IoT systems, thinking for example of sensor data that is important for several other services. The decoupling of producers and consumers of content is further beneficial as IoT networks should be extensible in an easy manner. If the location of a new device or processing node in the network is not relevant (because the location of their produced data also isn't), adding and removing devices is facilitated. Needless to say, that this in fact boosts scalability of a system largely.

The hop-wise replication spreads content over the network in different ways depending on the used algorithms and mechanisms. In most of the ICN approaches every device or node participating in the network can be a cache node what enables the previously mentioned in-network caching. In this way a requested content is once requested at the source and on its way back to the requester, the data is stored in cache nodes and can afterwards be retrieved from a nearer cache node in comparison to the distant source node. Overall this can lead to a significant improvement in terms of reduced network load and bandwidth usage. In the end the efficiency of in-network caching is also affected by the network topology, because the most suitable caching mechanisms may differ for tree-like, hierarchical or grid-like structured networks.

Stating that every node in an ICN network can be a cache node is theoretically true, but considering constrained devices or nodes can lead to a problem at real life application. The in-network caching needs to support more or less dynamic caching approaches to shift loads from constrained devices to devices with more spare power and memory. [62] Another solution may lie in the design of the ICN network. If all constrained devices are connected to more powerful network nodes, the whole caching, reasoning and processing can be done in these gateway nodes.

In fact, ICN is only a concept for a future networking paradigm and not a concrete approach. Therefore, the Named Data Network (NDN) and Content Centric Networking (CCN) approaches are introduced shortly to look at more specific ICN instantiations.

2.2.1 Named Data Network

Every node inside an NDN network maintains three tables which are Content Store (CS), Pending Interest Table (PIT) and Forwarding Information Base (FIB). The communication happens in form of Interest and Data packets, which can be seen as requests and responses. If an Interest reaches a node a lookup in the CS is done, as the CS is used for caching data. If the requested data is in the CS, the corresponding data is sent back. If a cache miss occurs a search in the PIT is done. The PIT is used to remember forwarded Interests that have not yet been served. If a matching entry in the PIT is found, the Interest is dropped, because an equivalent response is already on its way. Otherwise a new entry is added to the PIT and the Interest is forwarded according to the information in the FIB. The FIB holds the routing information based on content names. Data is generally sent back along the reverse path of the Interest and can be cached in every intermediate node. When a Data packet reaches a node, first the decision is made, whether the Data should be cached or not. By default, the strategy is caching everything everywhere, which is quite inefficient in most situations. If the decision is made for caching a Data packet and the cache of the node is full, a replacement mechanism decides which of the already cached Data objects is evicted. [75]

2.2.2 Content Centric Networking

Communication in CCN is based on two packet types called Interest and Data like in the NDN approach. Producer nodes in the network publish content in the form of so-called Named Data Objects (NDO) which can then be requested by consumer nodes via an Interest. The location of NDOs is distributed by routing protocols. A request for a certain NDO is forwarded to a publisher node in the form of an Interest package. CCN nodes hold a Pending Interest Table (PIT) to keep track of forwarded requests including the information, where the Interest came from. When a producer received an Interest it sends the corresponding Data package back on the reverse path of the Interest. A CCN node is able to cache requests as well as data objects and at each node there is a balance between requests and responses, because every request is answered by one response. One can think of it as the data package consuming the interest package. Producers have influence on Data package lifetime by applying a freshness parameter at data generation which indicates how long a Data package may be cached at an CCN node. [33]

2.3 Virtual State Layer

The VSL is a middleware designed for simplification and enhancement of smart space orchestration and it is the core of the Distributed Smart Space Orchestration System (DS2OS) [47]. With other words, the VSL provides an architecture that fits the needs of distributed IoT environments. Information inside the VSL is accessed through so-called (information) nodes, which represent either physical smart space states measured by hardware (sensors) or already processed data from software. The nodes and therefore also the information can be accessed via unique hierarchical addresses. This leads to a logical tree structure of information. The addresses are location independent and consist of context-aware names. This is the same principle used in ICNs making the VSL adapt ICN behaviour. The hierarchy implicitly describes the relation among pieces of information. A service for example may be available as node1/service1 and has two functions which offer information about temperature and humidity. These values are uniquely addressable as node1/service1/temperature and node1/service1/humidity. In this way an aggregation of services, functions, devices and much more can be obtained. Information nodes are stored in the VSL permanently in a distributed manner and with one or more types. This can be used to address a group of entities in the network, as for example all window blinds. Additional attributes of every single information node are version numbers and time stamps, which are useful when using asynchronous services that are interested in when information was produced.

A publish/subscribe mechanism allows nodes to subscribe for other nodes (if they have according access rights). When a value changes at a node, all other nodes that subscribed to the node with the change get notified. The new value nevertheless is not directly received. This makes sense because there might be services that are not interested in the exact value of a node, but only in the fact, that the value changed. But if the subscribing node is interested in the changed value, an additional request needs to be sent to the producing node. To speed up the latency virtual nodes were introduced in the VSL. Virtual nodes are addresses that do not point to nodes of the network but rather point at services directly. In this way information can be retrieved from the service directly without intermediate storage inside the VSL.

The VSL is realised as a Peer-to-Peer (P2P) overlay of Knowledge Agents (KA). KAs are nodes in the network that are powerful enough to run an instance of the VSL each. Information and sensor data are stored on KAs in the form of information nodes as already mentioned. The services in an IoT environment run on these KAs and devices like sensors and actuators are connected to a nearby KA. Due to this design services and devices don't store information, instead the KAs hold all the data on disk. This

grants improved resilience as services can be down, but their data is still available at the KA. The local storage at KAs lowers the network traffic compared to central storage, as data is mostly used by services close to the producer and therefore on the same KA. The current implementation of the VSL doesn't use caching between KAs with the advantage of strict data coherency in the whole network, as every piece of data is only stored at one KA. Nevertheless, if caching would be enabled at the level of KAs, the VSL might experience further improvements in terms of resilience if a KA malfunctions and reduced network traffic for communication among different KAs. The VSL implements an information-centric networking architecture. Due to the unique addresses the location of a specific service or KA is not relevant. A KA has a Knowledge Register, which is duplicated among all KAs and provides a local directory of the whole network structure for fast searches on VSL node types. In this way every KA knows about every other KA and every information node in the network at every time, but not their data. The distribution of this structuring data enables the ICN-like data location independence. The KAs are updated periodically to keep all Knowledge Registers up to date. There are also mechanisms to handle failed KA updates.

In terms of security and privacy the VSL implements access rights for every information node and service in the network. The access rights prevent services to get information which is not intended for them to be seen. The same goes for writing data to nodes. Access privileges can be set for single identities or groups. The communication between KAs is always encrypted and requests for information must be authenticated at every time. This kind of information management establishes security and privacy by design. Joining the overlay of KAs is the most security-critical task, as a compromised KA might be able to harm the whole network in drastic ways. Therefore, Knowledge Agents authenticate each other with signed certificates containing public keys.

The interface of the VSL for service programmers offers a fixed set of methods. With the search method nodes (information) can be gathered by their type. In this way all devices of a specified type (for example /light/lamp) can be identified but also all nodes with certain states can be addressed (for example all lights that are currently switched on via /light/lamp/isOn). The get method reads information from the specified node if the access rights allow it. In the same way the set method writes values inside a node with respective write access. The lock and unlock method can be used to lock read or write access to a subtree. By default, the lock is expiring after some time and can be renewed if necessary. Subscribe and unsubscribe trigger notifications on node changes as already explained. For the creation of new nodes in the VSL the method addSubtree can be used and removeSubtree for its deletion. At last the registerClient

and unregisterClient methods are used to connect or disconnect services to or from the VSL.

In short terms the VSL enables virtualizing smart environments in a secure and information-centric way. By utilizing the advantages of ICN architectures the VSL provides promising mechanisms for data distribution, addressing, scalability and resilience. The description of the Virtual State Layer is presented with more detail in [48] and [49]. The targeted approach of this thesis for an efficient data replication mechanism in distributed IoT environments shall extend the existing VSL by adding inter-KA caching to improve resilience against KA failure and to reduce network traffic between KAs while maintaining the systems scalability, consistency and security.

2.4 In-Network Caching

Caching is an extensively researched area for traditional systems like Web, P2P or CDN. The main goals are the reduction of bandwidth consumption and network traffic. The same goals are targeted in ICN networks as well, but there are new features that have to be taken into account when it comes to ICN's in-network caching as it's commonly called. Traditional caches are mostly application-dependent and isolated in contrary to ICN caches, which are designed to be transparent for applications. This means applications just request data as they do from the data origin. The response can come from a cache and the application doesn't even know that the data is not received from the source. Furthermore, caches are visible on the network layer and one cache can be utilized by multiple different applications. Cached objects have platform independent unique names and are not bound to their origin. Thus, it can't happen that two identical data objects received from different domains are treated as different contents like in traditional caching systems. Besides transparency another feature of in-network caching is ubiquity. While traditional caching systems often use fixed and predetermined nodes for caching (leading to linear cascading or hierarchical tree structures) in ICN networks every node can be a cache node and the cache localization is done dynamically. The caching topology can evolve towards arbitrary graphs. [72]

Minding transparency and ubiquity of in-network caching, strategies and mechanisms for optimal cache localization, replacement and coherency are analysed in the following subsections.

2.4.1 Cache Localization Strategies

When looking at a dynamic in-network caching system with multiple nodes, probably the first question is about where content should be cached at.

In [42] the localization of cache nodes is divided into On-path and Off-path caching strategies. The latter describe a system where dedicated cache nodes are selected in advance based on the network topology and every producer of data exactly knows which cache node is responsible. This is good for static systems with low scalability requirements. An initial configuration is applied to choose the best fitting nodes as caches and changes are relatively difficult. In On-path caching strategies the cache nodes in the system are selected at runtime and must be part of the request path in the network. This approach might be more promising for ICN-like networks as they are dynamic, scalable and volatile.

Nevertheless, Off-path caching can be a good choice under specific conditions. Hash-routing can be utilized in these cases, but every node already needs an associated cache node in prior. Edge nodes that receive a request use a hash function whose result is pointing to the associated cache node. The node then forwards the request to its cache node. If the cache can satisfy the request, the content is sent back. Otherwise the request is forwarded to the content source. Inversely when the content is sent back from the source, it can only be cached at the associated cache node of the requester. Here exist three possibilities. The first one is symmetric Hash-routing which means that the content is sent back on the reverse path of the request and therefore first reaches the associated cache node where the content can now be cached and then reaching the requesting node. When using asymmetric Hash-routing the content is sent back on the shortest path, leading to situations where the associated cache node is not in the path and thus no caching of the content is done. The last one is multicast Hash-routing, where the content is sent to the requester on the shortest path and also to the associated cache. [56]

The following paragraphs introduce different On-path caching strategies and highlight their advantages and short comings.

Leave Copy Everywhere (LCE)[57]: This is the default strategy in many ICN systems. The response to a request is sent back on the reverse path. In every forwarding node a copy of the requested content is stored. This leads to very high redundancy in the cache network and to frequent cache replacement operations resulting in computing overhead for the replacement algorithms. On the pro side this strategy can perform good in the case of flash-crowd events. In reality even randomized selection of cache nodes performs better for nearly all scenarios.

Leave Copy Down (LCD)[37]: LCD caches a copy in the next cache node in downstream direction after the requested content is found. This means content slowly moves towards the requesting node and only moves one node further with every request. One can

imagine the content growing in the direction where it is frequently requested from. Popular enough content makes it to the network edges where in most cases the users are located. This results in the relaxation of the network traffic after a relatively short time. This is especially good if there is at least a significant amount of data that doesn't change too rapidly.

Move Copy Down (MCD)[37]: Similar to LCD in MCD content moves towards frequently requesting nodes. The difference is that content is deleted at the node with the cache hit. This approach reduces cache redundancy more aggressively than LCD and shifts contents through the network towards its highly popular locations. This reduces the overhead at intermediate network nodes if the requesters of data are quite always the same. If data is requested from arbitrary locations at the network edges, it might be more efficient to keep copies at intermediate caches.

Probabilistic Caching[53]: The Prob Cache approach is a randomized version of LCE, as it stores a copy of the requested content on every node of the reverse path with a certain probability p. For p = 1 the Prob Cache degenerates to LCE. The probability on each node can vary meaning p doesn't need to be the same for all nodes on a path. Compared to LCE the redundancy is reduced and the caching of as many unique content items as possible along a path is targeted. As already mentioned these approaches perform better than LCE in most of the cases. Nevertheless, the difficulty is the optimal determination of p for every node. Furthermore, the evaluation of probability-based approaches is difficult as the behaviour for the same scenario differs due to the random part of this approach.

Betweenness Centrality Caching[12]: In this approach content is only stored once on a request path. The node to cache the data is the one with the highest betweenness centrality value. This value can be pre-computed for every node in a fixed network topology. For every pair of nodes in the whole network the shortest path is computed and the number how often a node lies on such a shortest path describes the betweenness value. If two nodes on a path have the same value the content is cached at the one nearer to the requesting node. Problems might occur in networks where nodes join and leave the network dynamically, leading to the re-computation of all the betweenness values, which can be expensive in terms of computational resources. This approach guarantees a quite low redundancy of content in the network and stores content where it can be accessed from many requesting nodes, but on the other hand bottlenecks can occur at the most traversed network nodes.

Edge Caching[22]: In this strategy content is only cached in nodes at the edges of a network. What exactly edges are depends on the network topology and definition,

but most of the time these are nodes where user interaction is happening. Under the assumption that users are also mainly generating requests, this approach might serve quite well, as the cache nodes are actually the requesting nodes or are at least located near them. Again, if the same content is requested from nodes on arbitrary network edges, this can lead to high redundancy.

2.4.2 Cache Replacement Strategies

After deciding at what nodes inside a network content should be cached, another question follows. What content should be evicted from the cache if it is full?

The authors of [51] have classified cache replacement strategies into five categories which are recency-based, frequency-based, recency-/frequency-based, function-based and randomized strategies. Most of the replacement strategies belong to one of these groups but often they are expanded with subroutines of other groups to perform better in different scenarios.

Recency-based: The family of recency-based cache replacement algorithms focuses mainly on when content was requested the last time. The most famous approach is called Least Recently Used (LRU) and the vast majority of recency-based algorithms is more or less based on LRU. The item in the cache which wasn't used for the longest time is deleted from the cache. If a cache hit occurs, the corresponding item's recency value is updated and it is moved to the top of the LRU list. The basis of LRU is the implication that recently accessed content is likely to be accessed in the nearer future. The advantages of recency-based approaches are the temporal locality assumption as this holds for many cases in the real world and makes this strategy quite adaptive for new popular content in the network. Furthermore, these algorithms are quite simple to implement and fast. Due to the strict insertion at the top and deletion at the bottom of the list these operations are of low complexity and the replacement decision is always obvious. On the down side of recency-based cache replacement strategies, most of the approaches don't consider object size in their eviction policy. In the most cases of networks there is content of different size. This fact shouldn't be completely ignored as such objects consume more cache size on the one side but are more expensive to fetch from the origin on the other side. Another negative point is that information about access frequency of content isn't considered at all. This could be a very valuable measure in more static environments.

Frequency-based: The frequency-based strategies consider the access frequency of cached objects as main driver. Most approaches of this type inherit the basic functionality of the Least Frequently Used (LFU) algorithm or are related to or derived from it to a cer-

tain degree. Here the popularity of content is important and is indicated by the number of requests during a certain time. The assumption that the more an object was used in the past the more likely it will be used in the future is the base of frequency-based strategies. They can in general be implemented in two ways, either as Perfect LFU or as In-cache LFU. The first means that for every data object in the network a counter exists which increments for every request of that object no matter if the object is in the cache or not. It counts beyond replacement of an object. This means every cache needs to maintain counters for every object resulting in significant memory overhead. Therefore, In-cache LFU is used more frequently. Here the counter is only maintained while an object resides in the cache. At replacement the counter is deleted and eventually restarted. In this thesis the In-cache version for all frequency-based eviction policies is observed as it is more popular in reality. The main advantage of frequency-based approaches lies in their deployment in static environments where the popularity of content doesn't change too much over a certain time. Of course, frequency-based mechanisms have also disadvantages. They are more complex than simple recency-based mechanisms, as they can't be held in a single list but also need to maintain at least a counter. The problem of cache pollution can occur, as frequency-based techniques are not able to handle dynamic, fast changing content. Data which was popular a year ago might stay in the cache very long even if it is not requested anymore at all. Therefore, other (for example recency-based) mechanisms need to be added to pure frequency-based approaches. These other mechanisms or values can then also be used as a tie breaker in the case of equal frequency values.

Recency-/Frequency-based: The hybrid class of recency-/frequency-based replacement strategies uses recency, frequency and depending on the approach maybe even more metrics for the replacement decision. The motivation behind these types of algorithms is to overcome the disadvantages of either solely recency-based or solely frequency-based strategies. Proper combination can lead to good behaviour in networks with mixed contents of static and dynamic nature. One of the simplest recency-/frequency-based approaches is LRU*. Objects are stored in an LRU list with one extension. Every object has an access counter. So, the LRU list embodies the recency part while the counter makes frequency abilities available. In the case of a cache hit the content is moved to the top of the list and the counter is incremented. In the case of a cache miss, the counter of the last object in the list (the least recently used one) is checked. If the counter is zero, the object is removed from the cache. Otherwise the counter is decremented and the object is moved to the top of the list. This hybrid mechanic tends to keep content which is either frequently or recently requested or both. The other way around this means, content is only evicted from the cache if it hasn't been requested for a certain

time and if it wasn't very popular during a certain time span compared to other content in the cache. The disadvantage of recency-/frequency-based strategies is that most of them grow in terms of complexity. LRU* is one of the few hybrid approaches with low complexity. Other approaches use additional information or data structures which adds complexity and could act in contrary to the goal of improved network performance. At last these strategies don't consider content size as well. If size matters, an additional metric needs to be considered which adds complexity.

Function-based: Probably the most versatile group of replacement strategies is called function-based strategies. Here lots of different metrics can be combined in a single function which calculates a value for every item in the cache. The cache objects can then simply be ordered by their value and commonly the one with the lowest value is removed. The versatility of function-based approaches comes from the free choice which metrics are considered in the function and of course how these are weighted. The variety of metrics can range from content size, request latency and time stamps to request frequency, hop count to data source and more. The advantage of function-based replacement strategies is the consideration of multiple factors. If the factors are chosen clever and the weighting is done right, an algorithm with good behaviour for different workload situations can be obtained. The weighting inside the function can also be used to boost specific performance requirements if it is for example more important to cache bigger objects than to cache objects that are far from their origin. Depending on the application domain and the network topology the importance of certain requirements may vary. The strength of function-based replacement strategies also introduces its weakness. The choice of relevant metrics and their weighting can be a very difficult task and little changes in the network may cause the need for a reconfiguration. Some metrics might not even be as suitable as it might seem in the beginning. Latency for example seems like a very good measure for network speed, but latency is influenced on many points between sender and receiver in a network and is subject to fluctuation and uncertainty. Finally, the complexity of function-based decision-making can get out of hand. Some metrics need additional memory or pre-computation and others need to be recalculated very often. All of this can lead to performance losses in contrary to the targeted performance improvement.

Randomized: Randomized cache replacement strategies work as their name suggests. Objects in the cache are either removed randomly or the deletion is influenced by more or less randomized probabilities. Some algorithms compute a probability based on a certain metric of an object, for example its size. The object is then placed on the top of the replacement list when a cache hit occurs with this probability and the last object in the list gets replaced in the case of a cache miss. The calculation of the according

probability might follow a rough pattern like larger cache objects get a higher probability or something similar. Such a pattern needs to be lightweight as too much logic would add complexity and diverge into a function-based strategy. Randomized algorithms try to achieve a certain performance improvement while still having very low complexity. The low complexity makes these algorithms simple to implement and doesn't add significant computational overhead. On the other side randomized replacement strategies are hard to evaluate as tests with the same setup might result in major different measurements. In practice there is for nearly every case a preferable alternative to randomized strategies.

Some representatives for all of the above families of replacement algorithms are shortly explained in [51]. The basic advantages and disadvantages are the same inside of each group of algorithms.

2.4.3 Cache Coherence Mechanisms

The current implementation of the VSL has a perfect coherency as data is only available at one KA at all times. But with the deployment of ICN-like in-network caching the coherency is no longer guaranteed by design. Instead specific algorithms need to be implemented to ensure that requests are not served from caches with expired data. The algorithms designed for facing the problem of cache coherency, cache freshness or cache validity as it's also called can be classified into several groups which are explained in the following with their advantages and disadvantages.

Validation Check[58]: Caches with coherence mechanisms that are based on validation checks store content items with an associated time stamp. The time stamp is applied by the data producer at generation time. When a request reaches a cache which holds the requested data, the cache checks the validity of this specific item. Therefore, a message with the time stamp and the content name is sent to the producer to check if meanwhile there exists a newer version. If the content is still up to date the cache gets a short notification from the producer and then satisfies the request. If the data is out of date the producer can directly send the newer content to the cache which replaces the old version with the new one and forwards the newer version to the requester. The advantage of these mechanisms is a very good consistency as outdated data can only be distributed when the original data changes in the very short time between the moment the producer sent a positive answer to a validating cache and the moment the requester gets the data from the cache. Nevertheless, even if the data would be outdated in this scenario, it would be a very short time difference between the two versions. For this mechanism to work every piece of data must be assigned a time stamp at generation and either every piece of data holds the information what producer in the network generated

it or every cache in the network knows for every piece of data from which producer it comes. The latter can be easily achieved in ICNs by proper data naming.

Call-back Mechanism[35]: Freshness mechanisms with call-backs put relatively much effort on the data producers. When a certain piece of content is updated the responsible producer sends the new data or at least notifications to all affected caches. This approach implements also good coherency as data is always kept up to date in the caches independent of the data being requested. Of course, there are the same small delays due to the latency of data or notification packages. The problem with this mechanism is that every producer needs knowledge of all caches and even of the data distribution among all caches (without the latter only multicast notification to all caches would be possible and this is definitely not desired). The maintenance of such caching indices at every data producer would result in computational and memory overhead that might void the targeted benefits of the caching.

HTTP-Headers[23]: The utilization of HTTP-Headers for coherency goals emerged from traditional web-caching. Conditional Get-requests are sent from the requester to the producer with an If-Modified-Since-Header. This header holds the time stamp of the affected piece of data in the cache and at the producer the conditional get is evaluated. If the data at the producer has been modified since the time specified by the header, the new data is sent back. If the data hasn't been altered, only a small notification is sent back to the requester and the cached version can be used. In an ICN this would be realized as an arbitrary node requesting a specific piece of data. If the request reaches a cache node that can satisfy the request at any time, the cache sends the conditional get to the producer of the data. In fact, this is a kind of validation check with stricter limitations. The network needs to support HTTP communication. The other pros and cons are the same as with validation checks.

Naive Coherence[23]: Even more specialized than HTTP-Headers are naive coherence mechanisms. They are HTTP-Headers designed only for hierarchical network topologies with hierarchical caches. The conditional get is not sent to the data producer but only to the next higher cache instance. Besides the capabilities of HTTP-Headers this approach additionally depends on a hierarchical cache node structure. The obvious problem is the fact that deprecated data can be retrieved from a cache even if the conditional get was answered with no new data. This is the case when the data changed at its origin but a first level cache hadn't requested that specific piece of data from the source yet and an underlying second level cache requests the same data from the first level cache. This whole mechanism works fine for hierarchical networks with quite static data where it is not crucial to always get the newest version of data. In these scenarios the naive

coherence approach reduces the load on data producers as not every validation is done against the producer but most validations are done against lower level caches.

Expiration-based Coherence[14]: The expiration-based freshness mechanisms also use time stamps for every generated piece of data. The difference to validation checks is that the time stamp doesn't hold the generation time of content but the expiration time and date. This is commonly also done at data generation time. The big advantage is that there isn't any communication overhead through notifications at all, because every cache can directly check if the current time is further than the expiration date and therefore satisfy requests or forward them to the origin. The drawback of expiration-based mechanisms is the difficulty of choosing the proper expiration date. This date can be configured at every producer individually but the expiration time might change over time so that dynamic algorithms are needed. Such dynamic algorithms might add a lot of complexity and computational overhead.

Pre-fetching[61]: In this approach all the caches refresh themselves periodically by re-requesting the already cached items at their origin. The positive thing is that there exists an upper bound of how long expired data can be used. This maximum is in the worst case equal to the update interval. But the difficult decision-making concerning the update interval time is also the biggest disadvantage of these algorithms. Unnecessarily communication overhead is reduced by using conditional get requests for the updates. In this way no valid content is uselessly updated. In the end there must be found a trade-off between the update interval and the communication overhead. More frequent updates improve the coherency but also increase the network traffic and less frequent updates reduce the network traffic but also reduce the coherency.

2.4.4 Cache Size

Along cache localization mechanisms and cache replacement algorithms the performance of a caching system is also influenced by the cache size. For fixed other parameters an increasing cache size means that more data can be cached at every cache node and thus leading to a better cache hit ratio. On the other side larger caches may result in additional lookup overhead. As caches in IoT systems target operation in line rate, a trade-off between hit rate and lookup overhead has to be made and therefore cache size is somehow limited [72].

The optimal cache size highly depends on the network topology and the data traffic of a system. In general, smaller data objects may benefit more from smaller cache sizes than larger objects which need a certain space to store a few objects at least. Even different hardware components can make a difference due to varying access speeds, memory size

and storage costs. In a homogeneous network where all caches have the same size a simulation with different cache sizes can be conducted to determine the optimal cache size.

2.5 Resilience and Availability

As already mentioned an improved performance is not the only high-level goal of data replication in the context of this work. This section focuses on the improvement of the resilience of a distributed IoT system in terms of content replication and content redundancy. In this way content is still accessible (at least a certain time) even if the content origin is not available.

In the literature only very few material focuses on replication for resilience, availability and fault tolerance in ICNs or the IoT. Most of the research effort is put on performance improvement in ICNs as already covered in the previous section about in-network caching. Of course, there is often a correlation of performance and resilience in many approaches. But the lack of pure resilience mechanisms in ICNs leads to the analysis of replication in related fields such as P2P systems and Database Management Systems (DBMS), which offer a lot of work putting stress on fault tolerance, node failure, data resilience and availability.

2.5.1 Replication in Peer-to-Peer Systems

Depending on their implementation Peer-to-Peer (P2P) systems offer a wide range of beneficial properties of decentralized distributed systems such as scalability, shared resources, locality independent collaboration and more. On the other side special interest needs to be put on availability and reliability of such systems. The most common way to ensure availability and reliability in P2P systems is to apply fitting data replication techniques [74]. In such P2P systems with replication either whole files or data objects can be replicated or any chunks of it, but in both cases redundancy is intentionally added to increase the data availability in case of peer failures. Static data that doesn't change after creation can relatively simple be distributed to several peers in the network and all other peers can just read data from any node that holds a copy of the desired data. This simplicity changes drastically when data in the system can be altered after its creation. This is definitely the case for most P2P systems and leads to the urge of consistency mechanisms for dynamic replica updates. If the consistency challenges can be overcome, replication improves distributed systems in availability, reliability, fault tolerance, scalability, performance and "Fail Safe" infrastructures [64].

P2P Data Replication and Update Management

The authors of [29] have classified replica control mechanisms into three criteria. The first is where updates take place, the second is when updates are applied to all replicas and the third is how replicas are distributed over the network. In the following these three criteria are analysed more deeply.

Where Updates Take Place: For this issue two general approaches exist. Single-master replication on the one side and multi-master replication on the other side. The single-master approach has exactly one primary copy of the replicated data and may have several copies at slave nodes in the network. The master is probably the node which generates data, but this is not a must. In any case the master is the only node with read and write access on its replica, which is therefore the original data. Every other replica in the network can only be read by all other nodes. The advantage of this approach is the centralized update point for every replicated piece of data. If data can only be altered at the master (what can also be seen as the origin), concurrency of updates can easily be handled as the master can only perform one update after another. The down side of the single-master technique is a single point of failure. If the master node is down, no updates can be performed on other replicas and the data availability can suffer. At least data can still be read from replicas as long as the data is not deprecated. The initiation of update propagation can be done in two ways, namely push or pull mode. In push mode the master is responsible for triggering updates at the replicas by telling them to update their data. In pull mode the slaves request new updates from the master if there are any. Which one is superior depends on the types of data in the network. If data is not used at all replicas it can save network load and bandwidth if updates are only pulled at replicas where data is actually requested. For other scenarios it is more important to have the newest version as fast as possible, then push mode is the better option. In the multi-master approach all replicas are primary copies or the other way around none is. Anyway, the replicas are equal and every node in the network can perform updates on its local replica. That means read and write operations can be performed at multiple instances of the same data which leads to a more flexible approach than single-master. In the case of a master failure, the data can still be altered at other replicas in the network and even if the faulty master is offline for a long time, the data can be used as if nothing happened. The disadvantage of multi-master approaches is that updates can occur in concurrency. If two nodes make a change on the same data at their local replicas at the exact same time, the two replicas who are supposed to be the same differ and may lead to inconsistency. If a system relies on strong consistency, additional algorithms are needed to ensure the consistency property.

When Updates are Propagated: Before reasoning about when to propagate updates one has to realize that there are two possible basic approaches for replica placement. The first is called full replication and differs significantly from the second one called partial replication. In full replication systems every participating node holds a copy of every object in the network. It can logically be thought of copying whole network nodes to other nodes or at least exchanging copies of everything what differs between two nodes. As a result, every node in the network should have the same memory capacities and every node can take every other node's place in the case of a node failure. This leads to very robust systems in the case that there are just a few nodes in the network with a lot of memory. Such approaches are often used in safety critical systems to ensure that the outage of one node doesn't affect the system behaviour. On the other side full replication puts high burdens on the memory capacity of the system. In contrast to full replication, partial replication mechanisms store only subsets of data at replica nodes. This means not every piece of data is present on every node, but only at certain nodes and every node on the other side holds only certain data. This approach can be utilized to hold only data that is really needed or popular at respective nodes. Partial replication needs far less storage space and can also reduce bandwidth usage because updates are only sent towards nodes that hold the affected data. On the down side this approach may lead to limitations in load balancing as not every node is able to perform every type of transaction as in full replication [16]. If popular data is only present in one node it can lead to significant network performance issues. Therefore, it requires good planning and decision-making regarding which data to replicate and at which nodes to store these replicas. This indirectly includes the decision of how many copies should be spread over the network.

How Replicas are Distributed: There are two different ways in that replication operations can be handled. These are synchronous and asynchronous replication or as they are also referred by [29] eager and lazy replication. In synchronous (eager) replication an update at a node directly triggers updates at all the replications of the altered data in the network. The initial update where the actual change happened gets committed after all the replications have been updated too. This means the update starts at the node where the change is made. Then before this process is finished the other replicas are instructed to adapt the change. Every replica node replies to the triggering node after it has altered its local copy. And in the end the initially changed replica (or the data origin) finishes the update at its local memory by committing it (but only after all replicas have answered and made the change). The advantage of synchronous replication is that strong consistency is ensured as all the replicas and the original data are always the same in the whole network. The disadvantage of this approach is that

an update takes quite a lot of time because all replicas have to be updated before the initial change is committed. If there are frequent updates to the same data and/or many replicas in the network this might lead to performance issues as data might not be accessible during an update process. The asynchronous (lazy) replication approach does not wait for the response of replica nodes. It actually finishes an update at the triggering local node and immediately commits the change. Only then the update is propagated to the other replicas in the network. The advantage of this approach is an increased availability compared to synchronous replication, because the data at the origin is directly available after the update even if a replica is not reachable. In synchronous approaches the original data is blocked until all replicas have responded. Asynchronous replication can further be classified into optimistic and pessimistic approaches in terms of update conflicts [46]. Pessimistic approaches add a kind of single-master approach to the lazy replication to achieve a so-called one-copy-serializability [7] which describes mutual consistency among replicas. Updates are in general done asynchronously but can only be triggered at one primary node for each data item. In this way there can't exist concurring updates for the same data, as the primary node performs one update after another. But it is still possible that two replicas are not consistent for a short period, but this is due to the asynchronous replication and not due to concurrently appearing update requests. At last needs to be said that a problem occurs if a primary replica node is down, because in this case no updates can be executed until the primary node has recovered. While pessimistic approaches still block a single replica while it is updated, optimistic approaches update in the background and might be requested during an update. This could lead to conflicts what is the reason why optimistic approaches are only used in scenarios where conflicts will not or at least very rarely occur. Another difference between pessimistic and optimistic approaches is that in optimistic replication every replica can be updated, not just a primary one. Overall this increases availability of optimistic lazy replication compared to pessimistic lazy replication and also yields better scalability, as peers can dynamically join or leave the network and take part in the replication process. Of course, these advantages come at the cost of increased possible inconsistency due to updates at arbitrary replicas without any synchronization and possible concurrent update conflicts.

P2P Replica Placement

Replica Placement has to face similar or partially even the same challenges as Cache Localization Mechanisms. Nevertheless, nine different replica placement techniques that have been identified for P2P systems are shortly introduced to get an overview of different approaches for the local distribution of data. Some approaches give concrete

instructions where data should be stored exactly while others propose more general ideas of data distribution.

Owner Replication[41]: In Owner Replication techniques a replica is only generated at requesting nodes. This means a node requests certain data and when this data arrives at the requester it is stored for future requests. In fact, this is quite the same approach as for edge caching. Here the network is populated with replicas of a certain data item proportional to the number of requests for this data item. The non-active nature of this replication mechanism might be more suitable for performance goals than resilience and availability of data. On the pro side owner replication has very little storage consumption or at least the storage space is usefully consumed where the corresponding data is needed. On the other side it takes quite a long time to populate the network with replicas as replicas of a data item are only created after that item was requested. For rarely requested data it takes very long to achieve the intended redundancy.

Path Replication[41]: The path replication techniques copy requested data to all nodes on a path from source to destination. Here active replication takes place, as the source pushes replicas to all the nodes on the path. This is very similar to the Leave Copy Everywhere caching approach. Peers with higher degrees forward more data than peers with lower degrees and are therefore part of much more paths between two arbitrary nodes of the network. With other words, the higher the degree of a node the more different data items are replicated on this node. This approach yields a quite high redundancy and therefore also large memory usage. On the other side the high availability of data at many nodes guarantees a good search performance. But it needs to be stated that in case of a node failure of a high degree peer, the system takes a long time for recovery. Just as Owner Replication, Path Replication is based on the requests for data, so data that is not requested at all won't be replicated.

Random Replication[41]: Random replica placement tries to make a compromise in data availability and memory overhead. Commonly random replication reaches quite good results in terms of small search delays but on the other side random algorithms are difficult to implement and even more difficult to evaluate as the results for the same test setup may differ due to the random decision-making. In most cases there is a better suiting strategy than randomness, assuming that a lot of knowledge about network topology, application domain, data traffic and more is available at the time the replication algorithm is implemented and applied. If that knowledge is initially not gathered randomness yields a reasonably good approach for multi scenario mechanism.

HighlyUpFirst[45]: This is a heuristic algorithm that favours peers with the highest uptime in the network as replica storage nodes. The idea is that peers with a high uptime seem to be less vulnerable to node failure and therefore improving the resilience and availability of data. A limitation of this approach is the required knowledge of the network topology to be able to query all peers for information about their uptime. Also, the nodes with the highest uptime have a very high memory consumption and might not be able to handle all the requests. On the other side the algorithm would adapt to a node that fails by just picking another node with high uptime. Of course, the problem can then arise again and nodes with the highest uptime fail all over again if a high burden is put on them.

HighlyAvailableFirst[45]: Similar to the previous approach this is also a heuristic algorithm with the difference that not the uptime of a node is determining where to store replicas but the availability of node is. Again, the availability of data is improved under the assumption, that highly available peers also stay highly available in the future. The node availability refers in this case to the time a peer was available in the network during a certain period. It is indeed quite similar to the previous approach and also faces the same problems. Further description of how node availability can be defined is described in [45].

Uniform Replication[41]: A uniform replication strategy treats all data and nodes equally. Replica locations are selected in a uniform manner across the whole network and approximately the same number of replicas is created for all data items. The level of redundancy of data can be controlled relatively easy according to application requirements. The replicas are distributed over the network with about the same distance to the next replica. This results in an improved search behaviour and a good maximal search time for requested data. For a sufficiently good uniform distribution knowledge of the network topology is needed, at least to a certain extent. The biggest flaw of this approach is that replicas will probably be stored in nodes where they are not needed at all. How exactly nodes are picked for replica storage is not described, but this is surely depending on the network topology and how many replicas are wanted.

Proportional Replication[41]: In proportional replication the number of replicas depends on the data itself. Or more precisely on the popularity of a data item. The number of data replicas is determined proportionally to the demand of that data. Highly requested data is replicated more often than unpopular data. This concept is similar to Owner Replication but here it is not suggested where exactly data should be stored. All in all, this results in reduced response times and search traffic regarding requests for popular data but unpopular data is more difficult to find and suffers from longer response times.

Square-Root Replication[41]: The basic function of Square-Root Replication is very similar to Proportional Replication, but instead of using just the popularity for determining the number of replicas for a data object, the square-root of the query distribution of a data object is used. If a network is populated with replicas via Square-Root Replication the number of hops that need to be traversed for data item search is highly reduced. In fact, it is not stated how exactly the target nodes for replicas are selected and a track of data item requests has to be maintained.

Pull-Then-Push Replication[38]: This replication technique consists of two phases, a pull and a push phase. In the pull phase a node requests any piece of data in the network and gets the demanded data from a node that holds the specified data. After receiving the data the requesting node replicates it and pushes replicas to all its neighbours. The positive effect is that providing nodes are unburdened as the creation of replicas and their distribution is shifted to the requesting nodes. On the other side the network overhead is increased because every neighbour gets replicas which also results in an increased overall memory usage.

General Thoughts on Replication in P2P Systems

The authors of [64] are analysing P2P Replication with a main focus on data availability and fault tolerance and describe performance gains as a beneficial side effect. There it is stated that a system's performance can greatly benefit from replication if replicated data is only read. If updates on data and its replicas have to be executed it might be possible that the beneficial effect of replication is neutralized by the effort of maintaining consistency between original data and replicas. For systems with strong consistency requirements and frequent update operations the system performance suffers from high synchronization overhead. This leads to the necessity of performance considerations. Even if resilience is the main driver for replication in a system, its implementation may not result in a significant performance reduction. Deciding about the number of required replicas has a high impact on the resilience, but decisions about the location of replicas mainly has to minimize additional network traffic. In the end replication is the art of finding a compromise between improved resilience, a satisfactory degree of consistency and a manageable drawback in system performance. It is again stated that in an optimal scenario the replication can achieve improved performance even if intentionally applied for resilience reasons.

2.5.2 Replication in Database Management Systems

A research field that is closely related to replication are database systems or more precisely Distribute Database Management Systems (DBMS). In distributed DBMS

multiple copies of data are stored at multiple sites inside a distributed system. This ensures the operation and availability even if some storage sites have failed. The main challenge and also the most important issue in such systems is the maintenance of the data consistency throughout the whole system. The target of a distributed DBMS is a behaviour as if there would only be one single instance of each data regardless of the number of replicas. This means a user (human, machine, service, etc.) should not be aware that a request got served from a cop and not the original data and there shouldn't be a difference. This property is commonly called one-copy serializability. [7]

In general, replication in DBMS can be classified similar to the previous P2P techniques. Synchronous or eager replication forces all replicas to be updated before the initial update is committed. Asynchronous or lazy replication instantly commits an update and just then forwards the update to other replicas. Again, eager replication ensures better consistency but needs additional mechanisms for handling of node failure and replicas are blocked during an update until it is committed. Lazy replication has not as strong consistency properties but doesn't block a replica during an update (or blocks it only very short).

Another distinction divides DBMS replication based on the concept of its primary copy in group and master replication [29]. Group approaches allow updates on every replica in the network and are therefore also referred as update anywhere. Master approaches on the other side only allow updates at a pre-selected primary copy of the data and all updates are delegated to the primary replica. Other replicas can only be read.

Most of the time databases focus more on resilience, availability and reliability of data instead of performance. But the consistency is always of prime importance. Keeping this in mind two different protocols for database replication are presented in the following paragraphs.

ROWA

One of the simplest replica control protocols is the Read-One-Write-All (ROWA) protocol. Nevertheless, this approach depends on knowledge about what data is replicated and where the replicas are stored. If a read on any piece of data in the network is requested, the system serves the response from the closest replication. Write operations on the other side require updates at all replicas. With the requirement of prior knowledge about replica placement the read requests can be satisfied fast and simple and the write operations can also be applied quickly at the affected nodes. No complex discovery algorithm is needed but the question remains how the system knows of all replicas and their location. As typical for many DBMS this seems to be a good strategy for centralized systems where a central topology service keeps track of replicas and their

distribution. All in all, the performance of a system can be improved greatly for read requests. Write operations on the other hand may significantly reduce the performance. Additionally, if a node with a replica is not available during an update, the write operation can't commit. For these reasons the extension ROWA-Available was introduced. This further developed version of ROWA can handle node failure during write updates on replicas. The read is performed analogue to ROWA, while the write operation only updates available replica sites. Offline nodes that store replicas are ignored. The problem is that nodes are able to recover after a failure and respective nodes store outdated data then. This case somehow needs additional attention at node recovery. [27]

Quorum Based

The second approach for handling writes on replicas is based on the update of only a subset of the replicas without struggling with data correctness or consistency and relies on quorums [7]. Again, information about the distribution of replicas in the network needs to be available. Every replica in the network gets assigned a non-negative vote called quorum. After that thresholds for reading and writing are defined for all data items. Here it is important that the sum of read and write threshold and the doubled write threshold for every data item are greater than the total vote assigned to that data item. This results in the property that any pair of quorums has a non-null intersection leading to a behaviour where any set of replica sites has a minimum of one copy with the latest version. To read from or write to any data the corresponding read or write quorum has to be collected. Such a quorum consists of any set of replicas where the sum of their votes surpasses the threshold. If all replicas initially get the same weight, meaning that every vote has for example value one, and the thresholds are selected as half of the replicas plus one, the decision if reading or writing is allowed is based on a majority. With other words, if more than half of the nodes that store replicas of the currently requested data agree a read or write can be executed. In this way consistency is maintained even if only a subset of replicas is updated. This means that data can be altered and accessed in a consistent way even if node failure of multiple nodes occurs.

Deployed DBMS Replication Approaches

In the following some replication algorithms are shortly described that made their way into different deployed systems.

The object-oriented systems Arjuna [50] supports active as well as passive replication. The default mode is passive replication with a primary copy where all updates are executed. The primary copy commits its update and then propagates the update in a lazy fashion to other replicas. In the active replication mode updates can be executed at

every replica. Therefore, mutual consistency between replications is maintained which results in synchronous behaviour.

Coda [36] is a distributed network file system where multiple servers respond to read requests of clients. Updates are performed at all servers by using a ROWA protocol to increase availability and gain fault tolerance. If one server fails another one takes action and the user doesn't even notice.

A different approach is implemented by the distributed file system Deceit [60]. Here concurrent reads and writes can be handled by utilizing write tokens for update operations. It is a configurable approach that offers different consistency levels ranging from no consistency to semantics consistency. Due to the write-token this is a synchronized replication, as updates can only be executed from sites that currently have the token.

Again, a primary copy-based protocol is used by the distributed file system Harp [40]. This is a pure server protocol and client caching is not supported. Servers are divided into groups where for each group a primary server is chosen and several secondary servers. Remaining nodes are designated as witnesses. If the primary site fails, a secondary takes its place. If enough primary and secondary servers have failed a witness is promoted to a secondary. In this case backups of the data are created to ensure future availability and to prevent data loss. Read and write transactions are performed in ROWA style.

Mariposa [59] is a distributed DBMS initially designed for scalability of distributed data servers and local autonomy of nodes. It deploys an asynchronous replication algorithm which means data can be outdated at certain replicas. Updates are forwarded to other replicas within a predefined time limit. This works very well for systems that can handle stale data for at least a certain time interval. An economic replica management is used to buy copies from other nodes and negotiations are performed before paying for updates.

The main focus of the distributed object-oriented DBMS Pegasus [2] is the support of heterogeneous data sources. Global consistency is achieved due to synchronous replication mechanisms.

At last Sybase [27] as a distributed DBMS supports heterogeneity and uses asynchronous replication. The replication is done on transaction level by distributing copies of stored procedure calls. Only rows affected by transactions at the primary node are replicated to remote nodes. Changes records are passed to a local replication server, which forwards the updates to specified distributed replication servers. After that updates can be

executed on replicated rows. The replication server ensures the correct order of all executed transactions. This guarantees consistency of the data.

2.5.3 Selection of Content

Some of the previously introduced replication mechanisms and techniques give instructions what content should be replicated and others only describe where and how the replication should be done. For this reason, this section makes some considerations of how the data to replicate can be chosen. The selection of the data can be done in different ways. First of all, two general approaches have to be distinguished. Either all the data present in the system gets replicated or only specific data is target of the redundancy mechanisms.

The first results in an easy policy that replicates every existing piece of content in the network. Regarding the content selection for the replication this approach doesn't need any reasoning or decision-making as everything gets replicated. But as already mentioned this might lead to lots of redundant contents which must not or even should not be stored more than once in the network. Furthermore, the memory overhead and the frequent updates triggered by changes of content would reduce the system performance.

If an approach is wanted that does not automatically replicate every existing piece of data in the network, the question remains how the replication algorithm knows what to copy and distribute and what not. A possible strategy is to take the characteristics of data items into account. Similar to some of the placement strategies, popularity could be used to determine if a data item should be replicated or not. The problem is that meta data for every data object has to be maintained over the whole system. Furthermore, unpopular data might not be replicated at all. There might be scenarios where certain data is only used very rarely but it might be very important nevertheless.

Another possible approach for content selection in partial replication systems is the tagging of data. In this case content is marked for example in its meta data with a tag that triggers replication. Only data that has the specified tag gets replicated. The advantage is that the selection of redundant data can be done very specific. At the same time the question regarding the number of replicas can be answered. By tagging every piece of data it can be individually configured if resilience should be guaranteed and maybe even which level of resilience in the form of k-redundancy should be provided. All this is done on data item level. The hard part might be the process of tagging the data. Either an additional mechanism for the tagging of data has to be employed or the tagged data is already considered at data generation. A default configuration at

system level or at network node level could facilitate the whole process. For example the whole data at a certain node could be replicated to achieve 2-resilience and some specific data items whose availability is even more important could be tagged for 3-way replication. Of course, this works only on a system that supports distribution of data with its associated meta data.

2.5.4 Replica Discovery

After successful replication of important or even all data in the network some degree of resilience may be achieved which is a good thing as this is one of the main efforts of this thesis. But the existence of replicas alone is not enough. If a source node fails other nodes have no indication of where the maybe existent replicas are stored. Due to the IoT and ICN properties that should be exploited a central entity for keeping track of replica placement is not acceptable. At this point network resource discovery comes in.

According to [39] search algorithms for decentralized unstructured P2P networks can be classified in two ways based on query forwarding. The first is deterministic which means that queries are forwarded in a deterministic way by using prior information about query paths for routing. The second is probabilistic. Here queries are routed either probabilistic or randomly. Random approaches select forwarding nodes by chance. Another distinction of search techniques for unstructured P2P networks is described by [66]. The authors have split resource discovery algorithms into two major groups, namely blind search techniques and informed search techniques. Blind search approaches do not use any information about object or peer location and act haphazardly. Informed search approaches utilize meta data to improve the search performance. In the following several blind and informed search techniques are analysed.

Blind Search Techniques

As already mentioned blind search techniques don't use any information about network topology, request routing, meta data or any other data that might give insights into localization of desired resources in the network. The absence of such information leads to the use of so-called flooding techniques to discover for example replicas of certain data. In the following different blind search techniques are introduced and their advantages and disadvantages are discussed.

Breadth First Search (BFS)[66]: A common employment scenario for BFS are file-sharing systems based on P2P architecture. The node that issues the request for a certain resource (in the case of this thesis replicas) checks all its neighbours if they can satisfy the request. If the request cannot be satisfied by any direct neighbour,

these direct neighbours forward the query again to their neighbours and so on until the target node is reached. This is a very simple approach that has a very high success rate but obviously the load that is put on the network is also very high. The query is literally forwarded over all paths in the network and links with low bandwidth can become bottlenecks. As an effort to reduce the networking overhead time-to-live (TTL) values are attached to the query which also avoids endless search loops. A query gets propagated through the network until either the target node was found or the TTL expired. An alternative of TTLs is the attachment of a maximal HOP value that defines how often a request is forwarded to the neighbours. With every forwarding the HOP count is reduced until it reaches zero. But even with these expansions BFS yields very high network overhead.

Random Breadth First Search (RBFS)[25]: RBFS developed from BFS with the intention of reducing the networking overhead during flooding-based resource discovery. Instead of forwarding a query to all neighbours a certain number of random neighbours is selected and these neighbours then again select the same number of neighbours randomly. TTLs or HOP counts can be applied analogue to BFS and the stop conditions are also the same. The optimal number of neighbours to select has to be defined by the search algorithm and could be fixed or for example be a fixed fraction of the available neighbours. As RBFS is a probabilistic approach its success rate is slightly lower than BFS but it has less communication overhead. All in all, RBFS still visits a lot of nodes causing high traffic.

Normalized Flooding[20]: In fact, normalized flooding is very similar to BFS but instead of flooding all neighbours only a subset of neighbours is chosen. This sounds exactly like RBFS but the selected number of neighbours is defined by the minimum degree d of all network nodes. This means every node in the network has at least d neighbours and when forwarding queries exactly d neighbours are uniformly selected at random. If d is chosen weakly low degree nodes with desired content might not be reached.

Iterative Deepening[39]: The Iterative Deepening search is a composition of multiple BFS employments. Additionally, a maximum depth is defined and the first round of BFS starts with a low depth. The depth is the same as the already mentioned HOP count. So, first all direct neighbours are queried. If the request couldn't be satisfied, a second round of BFS is triggered with increased depth. Now neighbours of second degree are also considered and this process keeps going until the request was satisfied or the maximum depth was reached. Iterative Deepening is mostly used if the number of returned data items is important, as a hit in the first round might return one replicated item whereas the second round with higher range might return more hits. The success of this approach depends on how replicas are distributed in the network. For many

uniformly distributed replicas this might often stop after the first round leading to relatively low communication overhead. But if the maximum depth is reached regularly this brings a lot more networking overhead than BFS, as the nearer neighbours are queried in every round.

Random Walks: A Random Walk as the name suggests is a path where every step is selected randomly among the available neighbours. This technique is also known as Markov chain and means that a system has no memory. More precise this means previous decisions are not considered for the next decisions [52]. Instead of BFS only one neighbour is selected which reduces the communication overhead drastically. On the other side the average time for a successful search is a lot longer. In order to reduce the search time several variations of Random Walks have been proposed. One is the deployment of k independent Random Walks [41]. The chance of finding the desired node are k times higher than with a normal Random Walk. If any of the walkers succeeds the algorithm is considered successful. In general, a Random Walk stops with either a failure or a success. Therefore, two termination methods are used. The first is again a TTL-based termination where a walker only traverses a certain number of HOPs before it fails. The second stop criteria is checking-based. Here the original query node is contacted every time before the walker moves to the next node. The communication overhead can be reduced significantly by k-random walkers but the overall performance is not really satisfying because of its random selection and inability to react to varying query loads. At last there is the two-level random walk [34] where the issuer node first selects k1 random walks with a certain TTL. When the TTL of the k1 random walks expire k2 random walks are started with another TTL at every node that has been reached by a random walker at that time. This approach has for the same number of walkers a reduced communication overhead compared to k-walker approach but on the other side it suffers from longer search times.

Local Flooding with k Independent Random Walks[20]: The last blind search technique combines flooding and random walks. First a local flooding in BFS nature is triggered towards all the neighbours of the issuer node. The flooding might continue for more than one step because it only stops after k new nodes have been reached for a predefined k. If the query can't be satisfied yet the k outer nodes start a random walk each. Replicas close to the origin are located fast by the first flooding phase with considerably low network traffic. Replicas further away from the requesting node are suspected to be found by one of the random walks. The network overhead is moderate as the flooding only occurs locally. If the random walks are deployed with high TTLs the communication overhead rises again and lowers the performance of the system.

Informed Search Techniques

Informed Search methods try to utilize some routing information to increase the chance of early success and reduce unnecessary communication overhead. The used parameters can be everything that increases the probability that the chosen neighbouring peer is the target peer or at least lies on the way to the target peer. Therefore, informed search approaches yield smaller response times and less communication overhead compared to blind searches but these advantages come at the cost of local overhead at peers for maintaining various indexes.

Directed Breadth First Search (DBFS)[70]: DBFS works quite similar to RBFS as queries are forwarded to a subset of neighbours. But instead of choosing the neighbours randomly peers store and use information of previous searches. Ever peer maintains the number of results that were received for each neighbour and maybe the latency of the connection with respective neighbours. The queries are then forwarded to the subset of neighbouring peers with the highest success rate and the lowest latency. Due to the forwarding to only a small subset of neighbours the number of nodes that are queried is significantly smaller than in BFS or RBFS while the success rate is improved.

Intelligent Search Mechanism(ISM)[71]: This approach is also called intelligent-BFS as peers maintain profiles of their neighbours to determine the peer that can most likely answer the query. Only peers with high probability of returning the desired data are contacted. The profiles develop over time as peers gather more and more information of their neighbours with every successful search query. Intelligent Searches work best in environments with high query locality and in networks with specialized knowledge peers. That means certain peers have data that is uniquely and only stored at this specific peer. The main advantage of ISM is the object discovery and not the message reduction. Unfortunately, no negative feedback is considered in the form of failed queries.

Adaptive Probabilistic Search (APS)[67]: Very similar to k random walkers APS also starts k random walks. The difference is how these k walkers are determined. A peer holds probabilistic data of how high the chances are that this neighbour delivers the requested resource. The probability for every data item is calculated from previous queries and is increased every time a walker succeeds and decreased every time a walker fails. Duplicate answers are counted as fail and the probabilities are adjusted after every random walk. The forwarding after the first neighbours is then also based on their probability list where only the best neighbour is selected. Random walks return on the reverse path and update probabilities for the requested data item at every peer on their way according to success or failure. APS shows improved results compared to the random walker strategy in terms of high success rate, increased number of discovered

objects, low bandwidth consumption and adaptation to varying network topologies. On the down side we have the probabilistic selection of neighbours instead of using additional parameters such as bandwidth, storage availability and more.

Adaptive Resource-based Probabilistic Search Algorithm (ARPS)[73]: ARPS extends APS with weighted probabilities based on node degrees and popularity of desired data. For every data item the estimated popularity is locally maintained and the forwarding probability is respectively adjusted. Searches for resources with high popularity are forwarded to neighbours with low probability and searches for data with small popularity is sent to neighbours with high probability. This is reasonable because commonly popular data is spread much more in the network than unpopular data. The weighted probability is smaller for high degree nodes than for low degree nodes. During a search the first step is finding the data object in the local popularity index. If there is no entry the query is flooded to all neighbours (meaning with a probability of 1). If an entry is found the proper neighbour is found depending on the probability index. ARPS is said to be as successful for low popularity content as for high popularity content and the memory usage lies under that of APS. Nevertheless, the flooding when no entry is in the local index yields high messaging overhead.

Activity Based Search (ABS)[3]: As a scalable search algorithm ABS dynamically determines the number of neighbours for every query. The number is based on the so-called activity level of each neighbour which is a metric to describe how much a node has contributed to successful searches in the past. Queries are only forwarded to neighbours with highest activity level. Furthermore, ABS creates a spanning graph from high traffic connections. After creation of such a spanning graph the number of hops for a single search can be approximated by the diameter of the graph. This algorithm performs better than k random walks but the forwarding to the most active neighbours puts relatively much load on them which may result in the reduction of the system performance because active nodes might become bottlenecks.

Preferential Walk[77]: This approach shortly called P-Walk is again a probabilistic technique based on trusty neighbours. According to search results peers assign each other trust rankings. If a peer got many successful responses from another peer it ranks it high and the other way around. Queries are forwarded to the neighbours with the highest trust ranking. This results in querying through neighbours that have the desired data with a high probability. Specific mechanisms to handle duplicate messages and partial coverage are not considered in P-Walks.

Hybrid Periodical Flooding (HPF)[76]: HPS allows fine-tuning of multiple parameters to achieve various performance requirements. The most important ability is that the

number of forwarding nodes can be changed periodically based on multiple metrics such as communication cost, bandwidth, number of successful searches via a neighbour and average number of hops to desired data. The successful searches are determined with iterative deepening which results in quite slow query processing.

Equation Based Adaptive Search (EBAS)[8]: This technique uses estimated popularities of contents to choose the random walk parameters. The popularity is simply estimated based on previous search results. EBAS has two components which are a parameter selection module and a popularity estimator. The success of random walks is highly dependent on the number of walkers k and the TTL. Therefore, the parameter selection module determines k and TTL based on the popularity of the requested data. The popularity estimator updates popularities for a data item after a respective search for that item. A table exists that correlates the popularity to certain values of k and TTL. The dynamic selection of these parameters results in a reasonable average success rate and an acceptable average overhead and delay within specified bounds. Normal random walks have only static parameters which can result in low success rates and excessive overhead.

Local Indices[69]: This scheme maintains relatively large index tables at every peer. These tables store indexes of the data of all neighbours in a system wide predefined range r. This means every node knows what data its neighbours in range r hold. For a query the local indexes are searched and if a hit occurs, the query can directly be forwarded to that specific neighbour. If no hit occurs a policy determines at what depth the next index lookup should be done. All intermediate nodes only forward the query without a index lookup until the specified depth is reached. When nodes join or leave the network or when nodes update their local data additional effort is required to update and maintain the index tables of all nodes. While the depth in iterative deepening specifies where a search should end, the radius of local indices tells how many nodes are skipped until the next local search. Accuracy and success rate of local indices mechanisms are quite good since every lookup searches an entire neighbourhood but on the other side the message production is similar to flooding and the extra cost of updates of the index tables increase the computational and message overhead further as updates are done by flooding neighbours within the specified range. All in all, the overhead for dynamic networks with volatile node and frequently changing data is enormous.

Distributed Resource Location Protocol (DRLP)[43]: Another probabilistic approach for informed searches is DRLP. A desired probability value can be obtained by adjusting several parameters. Every peer maintains a local directory (LD) in which locally stored data is referenced. Further every node has a directory cache (CD) which lists presumed resource locations at other peers. So, an entry in the LD at a certain node can be an

entry in another node's CD and the exchange of these entries is used for keeping the CDs up to date. When a request reaches a node the LD is searched for the desired data. If the data is found it is returned on the reverse path of the request to the issuer node. On the way the CDs of all nodes are updated with the LD entry of the node where the data was found. If the LD yields no hit the CD is searched and the request is accordingly forwarded to the respective node. If neither LD nor CD have the requested data the current node queries its neighbours with a certain broadcast probability. This approach overcomes the disadvantages of simple flooding by combining probabilistic query forwarding with an additional caching mechanism. Variations of the probability for broadcasting lead to varying success probabilities for searches. If the CDs are not filled in the beginning a lot of communication overhead is produced but this improves with time.

Distributed Search Technique (DST)[65]: The last informed search technique is called DST and utilizes a learning mechanism called Q-learning for decisions regarding the query forwarding. The learning process is based on several attributes like the number of successful queries, the number of results for each query, visited hops, dynamic nodes, TTL and more. With all these parameters DST yields good load balancing, high hit ratios, low network traffic and adaptability. This is also achieved by a row of supported functions which are Q-learning, two-way load balancing, priority for specialized peers, efficient handling of duplicate messages, TTL adjustment and use of past node performance in routing queries. Every peer has a Q-table which stores so-called Q-values. These Q-values are modified based on experiences of past searches. For load balancing reasons a highly loaded peer can redirect its future query load to less loaded peers listed in the Q-table. All searches terminate when either the search is successful or the TTL has expired. Certain peers can reassign the TTL to find content with low availability. In simulations DST has achieved better performance results than random walks and APS in terms of better success rates.

2.6 Summary of Requirements

From the previous sections a lot of requirements for the targeted distributed IoT replication mechanism can be derived which are summarized compactly in the following.

As already mentioned the two major high-level goals of data replication in this work are improved performance of the system and enhanced resilience of the data in case of node failure. The resilience in fact should be improved in a way that a certain level of intended redundancy can be achieved in the form of so-called k-resilience. The motivation for two high-level goals is self-explanatory but there are several other points of interest

which have to be addressed. IoT networks are commonly very volatile meaning that new devices and nodes can join the network or leave the network in a unforeseeable way. In the best case the network is completely self-organizing concerning registration and removal of nodes at runtime. Assuming such a behaviour the replication mechanism must also be able to handle such changes in the network topology and it also must maintain the existing scalability of the system. Regarding the volatile network assumption the replication needs dynamic decision-making about the best cache or replica placement. A predefined approach would not maintain the systems scalability and flexibility. This directly leads to the next requirement in form of a fast replica discovery algorithm in the case of a failure of a data source. The unstructured and dynamic network topology raises the need for also dynamic discovery algorithms. A main challenge triggered by replication is of course the consistency of the data. This means a satisfying degree of data freshness needs to be met to yield usable data for applications. Consistency is always a trade-off between system performance and data freshness. As IoT networks are deployed in a wide range of scenarios also different claims for consistency occur. Therefore, the replication algorithm should be configurable to work with different consistency levels and to be able to adapt to varying deployment scenarios. Another important question is what data should be replicated. Not all data benefits equally from replication and therefore certain selection criterion needs to be established to define what to replicate and cache and what not. This includes for example continuous sensors that produce live data which quickly varies. It might not make any sense to cache such data as it changes permanently and the targeted replication mechanism needs to be able to address this issue. ICN as promising network architecture for future IoT systems offers the benefits of network level caches to eliminate the need for application level caches. This ICN feature needs to be maintained to guarantee caching that is transparent for applications and services. In terms of the performance goal the coherence mechanisms should not cause too much communication overhead but this somehow depends on the required consistency and is therefore partly addressed by the configurable approach. All in all, the used algorithms for consistency, cache/replica placement, replica discovery and cache replacement need to be as fast and lightweight as possible to gain the targeted performance improvement. This is especially important as the exchange of data in IoT networks is expected to be of enormous amounts of transfers with very little size each. So, the algorithms need to handle the amount of operations without adding too much overhead. All in all, this contributes to the superior goal of reduced bandwidth usage and network load of the whole system. An additional requirement that boosts the previous is a targeted hop reduction for data requests which is achieved by replication and clever replica and cache placement leading to shorter response times. As IoT networks are very heterogeneous by their nature different nodes and devices have to be

taken into consideration. This means constrained devices with limited computational power, memory and storage capacity might be deployed in the network. Therefore, the cache and replica storage size should also be adjustable and configurable. In the end two very important requirements are security and privacy. These issues are very far reaching and are therefore not addressed in detail. It is rather suggested that the targeted IoT systems are already secured and the developed replication mechanism just may not expose the existing system to additional threats. The identified requirements can be over-viewed in table 2.1.

All in all, the outcome of this thesis should be an efficient replication mechanism for distributed IoT environments that boosts the system's performance especially when thinking of large multi hop scenarios and also adds significant resilience to overcome node failure for at least a certain time span. Besides these goals the replication should maintain the dynamic and scalable nature of IoT networks and be deployable for IoT scenarios with varying consistency requirements.

Requirements	
#	Short description
I)	Configurable redundancy to achieve certain k-resilience against (multiple) node failures
II)	Handling of volatile networks where nodes join and leave the topology arbitrarily at runtime
III)	Fully distributed and decentralized decision-making concerning storage locations for dynamic replica placement
IV)	Off-path replica discovery algorithms especially to maintain system operation during node failures
V)	Configurable consistency mechanism to reach satisfying degree of consistency for varying application domains
VI)	Per content item selective content replication to bypass replication of unsuitable data
VII)	Handling of continuous data as it's produced in IoT scenarios a lot
VIII)	Shared storage per node to gain transparency and replica unawareness for services and applications
IX)	Low traffic coherency mechanisms to guarantee data freshness and keep response latencies low
X)	Lightweight and fast replacement algorithms to ensure useful utilization of storage resources
XI)	Lowering the overall bandwidth usage to increase system performance in terms of response times and avoiding bottlenecks
XII)	HOP reduction via shorter request paths through replica placement nearer to requesting nodes
XIII)	Configurable storage size to support differently powerful nodes and adapt to varying scenarios
XIV)	Consideration of constrained devices in terms of storage capacity, energy and computational power
XV)	Maintaining of the security and privacy of a system

Table 2.1: List of gathered requirements

Chapter 3

Related Work

In this section several existing approaches for data replication in the context of ICN and IoT are described and compared against the previously gathered requirements. Some of these approaches satisfy more or less of these requirements and the advantages and disadvantages of the related works are highlighted.

3.1 Cache Freshness in Named Data Networking for the Internet of Things

The authors of [42] proposed a freshness-aware caching strategy for IoT networks based on the NDN architecture. NDN as an instance of ICN offers all the ICN-based benefits and is believed to be a very useful base for IoT systems. Two superior goals are stated as there are at first the reduction of caching costs while maintaining the system's performance regarding hop reduction and server hit reduction and second the enhancement of the percentage of freshness of requested content. These first goal is realised with an ICN like in-network caching strategy that favours caching at gateways near content consumers and the second goal is tackled by a cache freshness mechanism specifically tailored for IoT traffic patterns.

Meddeb et al. identified two different physical IoT models which describe how sensors and actuators are connected in an IoT environment. In the first one sensors and actuators are directly connected with each other and no intermediate node is necessary. The storage of measured sensor data is located directly at respective sensors and therefore relies on sensors with operating system, software applications and memory. With caching done at a sensor some constrained devices will be put to their limits pretty fast

as their little computational power is consumed by decision-making (cache replacement) and their storage is quickly flooded by storing the measured values. In the second scenario the sensors are connected to nearby gateway nodes which are more powerful and the storage of measured values is done at these gateways as well as the decision-making. This facilitates the handling of constrained devices enormously which is the reason why the latter IoT architecture is supposed as base for the replication approach in [42].

3.1.1 Consumer-cache Strategy

The first contribution is the consumer-cache as it's called by its developers. An on-path caching strategy is realised because it is believed that the distribution of cache nodes across the network is more beneficial in IoT context than caching at predetermined and fixed cache nodes. Such an off-path strategy might put enormous loads on certain cache nodes and additional network traffic is generated for caching data compared to on-path caching. The proposed strategy targets the reduction of overall cache nodes in the network because caching in fewer nodes can yield improved results concerning producer hits and hop reduction for content requests. LCE as an opposing example fills caches too quickly because in general cache sizes in IoT environments are limited and many evictions are the result, often of data that hasn't even been used once before deletion and thus leading to increased cache costs. A reduced number of cache nodes should all in all save resources but it cannot be said that less cache nodes are better in general. There is always a trade-off between the number of evictions, cache cost and data availability. Reducing the number of cache nodes increases the importance of the decisions for cache placement. It is shown that edge-caching makes best use of in-network caching benefits by storing the cached content near edges of network topologies. This originates from hierarchical topologies where caches are located at the leaves of a network graph and traditionally there are also consumers located most of the time. Keeping this in mind the placement decision is based on a connection to consuming nodes in the network.

Finally, one can shortly summarize that the consumer-cache strategy stores a copy of requested content at nodes on the reversed request path which have a connection to the consuming nodes of that content. If every node is connected to a consumer node the strategy degenerates to LCE and if consumers are only located at the leaves of the topology the proposed strategy becomes a normal edge-caching. Therefore, the proposed strategy is highly dependent on the number of consumers in a network and their location.

3.1.2 Event-based Freshness Mechanism

The second part of the replication algorithm handles the coherence problem of the caching with a mechanism that ensures the content freshness in a event-based manner. First of all, two general IoT traffic patterns are distinguished whereas transmissions appear either periodically or in an OnOff way. The fist relates to sensors that update their current value after a certain time elapsed. Continuous transmissions are also seen as periodic with really small time steps. On the other side OnOff transmission occur only when a sensor updates a value but without fixed time interval. These transmissions are triggered by certain physical events.

The proposed event-based freshness mechanism is based on expiration of cached values with dynamic expiration dates. How the expiration time is set depends on the type of data producer. For periodic content the valid time is defined by the period between the generation of two consecutive versions of a value. In the case of content shared via OnOff transmissions the past events concerning that content are crucial. Additionally, in both cases the cache time is recorded to have information about how long a content item has already been in the cache.

The freshness algorithm is triggered when a request reaches a potential cache node in the network. First it is locally checked if a cached version of the requested data is available. If so there are two different cases for periodic and OnOff content. If the content is generated periodically the lifetime of the latest version in the source is computed with the residual life paradox which states that the residual life of an exponentially distributed variable is also exponential with the same rate. It is supposed that the periodicity for each content is known at all cache nodes. Now the computed lifetime is compared to the time the content is already in the cache (calculated as current time minus cache time). If the lifetime is greater than the time the content is in the cache the request is satisfied with the cached item. If it is the other way round the item is deleted from the cache as it has expired and the request is forwarded to the next node on the way to the source. If the transmission type of a requested content is OnOff the lifetime at the source isn't useful as there is not a defined period or anything. Therefore, a prediction of how long the content should be valid is made based on previous values. Every producer stores a list of how much time elapsed between two consecutive events. When a request reaches the source the prediction of how long the value should be valid is made and appended to the response as meta data value called event time. Now all cached versions of a OnOff content also have this event time which is checked on future requests. So, if a copy is found in the cache the event time is compared to the current time and if the event time is greater than the current time a request can be satisfied from the cache. In the other case the content is deleted from the cache. The prediction process at the

source is based on Autoregressive Moving Average (ARMA) time series. The choice was made as ARMA only takes the last k data values into consideration whereas exponential smoothing for example needs all past values and in the IoT context it is preferable to limit the input data somehow to reduce computational overhead.

3.1.3 Evaluation

The evaluation of the proposed caching algorithm is based on a NDN network simulation with the ccnSim simulator [13]. As already mentioned in section 2.2.1 every simulated node has a pending interest table (PIT) and a content store (CS) and Interests are forwarded on the shortest path. The simulated NDN network is set up as a Transit-Stub topology [11] with a total of 260 nodes as shown generic in figure [transit-stub]. The network is structured in a way that producers and consumers commonly do not belong to the same transit domain. It is assumed that in IoT scenarios contents have close request probabilities and therefore interests are distributed uniformly across the network. A total of 4000 sensors each producing exactly one content item is connected to 40 gateways and the number of consumers ranges between 20 to 30. The cache size was generally set to four pieces of content and as cache replacement algorithm LRU was used. Different caching strategies (LCE, LCD, ProbCache, Betweenness, edge-caching) have been deployed and then compared to the consumer-cache approach. For each strategy the hop reduction ratio, the server hit reduction ration and the response latency have been used as measurement metrics. The hop reduction ratio tells how many hops have to be traversed to satisfy a request compared to the number of hops to get the response directly from the producer. The server hit reduction ratio is measure per consumer and compares the number of satisfied requests only from the producers to the overall number of satisfied requests (from producers and caches).

The simulation results showed that the proposed consumer-cache strategy consistently performs best but only with a small improvement compared to edge-caching. The achieved server hit reduction was at 89% and also the hop reduction ratio was at 89%. This means only 11% of hops needed to be traversed compared to a network without any caching and also only 11% of requests have to be satisfied directly from the producer. This significantly shortens request paths and network load which also results in the shortest average response time compared to the other caching strategies.

The proposed freshness mechanism was compared to a simple static expiration-based freshness mechanism and to a network without freshness check. For different caching strategies the content validity without freshness mechanism is again best for the proposed consumer-cache, as this strategy reduces the content evictions and therefore avoids removing of still valid content in a cache. But without the additional freshness check

the validity reached only about 20% for consumer-cache and much worse for other caching strategies. A static expiration-based freshness resulted in about 40% validity for consumer-cache and nearly as good results for edge-caching while other strategies again have much worse results (under 10%). The proposed event-based freshness mechanism reaches about 98% validity for all caching strategies and therefore performs the best by far.

3.1.4 Comparison Against Requirements

The consumer-cache strategy seems superior compared to more traditional caching strategies but the performance improvement is not the only goal a replication strategy can and should bring along. Replication adds redundancy by its definition and therefore somehow improves the resilience and data availability but the proposed algorithm does not propose a mechanism to add a specifically defined resilience to tackle the danger of node failure. Only data that is requested gets replicated and the number of replicas cannot be influenced except via the content requests. With the cache placement decision only one hop from consumers the algorithm needs no prior knowledge of the network topology and implements a dynamic replica placement. However, the network didn't change during the tests and no information is given about how the algorithm handles new or removed nodes during runtime. As there is no additional content discovery algorithm data can only be served from nodes on the request path. This can lead to problems if the data source is temporarily not available and a new node requests it without having a cached version on the path. For efficient deployment of caching in systems where data changes frequently a certain degree of consistency needs to be guaranteed to ensure the usefulness of the replicated content. The proposed approach yields very good consistency for IoT traffic that is generated periodically with fixed time intervals but not so good consistency for OnOff-based data. The prediction of the expiration time with ARMA time series relies on the assumption that data is always generated as it was in the past and no unpredictable changes appear. Not every piece of data in IoT networks qualifies for replication and caching and the proposed mechanism lacks a possibility to decide what data should be replicated and cached and what data shouldn't. Instead all data that is requested gets cached which might flood caches with unimportant content. Many sensors in IoT environments produce live data that can change at every moment and most of the time only the most recent value is of interest. The consumer cache handles these continuous data generators as periodic transmissions with tiny time steps and according small TTLs. The consumer cache strategy is developed for ICNs and therefore natively supports a transparent replication and cache for applications and services. Multiple services on a single node share the same cache and aren't even aware that they get cached data which facilitates service and application

development. The required coherency level differs from scenario to scenario depending on the deployment domain. Unfortunately, the coherency in the proposed strategy isn't adjustable and depends only on the data type. For periodic data the coherency is strong with no extra message overhead and for OnOff data the coherency can't be guaranteed. Due to the massive amount of data and the high number of transmissions caches get populated quite fast and the usage of simple LRU as replacement performs good and doesn't add computational complexity. The replica placement near consumers reduces the overall traversed hops for requests drastically. The influence of the cache size on the proposed replication scheme is not examined as the test set up was fixed to a cache size of four pieces of data at all nodes. Constrained devices are not a problem with this strategy as every sensor in the network is directly connected to a more powerful gateway node that handles computational issues and storage. No statements about security and privacy were made.

3.2 A Periodic Caching Strategy Solution for the Smart City in Information-Centric Internet of Things

In [44] a periodic caching strategy (PCS) is introduced to enhance content dissemination in Information Centric IoT (IC-IoT) environments by improving the reach of desired information to the end user. The proposed cache deployment strategy targets a high cache hit rate and a reduced retrieval latency. Additionally, the distance between source and destination should be minimized in terms of stretch.

This approach is designed for operation in NDN environments just like the previous consumer-cache approach.

3.2.1 Proposed Strategy

In the proposed PCS every node in the network has a unique statistics table which keeps track of all interests reaching or traversing the respective node. With these statistics tables nodes are able to determine the most popular data in terms of content name, frequency count, recency of access and a threshold. For each piece of content the number of associated interests is gathered at every network node locally. The threshold is determined by the maximum value of incoming interests and is managed by the strategy algorithm which predicts most frequently requested contents. Certain content is recommended for caching if its respective interest numbers reach the threshold. These pieces of data are now marked for caching and are actually replicated to edge nodes of the autonomous systems. In this scenario the overall system is composed of several smaller so-called autonomous systems. Edge routers are the ones that connect two

or more different autonomous systems. When the caches of edge routers become full, content needs to be evicted. In this case the content that should be removed from the edge router is moved to another node inside the same autonomous system. The node is chosen based on betweenness centrality which means the node with the most interfaces to other nodes inside the specific subsystem takes the roll of a second level cache. Incoming requests are routed through the betweenness centrality node and if the requested data can be served from its cache this is the case. Only if the local lookup fails the request is forwarded to the edge or further to the publisher of the content. The content eviction follows the simple LRU replacement strategy. The cache-to-hit ratio is improved because most of the requests can be satisfied by the betweenness central router. Furthermore, the content is generally cached in the subsystems where it is requested meaning near to the consumer which decreases path lengths and thus shortens content retrieval latency. Memory is used efficiently as there is only one copy of the same data inside a autonomous subsystem and redundant replicas inside a single subsystem are avoided. This also prevents bandwidth congestions as homogeneous content replications are decreased.

3.2.2 Evaluation

The performance evaluation of the proposed caching strategy is based on a simulation with the SocialCCNSim simulator [17]. The simulation time was one day and the chunk size of content items was 1 KB. The content traffic was taken from Facebook and a 0.75-Zipf-like popularity model has been used. The simulation has been repeated multiple times for Abilene and tree topologies and also for cache sizes of 100 and 1000 chunks. The total size of content items was 10^6. The proposed PCS strategy was compared to a Tag-Based Caching Strategy (TCS)[63] and a Client-Cache Strategy (CC)[18] in terms of cache-to-hit ratio, content retrieval latency and stretch.

Regarding the cache-to-hit ratio PCS, TCS and CC lie close together with 8%, 7% and 5% for a cache size of 100 and an Abilene topology. The same cache size results in 8%, 6% and 6% for the tree topology. With a cache size of 1000 the ratio rises significantly to 68%, 61% and 56% for the Abilene topology and respectively 56%, 49% and 45% for the tree topology. The results show that the proposed strategy is superior to different degrees for larger and smaller cache sizes and for different topologies. The strategy works best with bigger cache sizes on Abilene topologies where it has a 7% higher cache-to-hit ratio than TCS and even 12% more compared to CC.

The evaluation of the content retrieval latency yields a 6% shorter response time for PCS compared to TCS and 8% shorter response time compared to CC for a cache size of 100 in Abilene topologies. With a cache size of 1000 PCS outperforms TCS by 9%

and CC by 12%. For tree topologies the result for the simulation with cache size 100 results in a 4% improvement of PCS compared to both, TCS and CC. Increasing the cache size to 1000 leaves PCS with a 11% and 18% improvement compared to TCS and CC. In the end PCS is superior to TCS and CC for Abilene and tree topologies and also for different cache sizes.

The third evaluation criterion is stretch and indicates how many hops less are needed for content retrieval than from the original source. For the Abilene topology and a cache size of 100 PCS reaches 42% whereas TCS has 46% and CC performs worst with 48%. This means PCS improves the distance to requested content in average about 58% compared to a system without any caching but only brings a slight improvement over TCS and CC. By increasing the cache size to 1000 TCS and CC perform quite analogous to the smaller cache size but PCS performs 11% better than TCS and 16% better than CC. For the tree topology PCS performs 3% better than TCS and 5% better than CC for a cache size of 100. For the bigger cache of size 1000 PCS outperforms the other strategies by 11% and 20%. This results in PCS having shorter average paths for all scenarios.

In this work a new caching strategy for IoT scenarios was introduced and tested against two other strategies (TCS, CC). All three algorithms were tested in an extensive simulation in terms of cache-to-hit ratio, content retrieval latency and stretch. In the end the proposed PCS strategy performed better in all three evaluation criterion under different circumstances as they were different topologies (Abilene and tree) and different cache sizes.

3.2.3 Comparison Against Requirements

Obviously, the Periodic Caching Strategy performs quite good regarding the three evaluation metrics. On the other side a replication mechanism for IoT environments can or should bring more than only improved hit ratio, stretch and latency. Therefore, in this section the proposed strategy is compared against the gathered requirements from the analysis.

PCS has no mechanism to add intended redundancy to achieve a defined resilience in the case of node failure. The number of replicas in the network depends on the number of sub networks (autonomous systems) and the popularity of content and can't be influenced otherwise. The registration of new nodes in the network can't be done autonomously without further actions as the betweenness centrality has to be calculated new every time a node joins a subnet and maybe the central node has to be adjusted and also edge routers may require reconfiguration. The cache or replica placement is dynamic to a

certain degree as the centrality node is not completely fixed in an autonomous network. On the other side caching only happens in edge or centrality routers which limits the dynamic placement. The scenario of node failure is not covered at all and therefore a replica discovery mechanism is absent. Also, the consistency of the cached content is not addressed what forces the employment of third-party consistency mechanism to ensure the usefulness of the replicated data. As not every information in an IoT network might benefit from replication, selective content replication would be a desirable feature. In the case of PCS only the popularity of content is responsible for the decision whether to cache a piece of data or not. As certain devices in IoT environments produce information continuously the replication mechanism should be able to handle such data sources. In PCS it is very unlikely that continuous data gets cached as the decision only depends on popularity and continuously generated data is mostly only relevant for a very short time. As the proposed strategy depends on NDN and utilizes its in-network caching, the whole process is transparent for applications or services and multiple of those access the same cache. Ideally the coherency of data in IoT scenarios is configurable as there are countless different use cases for IoT environments in diverse application domains. The PCS approach does not address the coherency issue at all. The cache replacement deploys a simple, fast and lightweight LRU algorithm to avoid unnecessary computational overhead during the cache replacement process. The bandwidth usage of the system is reduced during caching by avoiding too many replicas of the same data in a single subnet. Furthermore, the centrality-based caching inside a subnet leads to a significant hop reduction for requested content as shown in the evaluation of the work. As the simulation showed the PCS performs good for different cache sizes and therefore supports caches of varying sizes. Constrained devices are not explicitly considered but as the caching only happens in edge routers and centrality routers one can assume that these nodes are commonly more powerful and not the weakest machines in the network regarding memory, computational power and energy. Security and privacy issues are explicitly shifted towards the NDN architecture which guarantees security by design and makes the deployment of additional security mechanisms needless.

3.3 Storage Replication in Information-Centric Networking

The authors of [24] propose an algorithm for efficient storage placement and replica assignment in an information centric way. Before the description of the algorithm four architectural invariants for an ICN are described to ensure the correct deployment of the proposed replication mechanism.

The first invariant is just about ICN-like labelling of data items with a unique name. The second one concerns scoping in the network which leads to an expansion of the unique names where the scopes are added before the actual content name similar to folder paths. So, items can have the same name but they are still uniquely identifiable by their corresponding scope. The third invariant is called the service model and states that the dissemination of information is separated in three parts which are rendezvous, topology management and forwarding. Furthermore, the whole model is based on a pure pub/sub communication. Rendezvous match publishers of information with interested subscribers to gather certain location information about matched publishers and subscribers. The topology management finds efficient topologies to transfer the requested information and at last the forwarding actually transfers the requested content.

3.3.1 Proposed Strategy

The authors of the proposed replication mechanism use the so-called greedy algorithm as most efficient one for placement decisions referring to [54]. It uses a distance metric and the request load to determine the best location for storing a replica in the network. The greedy algorithm works round-based where every node in the network is considered one after the other. For each node the traffic gain is computed which is a value that states how much less traffic is generated if a replica is stored in this node.

There is a total of T different information items in the network and every node can hold SC of these objects. Each object t should be replicated exact k_t times and therefore a part M of all nodes is selected as cache nodes. The proposed algorithm is divided into two phases, planning and assignment. In the planning phase the algorithm selects M out of the N network nodes which are most suitable for caching and in the assignment phase each object t is assigned at k_t of the M storage nodes in a way that minimizes the overall network traffic. With a closer look the whole algorithm can be split into four steps. First the greedy algorithm is executed for all information items in the network resulting in a list of suitable storage nodes for each item s_t ordered by significance. These lists of storage nodes are then weighted based on the request rate for the respective item at that specific node or more intuitively spoken they are weighted with the respective popularity of an information item. There are T weighted lists of nodes now and from these the M nodes are selected that appear inside the most of these T lists. This results in a list S with the overall most suitable cache nodes of the network. At last for every information item t starting with the most popular one its corresponding list s_t is checked. A node of s_t gets assigned a replica of t if the same node also occurs in the list S, this node has capacity left and t was assigned at less than k_t nodes.

For a successful operation the storage planning and the replica assignment phase are deployed by functional components that are not directly part of the information network and run off-line at long-medium time scales. The storage planning component needs the number of desired storage nodes as an input and needs also knowledge of the network topology and a long-term prediction of subscriptions. Changes of storage nodes is not easy and the planning algorithm is only meant to run once in a long-term period and not frequently while the system is in operation. The replica assignment component runs at medium-term scale and is able to work automatically. As input it relies on the result from the storage planning component, knowledge about the network topology and the medium subscription forecast. Storage nodes act as publisher and subscriber at the same time. On the one side they subscribe to the original information publishers to keep track of changes and on the other side they deliver requested cached data to other requesting nodes based on the selected policy (e.g. requests are satisfied from the nearest replica storage). The request forecasts are generated by a subscription forecast component that observes rendezvous points.

3.3.2 Evaluation

For the evaluation a test scenario with 10 nodes was installed and emulated on top of a Gigabit Ethernet LAN. Subscribers were only located at the edge nodes of the network. Only four information items with a size of 10MB each were located in the network. A fixed number of storage nodes M was predefined and every object t is replicated at least once. Popularity and locality of information items are calculated using a Zipf law. During the evaluation the proposed ICN strategy is compared to a scenario where content is delivered towards subscribers just via IP uni casts. The main metric is the overall network traffic which is measured with respect to varying number of storages, storage capacities and clients (in this case the clients are the subscribers). The storages are already filled with content according to the proposed algorithm and the test therefore measures the traffic while satisfying requests but the dissemination of replicas is not part of the testing.

The results show that an increasing number of dedicated storage nodes in the ICN scenario only leads to a very small reduction of network traffic while the IP uni cast scenario benefits heavily from additional storages. Nearly the same holds for the storage capacity. The ICN architecture benefits only slightly from bigger storage capacities while the IP case profits significantly from bigger storages. But in the end the ICN-based test results are significantly better for both, number of storages and their respective capacity.

Additionally, the proposed system was simulated in MatLab to verify the scalability of the approach concerning a bigger and varying number of nodes with also varying storage

capacities. Here the proposed replica placement was compared to a random placement in an ICN environment. The number of nodes started at 50 and was increased up to 200. Of course, the network traffic constantly rises with the number of nodes but the proposed algorithm yields a flatter graph than the random variant and with the rising number of nodes the gap between random and proposed algorithm grows in favour of the proposed strategy. The varying storage capacity between two chunks of data up to 8 chunks shows that in general a bigger capacity reduces the network traffic. At first the gain is quite high and with more and more capacity the network traffic gain gets smaller and smaller until asymptotic behaviour.

3.3.3 Comparison Against Requirements

The conducted evaluation shows that the proposed replication strategy in an ICN architecture brings remarkable improvements compared to IP-based information dissemination and also significant gains compared to random ICN content dissemination in terms of network traffic. Nevertheless, in the analysis of this thesis a lot more desirable requirements for an ICN-IoT network have been identified. First of all, the proposed replication algorithm brings the ability to specify the desired number of replicas of information items which enables predefined redundancy and therefore adding the feature of k-resilience. Unfortunately, this approach is not suitable for very volatile networks where nodes join and leave the network on the fly because of its periodical planning phase. Concrete this means for the optimal performance the planning phase had to be executed every time a node joins or leaves the network and due to the long or medium-term periods this is not possible. The replica placement at least is dynamically to a certain degree. The storage nodes are periodically predefined but among the multiple storages the decision is made at runtime which leads to a partly dynamical replica placement. The replica discovery works via the pre-calculated rendezvous points which hold a lot of forwarding information for incoming requests. In the case of a node failure there is no recovery mechanism mentioned that is responsible for the discovery of alternative replicas in the network. The consistency is not a problem in the proposed replication approach as the authors suppose the information to be static. In the test case and simulation all information items were already in the network and no new ones are produced. New information can only be taken into consideration during the next planning phase. The decision what data to replicate and cache and what not can somehow be made in a selective way with by applying the required value for k_t but it is not clearly specified how and where this value is defined. Possibly that happens initially while creation of a specific piece of content. As the system can handle new data only effectively after the next planning phase continuously generated content can't be processed reasonably. As all ICN architectures the proposed system has transparent

caches which are shared by multiple applications and services on a single network node. The processing of static data makes a coherence or freshness mechanism unnecessary. Stale data is handled during the next planning phase. The same accounts for cache replacement. Between two planning phases there is no change of data in a cache node and the following planning phase calculates the placement entirely new without information of the previous content distribution. The minimization of network traffic is the main feature of the proposed algorithm and works quite well due to static information and the pre-calculated placement during the planning phase. With the reduction of network traffic comes the hop reduction for content requests. The storage nodes are predefined based on information popularity and are chosen for maximal traffic gains. This leads to relatively central storage placement near locations where most requests are expected. The cache size doesn't have to be fixed and changes to storage nodes can be done at any planning phase and constrained devices are taken into account at least concerning the storage capacity. During the assignment phase the remaining capacity is checked before a replica is propagated to the cache node. If the capacity is full the next most suitable node becomes the cache node for this specific piece of content. At last the security and privacy are not addressed in the proposed work but for ICN scenarios it can be assumed that the privacy comes by design with the ICN architecture.

3.4 Data dissemination scheme for distributed storage for IoT observation systems at large scale

In [28] a replication mechanism is proposed to increase the resilience and storage capacity of an IoT-based surveillance system to achieve better system robustness in case of node failures and local memory shortages. The main focus of the proposed work lies on wireless sensor networks (WSN) that sense and store data which is then periodically collected by a sink node. The authors of [28] suggest a low complexity distributed data replication algorithm for best results concerning network storage capacity and system robustness in cases of multiple node failure.

3.4.1 Proposed Strategy

It is assumed that sensors in the network gather data endlessly in a periodic manner defined by a certain sensing rate. For support of limited memories the sensed data is periodically collected from the sink and cancelled from the sensor memory. To prevent data loss due to node failure the sensor nodes cooperate by replicating their sensed data to other near nodes with spare memory. The source node can hold a copy of the sensed data but mustn't necessarily. Updates with the current memory availability are

periodically broadcast to all direct neighbours from each node. This requires every node to have and maintain a memory table of all known neighbours.

The proposed algorithm is greedy which means when data should be replicated the best neighbour is chosen based on the memory table. This neighbour becomes now a donor node as it donates some of its memory to its producer neighbour. If no donor node can be found and the local memory is full the newly generated data is deleted. The size of the neighbour memory table is fixed and if there are too many direct neighbouring nodes only the best ones (regarding available storage capacity) are kept in the list. Assuming R copies of a data item should be achieved the first step is checking the local storage. If possible the generated item is stored locally and R is reduced by 1. Now the top node of the memory table is selected to store the next replica. This continues until R replicas (including the original one) are stored among nodes. If no or not enough suitable donor nodes are available the final number of replicas is less than R. The replication process works recursively which allows nodes to become donor nodes even if they are not direct neighbours from the initially generating node. The donors are chosen by a specific heuristic rule which takes the memory table into account and for second level or more donor nodes this heuristic rule is modified slightly. The recursion continues until all desired replicas are stored or until an intermediate not can't find a suitable neighbour which leads to less than R replicas. Already traversed nodes are not considered for the replica forwarding to avoid circles.

3.4.2 Evaluation

The deployment of replication mechanisms inside distributed systems leads to a trade-off between effectively usable storage capacity and reliability of the system against failure and data loss. Therefore, the evaluation of the proposed replication scheme inspects different metrics. The first one is the network storage capacity which represents the amount of unique data that can be stored in the whole system. The second metric is the time until the storage capacity is reached for the whole network. This means how long does it take until the memory of every single node is exhausted. As another metric the dropped data is used meaning how much data is lost due to local memory shortage. And the last metric is the system robustness in terms of how much data can be recovered after a bomb-like failure scenario.

The network storage capacity maximizes if no replicas are stored besides the original content item (R=1). For increased R the maximum capacity can be lower bounded by dividing the previous maximum capacity by the maximum number of replicas for a single unique data item. The lower bound can actually only be reached for very large local storages or quite low sensing rates that are much lower than the data retrieval rate from

the sink node. The time until data dropping occurs is the time it takes until the first node in the network completely saturates its local storage. For this a MatLab simulation was created with 10 nodes, a buffer size of 250 data units per node and a uniformly distributed sensing rate between 1 and 10 data units per second. The whole network capacity (2500 data units) was reached after 120 seconds without replication (R=1) but the first data drop occurred after 25 seconds. With an optimal data distribution strategy regarding the node storages and still without replication the first data drops occur after 40 seconds. The capacity is also checked for R=3 and R=5. In these cases the whole storage capacity is reached after about 13 seconds and 6 seconds. The data drops start just after the maximum capacity is reached. To avoid data loss in practical scenarios the retrieval from the sink node has to be adjusted to a period that is smaller than the time until the first data loss.

The robustness of the system was tested with a bomb-like scenario where nodes inside a spatially limited area failed. These test where conducted two times each for different numbers of replicas (3, 5 and 7). In the first run the failure radius was quite small only affecting direct neighbours of the failure centre and in the second run the radius was expanded to also affect two or three hop neighbours. Of course, the robustness was significantly better in the cases with a smaller impact radius. The highest robustness was achieved for R=5 and only little worse for R=3 while for R=7 the robustness was significantly lower. The reason is that for too many replicas per data unit the storage capacity of the system is reached much faster and at that point the replicas can't properly spread through the network which results effectively in less than 7 replicas to the point where in average not even 3 or 5 replicas can be distributed.

All in all, the evaluation carried out in [28] takes several different scenarios into consideration but unfortunately the proposed replication mechanism is not compared to any other type of caching or replication scheme.

3.4.3 Comparison Against Requirements

Even if there is no information about how the proposed replication mechanism performs compared to other strategies it can nevertheless be compared to the gathered requirements from section 2.6.

Whit its focus on robustness and node failure this approach offers the possibility for defined redundancy by adjusting R which results in the desired k-resilience. A little cutback is that for insufficient storage capacities the desired number of replicas might not be satisfied. Dynamically added or removed nodes in the network can easily be taken into account because the neighbour memory tables are always updated when

a change occurs. The placement process for replicas is completely dynamic as the memory tables are evaluated at runtime and no node is chosen in prior. Obviously, there exists a replica discovery algorithm because otherwise the robustness tests can't be executed but it is not stated how the discovery is done. In the experiment it is only important that at least one copy of a data unit is still inside any working node of the network. In the proposed scenario the data items do not change which makes the original and its respective replicas somehow always consistent. Information is only created periodically like specified via the sensing interval and then collected from a sink node. It is not clearly depicted if the number of replicas can be set for individual producers. If the value of R can be set per sensor a selective content replication would be possible but the described simulation seems to only pick R as a whole for the system which does not support selective replication. Continuous data can somehow be processed with small sensing intervals but as in this case only new values are gathered and not an existing one is changed this requirement is not matching for the proposed system. The storage of replicas is not restricted to any application or service and every node in the network is potentially able to store other replicas which means the proposed strategy is transparent. As the gathered data does not change but only new data is produced there is no coherence mechanism because it is not needed in the proposed scenario. Furthermore, there is no replacement algorithm. Excess data is either dropped or to avoid data loss the retrieval rate of the sink node has to be sufficient for the deployed storage capacity. The bandwidth usage of the proposed replication algorithm is quite low and doesn't add a lot of traffic as only small and minimal communication is required for replication. Theoretically the request paths are shortened which results in a hop reduction but in reality the data is not requested from the nodes by services or applications. All the further processing happens at the sink and what comes after that so there is no real hop reduction and request path shortening. The cache or replica storage sizes are configurable and one can even say they are nearly arbitrary because the free storage is taken into account at replica assignment during runtime and if a neighbour has too little space left another one is selected. Of course, in practical scenarios the storage capacity has certain bounds to ensure useful behaviour of the system. Also, constrained devices are considered with the dynamic storage-based replication and further devices with different sensing intervals and transmit powers can be deployed. Security and privacy issues are not mentioned at all in the proposed replication strategy.

3.5 A Decentralized Replica Placement Algorithm for Edge Computing

The authors of [4] propose an algorithm called distributed replica placement (D-ReP) algorithm. The main purpose of the algorithm is the placement of replicas in a network with the goal of minimizing the overall cost for satisfying content requests. The algorithm needs to work distributed and fully decentralized due to the high number of nodes in edge computing scenarios. A central node gathering topological and demand information would not scale well for scenarios with many participating network nodes. Furthermore, the communication for finding suitable replica locations should be kept at a minimum. As edge computing environments tend to be very dynamic in terms of spontaneously joining or leaving nodes in the network the targeted algorithm needs to be highly dynamic and on-line to react accordingly to varying demands and popularity of content.

Every node has only knowledge about itself and its direct neighbours. As input a node needs information about the number of requests for each local replica, the neighbour that forwarded a request to it, the latency to the afore mentioned neighbour and the storage price of each neighbour. This information can be gathered with standard networking tools like routing tables. The consistency of replicas is explicitly ignored and either left for an independent consistency algorithm or just read-only data is processed.

3.5.1 Proposed Strategy

The D-ReP algorithm pushes replicas from producer nodes towards requesting consumer nodes on the edges of the network. This leads to a migration of content into directions of the respective consumers. When the demand falls or changes replicas can be dropped or move into other directions of the network.

The proposed D-ReP algorithm runs in two different versions depending on the node where the specific implementation runs. The source version runs on producer nodes and can only create replicas of content items at direct neighbour nodes. The edge version of the algorithm runs at nodes that hold at least one replication of a content item at the edges of the network. It is able to duplicate a replica, migrate it to its neighbours, remove it or do nothing at all. If no replica is left at an edge node the edge version of the algorithm can turn itself off. Different instances of the algorithm do not communicate with each other (regardless of source or edge version) to stay fully decentralized. For all operations the expected cost is compared to the expected benefit and only if the benefit surpasses the cost the operation is executed. The analytical formulas describing costs are precisely defined in [4] and are based on latencies, number of requests per

item, size and unit price. Replicas can be dropped if demand changes in a node either due to creation of other replicas or just due to changes in the general demand. In this case a copy is removed to avoid unnecessary storage costs. For replica replacement a threshold-based weight calculation is utilized to remove a replica if its demand during a certain time was under the dynamic threshold. The threshold is calculated as proportion of expected replica utilization (created at replica generation).

For the discovery of replicas an algorithm is proposed to complement D-ReP. The discovery algorithm enables in case of a content request the discovery of near replicas that are not on the direct path to the source node. No broadcast messages or central control entities are needed to maintain the decentralized nature of D-ReP. Nodes are notified about near replicas only if it this node is expected to request the replicated data in the near future. Therefore, each node has a known replica locations (KRL) table. This table holds the replica id (what data it stores), the replica node (location of the replica) and latency to this node. Multiple replicas of the same data can occur in these tables. Every time a replica is migrated or duplicated to a new neighbour the replica receiving node notifies all nodes that have accessed the replica data on the previous node during a recent time interval. The information about which nodes have accessed the replica is sent with the actual replica to the new replica storage node. The node holding the new replica now notifies all direct neighbours about the new replica so that these can update their KRL table if necessary.

3.5.2 Evaluation

For evaluation of the D-ReP algorithm the CAIDA Anonymized Internet Traces 2015 Dataset [68] was used in a simulation based on the CloudSim [10] framework. Besides the proposed D-ReP algorithm a common caching system and a central storage (without any replication) system are simulated against which the proposed D-ReP is compared. For the central storage the network node with the highest closeness centrality in the test set-up is chosen. In the caching system replicas are directly created at the requesting nodes and can only be used for that specific node exclusively. The cache size at each node is limited and in the case of full capacities LRU is used for content replacement. The simulation variables are epoch length (1 to 20 minutes), λ (level of replica expansion, 0.01 to 0.20) and cache capacity (10 to 200).

The variation of λ has only influence on the results for D-ReP as the parameter is used in the algorithm's cost function. In the test scenario the latency improvement for common caching was constant at about 25% whereas the improvement with D-ReP rises constantly with a higher value of λ. With values greater than 0.16 D-ReP surpasses the common caching in terms of latency improvement. In general, D-ReP has even better

results for longer epoch lengths. A 10-minute epoch length yields constantly around 3% better latencies than a shorter 3-minute epoch length. The other way was also tested where the epoch length was varied and λ was set to 0.10 and 0.16. In this case the common caching is also constant at 25% latency improvement as it is independent from λ. D-ReP improves with longer epochs until about 10 minutes and then converges. For λ=0.10 the improvement stays under the caching approach but for λ=0.16 D-ReP reaches up to 33% latency improvement.

Concerning the additional cost evaluation the same test parameters were applied. Again, the common caching was constant at about 15% additional cost compared to no replication. For varying λ the replication costs constantly increased with a higher value of λ. For values greater than 0.16 the cost of D-ReP surpassed caching with 19% additional costs in the case of 10-minute epochs. However, for λ smaller or equal to 0.15 the costs are lower than for caching. The 3-minute epoch leads to a maximum of 11% additional costs for D-ReP even for high values of λ and is therefore better than common caching in all scenarios. When the epoch length is variable caching stays at constant 15% additional cost whereas D-ReP rises nearly linearly with increasing epochs. For λ greater or equal to 0.16 the the additional costs surpass caching at 6-minute epochs. For smaller values of λ (0.10) the additional costs stay under that of caching even for 20-minute epoch length.

The third metric measured was the benefit-cost ratio (BCR) which was also determined for varying λ and epoch length and additional with varying cache capacities (number of replicas). The benefit-cost ratio is the latency improvement divided through the additional cost and is used to couple the previous results and outlines the trade-off between latency improvement and additional replica cost. The higher the BCR the better. For varying λ caching is constant at a BCR of around 1.9. D-ReP's BCR falls with higher values of λ but converges against 1.9 for 10-minute epoch length. For smaller values than 0.16 the 10 minutes epoch D-ReP is still better than caching. Shorter epochs of 3 minutes yield a constantly higher BCR which converges against 2.8 for high λ values. For varying epoch lengths the BCR of D-ReP with 0.16 λ falls under caching for epochs with 10 minutes or more. With 0.10 as value for λ the BCR of D-ReP meets caching at 20-minute epochs. This means the smaller λ is and the shorter the epoch length is the more efficient is the proposed algorithm. Further the measurements of the BCR were done for varying cache capacities which means the D-ReP values were constant while the caching BCR fell quite rapidly for increasing number of replicas. D-ReP was compared with kind of worst and best case configuration. The lower BCR at 1.8 was measured for D-ReP with epoch length of 10 minutes and λ=0.16 and a better BCR of 2.9 was achieved for D-ReP with 3-minute epochs and λ=0.10. For a cache capacity

of 10 replicas the caching started with a very good BCR of 3.0 but for a capacity of 20 replicas the BCR of common caching dropped to nearly 2.0 and for capacity of 30 caching was nearly as good as the worse D-ReP configuration. The BCR constantly falls for greater cache capacity and is for a capacity of 100 already below a BCR of 1.0.

Compared to a basic cache approach and a central storage scenario without replication the proposed D-ReP algorithm can achieve great performance gains with appropriately configured parameters.

3.5.3 Comparison Against Requirements

The proposed algorithm implements a configurable replication strategy that can be adjusted to fit different scenarios in which basic caching mechanisms and central storage approaches can be outperformed. However, there are more desired requirements than improvement of request latency. B-ReP does not support a defined number of replicas and therefore can't offer k-resilience. Replicas are created only due to requests and an analysis of the expected cost and benefit of such a new replica. Volatile networks with frequently joining and leaving nodes can be handled by the proposed replication strategy because of its fully distributed and decentralized architecture. At no phase of the algorithm a central entity is involved and all information is gathered individually at every node and only takes its direct neighbours into consideration. The replica placement is initially triggered by content requests which makes it very dynamic. Content migrates towards locations where it is more popular and the proposed algorithm even reacts to changes in the demand. The complementary replica discovery algorithm ensures that in case of node failure replicas are found fast and at near nodes. The algorithm doesn't only work in failure scenarios but even reduces network traffic by satisfying requests from near replicas that are not directly on the path to the original content producer. In the proposed test scenario for D-ReP the authors assumed just read-only data and stated that for write data an independent consistency algorithm had to be deployed. Thus, the proposed system does not natively support replica consistency. The decision what content should be replicated is based on the requests for this specific content. Only if there is enough interest for certain data items a replication can occur. But the demand in a node alone is not enough as then the algorithm starts its work and decides if a replica improves the overall latency for requests. With this mechanism there is no way of selecting certain contents to be replicated (only very vaguely via influencing the requests accordingly). The origin and type of data used for D-ReP is not specified (except read-only) in the proposal paper. Therefore, no statement about the algorithms ability to handle continuously generated data can be made. Replicas on a node are available for all services and applications running on that node and the

requesting process doesn't need knowledge about being served from a replica as read-only data is exclusively used. This makes the proposed strategy a transparent replication approach. Coherency is not addressed at all. The replacement of replicas is handled with a utility threshold and not only triggered when the storage capacity is full and a new replica should be stored. Continuously a node checks if its locally stored replicas are still utilized to a certain degree (defined by the dynamic threshold). If that is not the case replicas are dropped. This adds a bit more computational complexity than for example simple LRU but useless data gets removed from the storage faster. The overall used bandwidth can be reduced employing D-ReP. This is indirectly shown by the latency reduction during the evaluation and testing. Replicas are only generated when the system as a whole benefits from it in terms of latency and therefore also network traffic and bandwidth usage. Request- and cost-based replication moves content replicas towards popular locations which reduces the average number of hops that are required to satisfy a request. The size of single content items are considered for replica migration but the size of the storages per node are not directly mentioned. The authors expected sufficiently sized storages. Constrained devices were not considered in terms of memory or other properties. The same accounts for security and privacy issues.

3.6 Summary and Comparison

In this section the previously introduced strategies are compared against each other and the requirements gathered in section 2.6. If each of the proposed mechanisms satisfies the desired requirements or not can be seen in table 3.1. For better readability the respective name of the works were shortened and are identified by their section number. Also, the explanation of the requirements in the table was shortened for layout reasons. For more detailed information about what is meant with each requirement have a look at table 2.1. A + indicates that the affected requirement is satisfied by a related work. The degree to which this is the case can be taken from the respective previous comparison against requirements section. This means a + is present if the requirement is fulfilled at least to an extent that allows consideration of the technique for this thesis. On the other side a − indicates that the specified requirement is not satisfied or that it is only covered partly in a way that makes its implementation unsuitable for further consideration in this thesis. Also, if a proposed related work strategy doesn't touch a requirement in any kind a − is present.

Only two of the five observed strategies offer the possibility of a defined redundancy what enables k-resilience. K-resilience is a feature that is ultimately desired for critical systems in different application domains to ensure the system operation during node

failures (at least for a certain time). Also, a volatile network topology is only supported by two out of five mechanisms. This requirement is especially useful for IoT environments as these are extended frequently during runtime with new sensors or actuators or old ones are changed. Supporting these changes in topology leads to a decentralized and distributed behaviour which favours the scalability of a system. The fact that all introduced replication strategies support dynamic replica or cache placement (some more and some partly) indicates that dynamic placement is widely accepted as efficient and best practice in IoT and ICN scenarios. Four approaches miss a proper replica discovery mechanism which can lead to problems in the case of node failures. But also, the request response time can greatly benefit from off-path replica discovery. The influence of off-path replica discovery on the performance but also on the reliability of a distributed system can make it a very beneficial feature. The importance of consistency of data in a distributed system depends on the application domain but in general consistency should not be neglected. Some of the introduced replication approaches just rely on static data but this is not realistic for IoT scenarios. Ideally the consistency should be configurable to adapt to different scenarios but for most scenarios a certain degree of consistency is required. In this respect it would seem the thing to directly include consistency mechanisms to distributed replication schemes. The question what content should be replicated is difficult but reasonable as it doesn't make sense to replicate every type of content. In most four of the introduced related works such a selection can't be done as either all content gets replicated or the decision is based on the popularity of the content. This is not bad per se and can even lead to performance gains but especially in the case of failure scenarios there might be some very important data that is not very popular. Such data is not replicated by popularity-based decisions and could be protected from node failure when per data selective caching/replication would be a thing. The handling of continuously produced data is similar to the selective replication. Only one introduced work really handles continuous data. In IoT scenarios there are many use cases where certain values change very rapidly and frequently and only the current value is important. An IoT replication mechanism should be able to handle such data by either excluding it from replication or by other appropriate processing. No problem is the transparency of the replication for services and applications. All introduced strategies replicate on the network layer and offer a shared cache or storage for all applications and services on the respective network node. Moreover the content requesters do not know that they have been served from a replicated version and it shouldn't matter. Here the proper degree of consistency needs to be ensured. Coherency is also a point that was not well represented by the mentioned related works with one exception. Ensuring the coherency of data always is a trade-off between usefulness of data and the network load. But in fact, a low network load is nothing when

the processed data is not fresh enough. A coherency mechanism should be part of a replication system but it should not add more communication overhead than necessary. Replica replacement is necessary for solutions with limited storages. Three strategies brought simple replacement algorithms to keep the computational overhead low as desired. The strategies without replacement algorithms considered static data in their tests and took storage capacities into account for replica placement. For most IoT scenarios the information inside the network is very volatile and changes frequently which leads to continuous changes in the replica storages/caches which can be handled most efficiently with replacement algorithms. All proposed mechanisms reduce the overall network traffic. This requirement is crucial as the adding of replication may not affect the performance in a negative way (or at least not too much). Much more the replication mostly benefits to the system performance by lowering the average bandwidth consumption through hop reduction for content requests. Configurable storage sizes are supported by three replication strategies and can help to maximize the benefit of the replication. The utility of such mechanisms varies depending on different parameters including the storage size. Also, a per node configurable or varying storage size helps with the deployment of heterogeneous devices which is important for IoT environments. Constrained devices are handled by some of the introduced strategies in different ways but in general the heterogeneity of IoT environments concerning the used hardware is a good reason to keep constrained devices in mind. At last security and privacy are not addressed directly by any of the related works but some inherently implement the security features of the base architecture they are built on. This is true especially for ICN-based approaches.

All in all, the five presented strategies have important and interesting features and implement more or less of the identified requirements. Nevertheless, none if it meets all desired features, mostly because the focus is either set on improving the system performance or on improving the reliability of a system. In this thesis a strategy should be outlined that merges these two high-level requirements and focus on them equally. A mechanism that is developed for both should be designed from root in a way that performance and reliability work hand in hand to maximize the benefits for both goals. The extensive literature research showed that such a system is not realized yet in the IoT context.

		RW 3.1	RW 3.2	RW 3.3	RW 3.4	RW 3.5
I)	Defined resilience	−	−	+	+	−
II)	Volatile network topology	−	−	−	+	+
III)	Dynamic replica placement	+	+	+	+	+
IV)	Replica discovery	−	−	−	−	+
V)	Satisfying consistency	+	−	+	+	−
VI)	Selective content replication	−	−	+	−	−
VII)	Continuous data	+	−	−	−	−
VIII)	Transparency for services	+	+	+	+	+
IX)	Low traffic coherency	+	−	−	−	−
X)	Lightweight replacement	+	+	−	−	+
XI)	Low bandwidth usage	+	+	+	+	+
XII)	HOP reduction	+	+	+	−	+
XIII)	Configurable storage size	−	+	+	+	−
XIV)	Constrained devices	+	+	−	+	−
XV)	Security/privacy	+	+	+	−	−

Table 3.1: Comparison of related works to gathered requirements

Chapter 4

Design

In this chapter the previously analysed bricks are put together to find a solution for efficient replication in distributed IoT environments. In the end there should be a concept that satisfies all the gathered requirements by reusing parts of existing solutions that already meet a fraction of requirements and by improving and adding pieces to fill the remaining holes.

As shown in section 2.2 ICN is an architectural concept that brings a lot of helpful and facilitating features for IoT solutions by its design. The most desirable ICN feature is the focus on data instead of network addresses and the in-network caching that is enabled through this. The benefits of ICN for IoT are significant and lead to the decision to design an Information Centric replication approach for IoT networks.

In fact, the proposed replication mechanism consists of two parts that are tightly coupled. The one part is responsible for the performance goal and the other part focuses on the resilience. Certain requirements have to be taken into account at both sides and are kind of omnipresent. Most of all this accounts for the assumption that distributed IoT environments are very volatile and the topology and participating nodes change quite frequently and arbitrarily. This means all deployed bricks need to maintain fully distributed and decentralized properties.

4.1 Performance Goal

The system performance can greatly benefit from proper deployment of caching strategies.

4.1.1 Cache Localization

For improving the performance in terms of shorter response times, less network traffic, hop reduction etc. an on-path caching approach is chosen. Off-path caching either requires a central control entity which is against the targeted decentralized architecture or extensive previous planning which contradicts the dynamic topologies and reduces scalability. With this in mind on-path caching brings the required features for distributed and decentralized caching. The exact cache placement can be done in different ways but to keep the complexity low strategies like leave copy down or move copy down seem appropriate. In this way content moves toward the locations it is most requested. LCE would add too much network traffic and therefore spam the network with copies in nodes where they are never requested. Probabilistic caching could also be implemented but this is more unpredictable and the probabilities must somehow be generated which might add additional complexity. Betweenness centrality approaches require complete knowledge of the network topology which again is not conform with fully decentralized behaviour. As already mentioned earlier a configurable approach is desirable and in this case this means MCD and LCD should be implemented. Network operators should then have the possibility to switch between these cache localization mechanisms according to the application domain.

4.1.2 Cache Replacement

Concerning the cache replacement a simple approach is favoured to keep the computational complexity low. Function-based replacement algorithms need relatively much input that has to be gathered over time and this can take some time. This is a disadvantage in respect to the volatile IoT topology assumption. Randomized strategies are too unpredictable which leaves recency- and frequency-based approaches. These can both be maintained with very little effort and again the implementation of both variants seems the best solution. The network operator can then configure which replacement algorithm to use based on the specific use case. As default replacement a hybrid of recency- and frequency-based mechanisms is promising. Frequency alone as decision metric can keep outdated data in the cache for long when it was heavily requested in the past and recency only metrics might drop replicas that are heavily requested over a longer time span in the event of multiple other one-time requests. Frequency-based approaches also need an additional tie-breaker. This leaves the hybrid implementations as the most promising approach. A lightweight candidate is the in section 2.4.2 already introduced LRU*.

4.1.3 Cache Coherence

The importance of the coherency of cached content varies from deployment domain to deployment domain. Some systems generated solely read-only data that does not expire. In such cases a coherency algorithm is not necessary at all. But other systems produce great amounts of data on an unpredictable basis and it might be crucial to always get the newest data. The dependence of the concrete IoT use case makes it impossible to have a single solution that fits all scenarios. Because of this a configurable approach is the only reliable answer. Validation checks add quite a lot of communication overhead but guarantee the freshest data whereas expiration-based mechanisms can go without any additional communication overhead but there is no guarantee that the requested data has the most recent value. Expiration-based TTLs work very well if data is not altered too frequently or only at fixed time intervals. With the implementation of both, expiration and validation mechanisms, all cases can be handled. The both algorithms ideally run parallel on every node and a service or application decides if coherency is of crucial interest or not. This means a service that requests certain data sends that request and along with it if the response must be the freshest version or if maybe a little outdated data is also acceptable. When the request arrives at a cache node the respective coherency mechanism is triggered. The TTLs on the other side get applied at content creation at the producer and are transferred on top of the distributed content for the on-path caches. Call-back mechanisms are avoided because every producer needs knowledge about all existing replicas of its data and this is also difficult in dynamic and volatile networks, especially when content also changes frequently.

4.1.4 Cache Size

The cache size is a parameter that should be configurable due to many reasons. First of all, different cache sizes can influence the performance of a caching system. Caches with too much capacity can suffer a performance decrease due to high search times inside the huge crowded caches. Too small capacities on the other side lead to an increase in replacement operations what also slows the system and furthermore the cache hit ratio can drop drastically for very small cache sizes. Again, it depends heavily on the concrete IoT topology. Therefore, the cache size should be adjustable at every network node to react to undesired behaviour by just increasing or decreasing the available cache size. In practical scenarios it would take a little testing to find the proper capacity value. Varying cache sizes per node also mean that different devices can be deployed including memory constrained devices. These just have a smaller cache than.

4.2 Resilience Goal

The resilience of a distributed IoT system shall be improved to achieve a feature called k-resilience where at least k+1 replicas of data are distributed in the network. This guarantees that even in the case of simultaneous failure of k nodes no data loss happens. The analysis showed that P2P systems have a lot in common with IoT networks which qualifies P2P-replication approaches as a good starting point for IoT-based replication. The DBMS approaches focus more on complete replication of databases and aren't as dynamic as desired for IoT scenarios. In comparison to the caching where the creation of cachable replicas is triggered from the requesting side the resilience-based replicas should be pushed whenever content is updated at the producer to ensure the desired degree of resilience at any time.

4.2.1 General Replica Decisions

There are some general issues that can be taken from P2P replication. The first is the question where data can be altered. Is only the original copy at the producer writeable or can every replica in the network be changed? The latter is referred to as multi-master replication. If any copy can be changed the problem is how to update all other copies in the network. As the desired replication approach should be full decentralized only flooding could be used to reach all replicas from an arbitrary node. Flooding puts a lot of communication overhead on the network which reduces the system performance. The other approach was introduced as single-master replication and only allows changes at the content origin. Regarding IoT scenarios one needs to consider that almost all produced data is generated from sensors and these data items are mainly consumed. This means requesting services hardly change data at other nodes but only read that data and process it locally. With this in mind the single-master approach seems more suitable as the producer is the only node that can change its own data and it is (mostly) also the only one that wants to change it. If the other case occurs the remote node needs to process the write to the producer who then executes the operation. The single-master approach has the danger of a single point of failure but in reality as already stated the only updates usually come from the producer node and if it fails there are probably no more updates which negates the danger of single point of failure.

Another decision has to be made between full or partial replication. The first one means that every node has a replica of every content item in the network. This seems very inappropriate for IoT scenarios as there is a lot of data and every piece of content is only needed at certain nodes. Regarding the frequent changes of data in IoT scenarios an enormous amount of communication overhead would be put on the network by

replicating everything to every node. This leads to the decision for partial replication allowing to define what should be replicated and what shouldn't.

4.2.2 Replica Placement

The replica placement could be done the same way as the cache placement but for performance reasons the caching is based on content requests and therefore on popularity. As this section handles the resilience motivated replication popularity doesn't need to be taken into consideration. Also, the placement decision is heavily restricted by the condition that the whole replication should be decentralized and fully distributed. A seemingly good possibility is to only take the direct neighbours into the process. Every node therefore needs a memory table where the free replica storage of its neighbours is maintained. From this table the neighbours with most free replica storage capacity are chosen to store a replica. The replica storage size should be dedicated just like the cache size but each on its own to avoid negative interaction between the request-based caches and the pushed replicas. The producer also has a table in which it stores a list of neighbours that got a replica. With this the replicas can directly be updated. If a node has not enough neighbours (or several neighbours without enough remaining storage capacity) the first neighbour in the list also distributes replicas to its neighbours. When now the source updates replicas a node that forwarded replicas knows this and also forwards the updated data.

4.2.3 Selective Replication

In the caching mechanism the decision what to cache is simply done request-based. But in the case of the resilience scenario the replicas get pushed to neighbouring nodes. How does the producer know how many replicas should be distributed? The selection of content is done by the producer itself. This means when an application or service is created the developer can specify the number of desired replicas to achieve k-resilience. If data is not suitable for replication it can just be tagged to distribute zero replicas. Furthermore, the number of replicas should also be configurable with different granularities. A service can tag its produced data on item level, but an administrator should have the possibility to set the k for the respective resilience node-wide. In this case every produced data item on this node gets replicated the same times. And at last the configuration should be available system wide.

4.2.4 Replica Discovery

Now in the case of a failure scenario a node needs to know where replicas are. In cases of proper operation a node requests certain data and the request gets forwarded to the data source. On the way at every node the cache is checked for the requested item.

Here the cache and also the replica storage can be checked to increase the hit ratio. In the best case the request can be satisfied as fast as possible by a nearer node than the original producer. But if that's not the case and the original producer is offline, the question is how a node anywhere in the network should find the replicas. This is where replica discovery is triggered. There are two general possibilities called blind search techniques and informed search techniques. The informed search techniques rely on more or less information about the topology and further forwarding information. These algorithms need additional storage for the required information and put extra computational overhead on a node. The blind search techniques do not rely on additional data but create on average more communication overhead which influences the system performance. In this case the focus still lies on resilience which makes a little drawback in performance seem quite acceptable. Especially as the discovery algorithm is only triggered in the case of a node failure. Assuming that node failures are quite rare the blind search is preferred as it doesn't load burdens on the system during normal operation. Now different search techniques can be implemented and chosen based on the node configuration similar as for the caching bricks. The success of the search algorithms is increased as not only replica storages are checked but also caches. The replicas are spatially relatively close to the producers but the cached versions spread further through the network which facilitates the replica discovery in a failure scenario.

4.3 Comparison Against Requirements

In this section the proposed replication strategy is shortly compared to the requirements from section 2.1.

K-resilience is achieved through tagged data or via node- or system-wide configuration of the replica numbers. The algorithm ensures that there are always at least $k+1$ copies of the respective data item in the network. If one adds the cached copies there might be even a lot more replicas of the same data. This is due to the separation of replication regarding resilience and performance. But this can't be seen as wasted memory due to too many redundant data items because the cached versions are located at nodes where the overall network traffic benefits from it. The other replicas can be seen as the last safety instance to maintain operation during node failures. The proposed replication strategy is designed for volatile topologies like IoT networks are by their nature. All the combined mechanisms and algorithms maintain the fully distributed and decentralized behaviour as initially demanded including the cache and replica placement. An additional flooding-based discovery mechanism ensures that in case of a node failure the requested data can be found in one of the caches or replica storages. The consistency

is guaranteed as data can only be changed by the original producer in a single-master way. The resilience-based replication can be done selective on data item level, node level or system level. The same process can also be used by the caching algorithm if certain data should explicitly not considered for caching. Continuous data can be handled in different ways. If the specified data does definitely not qualify for caching or replication (maybe because of very short update intervals) the data can just be tagged for exclusion regarding caching and replication. If the data generation is quite regular or even based on precise time intervals a simple TTL-based on the sensing interval can be applied. The underlying ICN architecture natively supports transparency of the caches for applications and services and furthermore, requesting services do not even now if they have been served from a cache or not. If the respective service really relies on fresh data it can be configured at service creation to utilize the validation check coherency method. This leads to the coherency requirement. By default, TTLs are used to ensure a certain level of coherency but for critical services the validation check method can be executed. This leads to low additional networking traffic as mostly the TTLs are used but keeps the possibility for strong coherency if necessary. The cache replacement is configurable and offers several low complexity algorithms to keep the performance up. Through request-based cache placement the overall bandwidth usage is reduced by reducing the averagely traversed hops for content requests. The possible configuration of the placement algorithm offers adaptability to different network topologies. Storage sizes of replica storage and caches can be adjusted per node separately to also support different topologies and scales of IoT environments. Constrained devices are no problem because all the storage and processing are done by gateway nodes that are commonly powerful enough. All constrained devices are directly attached to a gateway that takes most of the burden of the devices like sensors and actuators. At last the security and privacy was not addressed explicitly because ICN architectures like the VSL implement security by design and the replication process takes place completely inside the already secured network. No possible attack points are offered in the form of interfaces to external components or similar.

Chapter 5

Implementation Outlook

A suitable framework for a prototype implementation of the previously designed replication approach was already presented in section 2.3. The VSL is the base and enables the composition of information centric IoT networks. Currently the VSL supports only single hop connections which means every node in the peer-to-peer overlay has a direct connection to every other node. In the future the VSL architecture will be extended to support multi-hop connections which enables greater and more complex IoT environments. The VSL is implemented in Java to guarantee compatibility with all operating systems and devices. This means a prototype implementation of the replication algorithm can also be written in Java by extending the source code of the Knowledge Agents (KA).

Chapter 6

Evaluation Outlook

After the implementation of a functioning prototype a test case needs to be built. Ideally the scenario could be ran in the iLab environment of the chair for networking at the TUM. In this case real physical devices are involved and the results are not based on latency and throughput assumptions as it is the case in pure simulations. For the performance tests a meaningful metric would be the average response time for a request. This has to be tested several times with different configurations for example of the placement and replacement algorithms. If possible, it would also be interesting to build varying network topologies to examine the influence of different algorithms on different topologies. Of course, a reference test needs to be conducted for every test set up. In the reference tests the replication approach is disabled and all requests are served from the original producer. With these results the different configurations of the caching strategy can be compared.

For evaluation of the resilience failure scenarios have to be set up. This could be done by taking nodes down incrementally for different resilience values. Interesting would also be the correlation between performance and resilience. In this respect the performance measurements can be evaluated during node failures to detect if the system is still reasonably efficient even if the replica discovery is triggered.

Chapter 7

Conclusion

In this thesis a lot of bricks used for replication have been examined in detail with their various advantages and disadvantages. Based on this analysis several requirements for an efficient replication mechanism in the context of IoT environments have been identified. An extensive literature research led to the result that there are currently no replication solutions for IoT scenarios that pay equally respect to improving the resilience and boosting the system performance. The analysis of different related works in similar fields emphasized the lack of such a solution. This work gives a rough guiding towards such a resilience and performance aware approach and can be used as foundation for further research and development in that direction.

Chapter 8

Outlook

The replication in distributed IoT environments might benefit significantly by extending the approach with predictive caching mechanisms that might have the potential to increase the cache-hit-ratio by fetching the right content but with machine learning strategies also the replica/cache placement and TTLs could be applied near an optimal strategy to additionally increase the benefits of replication for IoT scenarios. Nevertheless, this needs a lot of research and might be a medium-term challenge. Also, the performance evaluation regarding the caching would be more informative in a multi-hop scenario. This can probably be done is not so far future as the VSL shall be extended to support multi-hop communication. Also, extensive test beds and simulations with different network topologies can then be applied to find suitable default properties for the configurable parts of the proposed approach.

Bibliography

[1] Mohamed Abomhara and Geir M Køien. "Security and privacy in the Internet of Things: Current status and open issues". In: *2014 international conference on privacy and security in mobile systems (PRISMS)*. IEEE. 2014, pp. 1–8.

[2] Rafi Ahmed et al. "Using an object model in Pegasus to integrate heterogeneous data". In: *Database Technology Department, Hewlett-Packard Laboratories, Palo Alto, CA* (1991).

[3] Amit Gud Masaaki Mizuno Daniel Andresen. "A Scalable Search Algorithm for Unstructured Peer-to-Peer Networks". In: ().

[4] Atakan Aral and Tolga Ovatman. "A decentralized replica placement algorithm for edge computing". In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 516–529.

[5] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey". In: *Computer networks* 54.15 (2010), pp. 2787–2805.

[6] Emmanuel Baccelli et al. "Information centric networking in the IoT: Experiments with NDN in the wild". In: *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ACM. 2014, pp. 77–86.

[7] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. "Concurrency control and recovery in database systems". In: (1987).

[8] Nabhendra Bisnik and Alhussein A Abouzeid. "Optimizing random walk search algorithms in P2P networks". In: *Computer networks* 51.6 (2007), pp. 1499–1514.

[9] Carsten Bormann, Mehmet Ersue, and Ari Keranen. *Terminology for constrained-node networks*. Tech. rep. 2014.

[10] Rodrigo N Calheiros et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In: *Software: Practice and experience* 41.1 (2011), pp. 23–50.

[11] Kenneth L Calvert, Matthew B Doar, and Ellen W Zegura. "Modeling internet topology". In: *IEEE Communications magazine* 35.6 (1997), pp. 160–163.

[12] Wei Koong Chai et al. “Cache “less for more” in information-centric networks”. In: *International Conference on Research in Networking*. Springer. 2012, pp. 27–40.

[13] Raffaele Chiocchetti, Dario Rossi, and Giuseppe Rossini. “ccnsim: An highly scalable ccn simulator”. In: *2013 IEEE International Conference on Communications (ICC)*. IEEE. 2013, pp. 2309–2314.

[14] World Wide Web Consortium et al. “W3c httpd”. In: *At URL http://www. w3. org/hypertext/WWW/Daemon/Status. html* ().

[15] correctly. “Internet of things: Vision, applications and research challenges”. PhD thesis. 2012.

[16] Cedric Coulon, Esther Pacitti, and Patrick Valduriez. “Consistency management for partial replication in a high performance database cluster”. In: *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*. Vol. 1. IEEE. 2005, pp. 809–815.

[17] Ikram Ud Din, Suhaidi Hassan, and Adib Habbal. “SocialCCNSim: A simulator for caching strategies in information-centric networking”. In: *Advanced Science Letters* 21.11 (2015), pp. 3505–3509.

[18] Ikram Ud Din et al. “Caching in information-centric networking: Strategies, challenges, and future research directions”. In: *IEEE Communications Surveys & Tutorials* 20.2 (2018), pp. 1443–1474.

[19] Simon Dobson et al. “A survey of autonomic communications”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 1.2 (2006), pp. 223–259.

[20] Reza Dorrigiv, Alejandro Lopez-Ortiz, and Pawel Pralat. “Search algorithms for unstructured peer-to-peer networks”. In: *32nd IEEE Conference on Local Computer Networks (LCN 2007)*. IEEE. 2007, pp. 343–352.

[21] Wilfried Elmenreich et al. “A survey of models and design methods for self-organizing networked systems”. In: *International Workshop on Self-Organizing Systems*. Springer. 2009, pp. 37–49.

[22] Seyed Kaveh Fayazbakhsh et al. “Less pain, most of the gain: Incrementally deployable icn”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. ACM. 2013, pp. 147–158.

[23] Roy Fielding et al. *Hypertext transfer protocol–HTTP/1.1*. Tech. rep. 1999.

[24] Paris Flegkas et al. “Storage replication in information-centric networking”. In: *2013 International Conference on Computing, Networking and Communications (ICNC)*. IEEE. 2013, pp. 850–855.

[25] George HL Fletcher, Hardik A Sheth, and Katy Börner. “Unstructured peer-to-peer networks: Topological properties and search performance”. In: *International Workshop on Agents and P2P Computing*. Springer. 2004, pp. 14–27.

[26] Ali Ghodsi et al. “Information-centric networking–Ready for the real world?(Dagstuhl Seminar 12361)”. In: *Dagstuhl Reports*. Vol. 2. 9. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2013.

[27] Sushant Goel and Rajkumar Buyya. “Data replication strategies in wide-area distributed systems”. In: *Enterprise service computing: from concept to deployment*. IGI Global, 2007, pp. 211–241.

[28] Pietro Gonizzi et al. “Data dissemination scheme for distributed storage for IoT observation systems at large scale”. In: *Information Fusion* 22 (2015), pp. 16–25.

[29] Jim Gray et al. “The dangers of replication and a solution”. In: *ACM SIGMOD Record* 25.2 (1996), pp. 173–182.

[30] Mohamed Ahmed Hail et al. “Caching in named data networking for the wireless internet of things”. In: *2015 international conference on recent advances in internet of things (RIoT)*. IEEE. 2015, pp. 1–6.

[31] Yong Ho Hwang. “Iot security & privacy: threats and challenges”. In: *Proceedings of the 1st ACM Workshop on IoT Privacy, Trust, and Security*. ACM. 2015, pp. 1–1.

[32] Isam Ishaq et al. “IETF standardization in the field of the internet of things (IoT): a survey”. In: *Journal of Sensor and Actuator Networks* 2.2 (2013), pp. 235–287.

[33] Van Jacobson et al. “Networking named content”. In: *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM. 2009, pp. 1–12.

[34] Imad Jawhar and Jie Wu. “A two-level random walk search protocol for peer-to-peer networks”. In: *Proc. of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics*. 2004, pp. 1–5.

[35] Michael Leon Kazar et al. *Synchronization and caching issues in the Andrew file system*. Carnegie Mellon University, Information Technology Center, 1988.

[36] James J Kistler and Mahadev Satyanarayanan. “Disconnected operation in the Coda file system”. In: *ACM Transactions on Computer Systems (TOCS)* 10.1 (1992), pp. 3–25.

[37] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. “The LCD interconnection of LRU caches and its analysis”. In: *Performance Evaluation* 63.7 (2006), pp. 609–634.

[38] Elias Leontiadis, Vassilios V Dimakopoulos, and Evaggelia Pitoura. “Creating and maintaining replicas in unstructured peer-to-peer systems”. In: *European Conference on Parallel Processing*. Springer. 2006, pp. 1015–1025.

[39] Xiuqi Li and Jie Wu. “Searching techniques in peer-to-peer networks”. In: *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks* (2006), pp. 613–642.

[40] Barbara Liskov et al. *Replication in the Harp file system*. Vol. 25. 5. ACM, 1991.

[41] Qin Lv et al. "Search and replication in unstructured peer-to-peer networks". In: *Proceedings of the 16th international conference on Supercomputing*. ACM. 2002, pp. 84–95.

[42] Maroua Meddeb et al. "Cache freshness in named data networking for the internet of things". In: *The Computer Journal* 61.10 (2018), pp. 1496–1511.

[43] Daniel A Menascé. "Scalable P2P search". In: *IEEE Internet Computing* 7.2 (2003), pp. 83–87.

[44] Muhammad Naeem et al. "A periodic caching strategy solution for the smart city in information-centric Internet of Things". In: *Sustainability* 10.7 (2018), p. 2576.

[45] Giwon On, Jens Schmitt, and Ralf Steinmetz. "The effectiveness of realistic replication strategies on quality of availability for peer-to-peer systems". In: *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*. IEEE. 2003, pp. 57–64.

[46] Esther Pacitti, Pascale Minet, and Eric Simon. "Fast algorithms for maintaining replica consistency in lazy master replicated databases". PhD thesis. INRIA, 1999.

[47] Marc-Oliver Pahl and Georg Carle. "Taking smart space users into the development loop: An architecture for community based software development for smart spaces". In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM. 2013, pp. 793–800.

[48] Marc-Oliver Pahl and Georg Carle. "The missing layer—Virtualizing smart spaces". In: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE. 2013, pp. 139–144.

[49] MO Pahl and S Liebald. "Designing a Data-Centric internet of things". In: *2019 International Conference on Networked Systems (NetSys)(Net-Sys' 19), Garching b. München, Germany*. 2019.

[50] Graham D Parrington et al. "The design and implementation of Arjuna". In: *Computing systems* 8.2 (1995), pp. 255–308.

[51] Stefan Podlipnig and Laszlo Böszörmenyi. "A survey of web cache replacement strategies". In: *ACM Computing Surveys (CSUR)* 35.4 (2003), pp. 374–398.

[52] Alok Prakash. "A Survey of Advanced Search in P2P Networks". In: *Department of Computer Science, Kent State University* (2006).

[53] Ioannis Psaras, Wei Koong Chai, and George Pavlou. "Probabilistic in-network caching for information-centric networks". In: *Proceedings of the second edition of the ICN workshop on Information-centric networking*. ACM. 2012, pp. 55–60.

[54] Lili Qiu, Venkata N Padmanabhan, and Geoffrey M Voelker. "On the placement of web server replicas". In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Com-*

puter and Communications Society (Cat. No. 01CH37213). Vol. 3. IEEE. 2001, pp. 1587–1596.

[55] José Quevedo, Daniel Corujo, and Rui Aguiar. "A case for ICN usage in IoT environments". In: *2014 IEEE Global Communications Conference*. IEEE. 2014, pp. 2770–2775.

[56] Lorenzo Saino, Ioannis Psaras, and George Pavlou. "Hash-routing schemes for information centric networking." In: *ICN*. ACM. 2013, pp. 27–32.

[57] Lorenzo Saino, Ioannis Psaras, and George Pavlou. "Icarus: a caching simulator for information centric networking (ICN)". In: *SIMUTools*. Vol. 7. ICST. 2014, pp. 66–75.

[58] Russel Sandberg et al. "Design and implementation of the Sun network filesystem". In: *Proceedings of the Summer USENIX conference*. 1985, pp. 119–130.

[59] Jeff Sidell et al. "Data replication in Mariposa". In: *icde*. IEEE. 1996, p. 485.

[60] Alex Siegel, Kenneth Birman, and Keith Marzullo. "Deceit: A flexible distributed file system". In: *[1990] Proceedings. Workshop on the Management of Replicated Data*. IEEE. 1990, pp. 15–17.

[61] Swaminathan Sivasubramanian et al. "Analysis of caching and replication strategies for web applications". In: *IEEE Internet Computing* 11.1 (2007), pp. 60–66.

[62] Yuning Song, Huadong Ma, and Liang Liu. "Content-centric internetworking for resource-constrained devices in the Internet of Things". In: *2013 IEEE International Conference on Communications (ICC)*. IEEE. 2013, pp. 1742–1747.

[63] Yuning Song, Huadong Ma, and Liang Liu. "TCCN: Tag-assisted content centric networking for Internet of Things". In: *2015 IEEE 16th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE. 2015, pp. 1–9.

[64] Evjola Spaho, Leonard Barolli, and Fatos Xhafa. "Data replication strategies in P2P systems: A survey". In: *2014 17th International Conference on Network-Based Information Systems*. IEEE. 2014, pp. 302–309.

[65] Sabu M Thampi and Ra Sekaran. "Collaborative loadbalancing scheme for improving search performance in unstructured p2p networks". In: *Proc. Of the 1 st Int. Conf. Contemporary Computing*. Citeseer. 2008.

[66] Sabu M Thampi et al. "Survey of search and replication schemes in unstructured p2p networks". In: *arXiv preprint arXiv:1008.1629* (2010).

[67] Dimitrios Tsoumakos and Nick Roussopoulos. "Adaptive probabilistic search for peer-to-peer networks". In: *Proceedings third international conference on peer-to-peer computing (P2P2003)*. IEEE. 2003, pp. 102–109.

[68] C Walsworth et al. *The CAIDA UCSD anonymized internet traces (2011)*. 2015.

[69] Beverly Yang and Hector Garcia-Molina. “Efficient search in peer-to-peer networks”. In: *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. 2002.

[70] Beverly Yang and Hector Garcia-Molina. “Improving search in peer-to-peer networks”. In: *Proceedings 22nd International Conference on Distributed Computing Systems*. IEEE. 2002, pp. 5–14.

[71] D Zeinalipour-Yazti, Vana Kalogeraki, and Dimitrios Gunopulos. “Information Retrieval in Peer-to-Peer Networks”. In: *International Conference on Computer, Information, and Systems Science, and Engineering CISE*. 2003.

[72] Guoqiang Zhang, Yang Li, and Tao Lin. “Caching in information centric networking: A survey”. In: *Computer Networks* 57.16 (2013), pp. 3128–3141.

[73] Haoxiang Zhang et al. “Probabilistic search in p2p networks with high node degree variation”. In: *2007 IEEE International Conference on Communications*. IEEE. 2007, pp. 1710–1715.

[74] Jiaying Zhang and Peter Honeyman. “A replicated file system for Grid computing”. In: *Concurrency and Computation: Practice and Experience* 20.9 (2008), pp. 1113–1130.

[75] Lixia Zhang et al. “Named data networking (ndn) project”. In: *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* 157 (2010), p. 158.

[76] Zhenyun Zhuang et al. “Hybrid periodical flooding in unstructured peer-to-peer networks”. In: *2003 International Conference on Parallel Processing, 2003. Proceedings.* IEEE. 2003, pp. 171–178.

[77] Hai Zhuge, Xue Chen, and Xiaoping Sun. “Preferential walk: towards efficient and scalable search in unstructured peer-to-peer networks”. In: *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM. 2005, pp. 882–883.